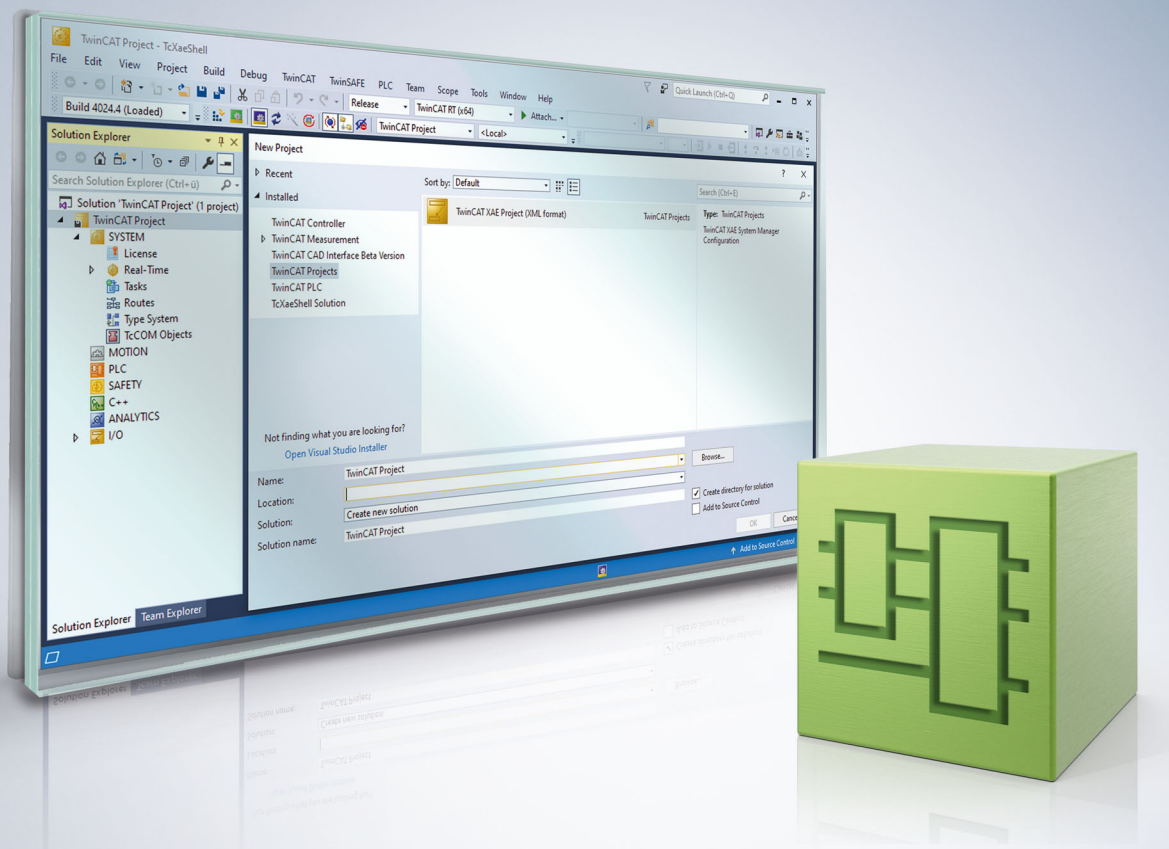
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC



07.02.2024 | Version: 3.0.1

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, stets die aktuell gültige Dokumentation zu verwenden.

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

Disclaimer

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

Marken

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

Patente

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

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

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

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

Haftungsausschluss

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

Qualifikation des Personals

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

Signalwörter

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

Warnungen vor Personenschäden

⚠ GEFAHR

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

⚠ WARNUNG

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

⚠ VORSICHT

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

Warnung vor Umwelt- oder Sachschäden

HINWEIS

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

Information zum Umgang mit dem Produkt

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

1.3 Hinweise zur Informationssicherheit

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

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

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

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

2 Quickstart

Systemüberblick

TwinCAT 3 PLC realisiert auf einer CPU eine oder mehrere SPS mit dem internationalen Standard IEC 61131-3 3rd Edition. Zur Programmierung können alle in der Norm beschriebenen Programmiersprachen verwendet werden. Die Bausteine vom Typ PROGRAM können mit Echtzeittasks verbunden werden. Verschiedene komfortable Debugging-Möglichkeiten erleichtern die Fehlersuche und Inbetriebnahme. Programmänderungen können zu beliebigen Zeiten und in beliebiger Größe online, d. h. bei laufender SPS, durchgeführt werden. Alle Variablen stehen per ADS symbolisch zur Verfügung und können in entsprechenden Clients gelesen und geschrieben werden.

Funktionalitäten

TwinCAT 3 PLC stellt Ihnen vielfältige und komfortable Engineering-Funktionen für Ihre Entwicklungsarbeit zur Verfügung:

	Siehe hierzu in dieser Online-Hilfe:
Projektkonfiguration durch Assistenten (Wizards)	Projekt anlegen und konfigurieren [▶ 55]
Anpassungsmöglichkeit der Oberfläche	Oberfläche anpassen
Erzeugung von professionellen Steuerungsprogrammen nach IEC 61131-3 mit vielen Standardfunktionen	SPS-Projekt programmieren [▶ 68]
Komfortable Programmierung mit Maus und Tastatur in allen Sprachen der IEC 61131-3 Entsprechende Editoren für FUP, KOP, AWL, ST, AS, dazu die Varianten CFC und Erweiterter CFC	Programmiersprachen und ihre Editoren [▶ 654]
Eingabeunterstützung für die Eingabe und Konfiguration unterschiedlichster Daten	Eingabeunterstützung nutzen [▶ 141]
Unterstützung objektorientierter Programmierung Echte objektorientierte Programmierung nach 61131-3 3rd Edition in allen Sprachen der IEC 61131-3 ohne zusätzliche Tools möglich Vererbung von Programmbausteinen auf ähnliche Programmteile zur Reduktion von Entwicklungszeit und Fehlern Objektorientierte Programmierung ist kein Muss: Funktionale oder objektorientierte Programmierung sind beliebig einsetzbar und mischbar	Objektorientiert programmieren [▶ 178]
Umfassender Projektvergleich, auch für grafische Editoren	Projekt vergleichen
Bibliothekskonzept zur einfachen Wiederverwendung von Programmcode	Bibliotheken verwenden [▶ 278]
Debugging- und Online-Eigenschaften zur Optimierung des Programmcodes und zur Beschleunigung von Test und Inbetriebnahme	Testen und Fehler beheben [▶ 222]
Integrierte Compiler für verschiedene CPU-Plattformen	Projekteigenschaften - Kategorie Übersetzen [▶ 951]
Security-Eigenschaften zur Absicherung des Quellcodes und des Betriebs der Steuerung	Projekt schützen und speichern Verschlüsseln des SPS-Projekts

Hilfe verwenden

Jede Hilfekomponente besteht aus einem Konzept- und einem Referenzteil. Im Konzeptteil werden detailliert alle Themen erklärt, die bei der Erstellung, Verwaltung und Ausführung eines TwinCAT-3-SPS-Projekts relevant sind. Ergänzt werden die Beschreibungen durch Handlungsanweisungen, die Sie schritt-für-schritt

zu einem gewünschten Ergebnis führen. Die Referenzteile sind vollständige Nachschlagewerke für die Benutzeroberfläche und die Programmierung der TwinCAT 3 PLC. Beachten Sie auch die Dokumentation „TC3 User Interface“.

2.1 Unterschiede zu TwinCAT 2

Bibiliotheksversionen

- Mehrere Versionen einer Bibliothek sind im gleichen Projekt möglich. Eindeutige Zugriffe erfolgen durch Angabe des Namensraums.
- Installation in Repositories (Datenbanken mit Bibliotheken in verschiedenen Versionen)
- Automatische Aktualisierung
- Debugging möglich
- Bibliotheksprofile und Platzhalter für erleichterte Versionskompatibilität

Kompatibilität mit Projekten in anderen Dateiformaten

- Projekte unterschiedlicher Formate können mit dem Befehl **Vorhandenes Element hinzufügen…** automatisch in das TwinCAT-3-Format konvertiert werden. Sie können definieren, wie dabei eingebundene Bibliotheken gehandhabt werden sollen.
- Ein Konverter für das TwinCAT-2-Format ist Teil des Standard-Programmiersystems. Wenn Sie ein TwinCAT-2-Projekt auch in TwinCAT 3 verwenden wollen, müssen Sie jedoch einige Voraussetzungen und Einschränkungen beachten.

Datentypen

Folgende Datentypen sind neu:

- any_type
- UNION
- LTIME
- BIT
- References
- Aufzählungen / Enumerationen: Basisdatentyp kann angegeben werden.
- `di : DINT := DINT#16#FFFFFFFF ;` nicht erlaubt

Editoren

ST-Editor:

- Klammerungen, Umbrüche, Code-Komplettierung, Inline-Monitoring, Inline-set/reset-Zuweisung

FUP/KOP/AWL-Editor:

- FUP, KOP und AWL sind ineinander konvertierbar und haben einen gemeinsamen Editor.
- AWL-Editor als Tabelleneditor
- Der Hauptausgang in Bausteinen mit mehrfachen Ausgängen kann gesetzt werden.
- Verzweigungen und „Netzwerke in Netzwerken“

AS-Editor:

- Nur ein Schritttyp, Makros, Mehrfachselektion von unabhängigen Elementen, keine Syntaxüberprüfung während des Editierens

Visualisierungskonzept

- Der Visualisierungseditor arbeitet standardmäßig mit einem Werkzeugkasten und einem Editor für die Elementeigenschaften zusammen.

- Teile der Visualisierungsfunktionalität sind gemäß IEC 61131 realisiert und werden somit über Bibliotheken bereitgestellt. Ein internes Laufzeitsystem führt die wichtigsten Visualisierungsfunktionen aus.
- Textlisten und Bildersammlungen werden für die Verwaltung von Texten und Bilddateien verwendet.
- Ein Visualisierungsmanager handhabt verschiedene Visualisierung-Clients (wie Web Client, Standalone Client...), die flexibel und mit geringem Aufwand für die unterschiedlichsten angepassten SPS-Projekte verwendet werden können.
- Visualisierungsprofile definieren, welche Bibliotheksversionen und welche Elemente aktuell verfügbar sind.
- Visualisierungsstile erlauben eine einfache Anpassung des „Look-and-Feel“ der Visualisierungen.

Operatoren und Variablen

- Neue Gültigkeitsbereich-Operatoren, erweiterte Namensräume
- Init-Methode ersetzt den INI-Operator
- Exit-Methode
- Ausgangsvariablen in Funktions- und Methoden-Aufrufen
- VAR_TEMP, VAR_STAT
- Beliebige Ausdrücke für Variablen-Initialisierung
- Zuweisung als Ausdruck
- Index-Zugriff mit Zeigern und Strings

Objektorientierung

- Erweiterungen für Funktionsblöcke (Funktionsbausteine): Eigenschaften, Schnittstellen, Methoden, Vererbung, Methodenaufruf

Weiteres

- Konfigurierbare Menüs, Werkzeugleiste und Tastaturbedienung
- Unicode-Unterstützung
- Einzeilige Kommentare: // Kommentar
- CONTINUE in Schleifen
- Bedingte Kompilierung
- Bedingte Haltepunkte
- Debuggen: Ausführen bis Cursor, Ausführen bis Rücksprung

Änderungen gegenüber TwinCAT 2 PLC:

- FUNCTIONBLOCK ist nicht länger ein gültiges Schlüsselwort anstelle von FUNCTION_BLOCK für Funktionsbausteine
- Nach TYPE (Deklaration einer Struktur) muss ein „:“ stehen.
- ARRAY-Initialisierung muss mit runden Klammern versehen sein.
- INI wird nicht mehr unterstützt und muss durch die Init-Methode ersetzt werden.
- In Funktionsaufrufen ist es nicht länger möglich, explizite Parameterzuweisungen mit impliziten zu mischen. Deshalb kann die Reihenfolge der Parametereingangszuweisungen verändert werden:

```
fun(formal1 := actual1, actual2); // → Fehlermeldung
fun(formal2 := actual2, formal1 := actual1); // gleiche Semantik wie ...
fun(formal1 := actual1, formal2 := actual2);
```

- Pragmas (Import von TwinCAT-2-Pragmas ist noch nicht implementiert.)
- Der TRUNC-Operator konvertiert nun in den Datentyp DINT anstelle von INT. Bei einem TwinCAT-2-Import wird automatisch eine entsprechende Typkonvertierung hinzugefügt.
- Direkte Adressierung von allokierten Variablen:

Die direkte Adressierung [▶ 790] von allokierten Variablen hat sich geändert. Während unter TwinCAT 2 unabhängig vom Datentyp immer von derselben Stelle ausgegangen worden ist, wird unter TwinCAT 3 unterschieden:

TwinCAT 2: W0 enthält B0 und B1, W1 enthält B1 und B2, W100 enthält B100 und B101

TwinCAT 3: W0 enthält B0 und B1, W1 enthält B2 und B3, W100 enthält B200 und B201

2.2 Ihr erstes TwinCAT-3-SPS-Projekt

Zum Inhalt Ihres ersten Projekts

In diesem Tutorial programmieren Sie eine einfache Kühlschranksteuerung.

- Wie bei einem handelsüblichen Kühlschrank wird die Solltemperatur über einen Drehregler vom Benutzer vorgegeben.
- Über einen Sensor ermittelt der Kühlschrank die Isttemperatur. Wenn diese zu hoch ist, startet der Kühlschrank mit einer einstellbaren Verzögerung den Kompressor.
- Der Kompressor kühlt, bis die eingestellte Solltemperatur abzüglich einer Hysterese von 1 Grad erreicht ist. Die Hysterese soll verhindern, dass die Isttemperatur zu sehr um die Solltemperatur schwingt und sich der Kompressor ständig ein- und ausschaltet.
- Wenn die Tür offensteht, leuchtet im Inneren des Kühlschranks eine Lampe.
- Steht die Tür zu lange offen, ertönt ein getaktetes akustisches Signal.
- Wenn der Kompressor die Solltemperatur trotz Aktivität des Motors über längere Zeit nicht erreicht, gibt der Pieper ein durchgehendes, akustisches Signal aus.

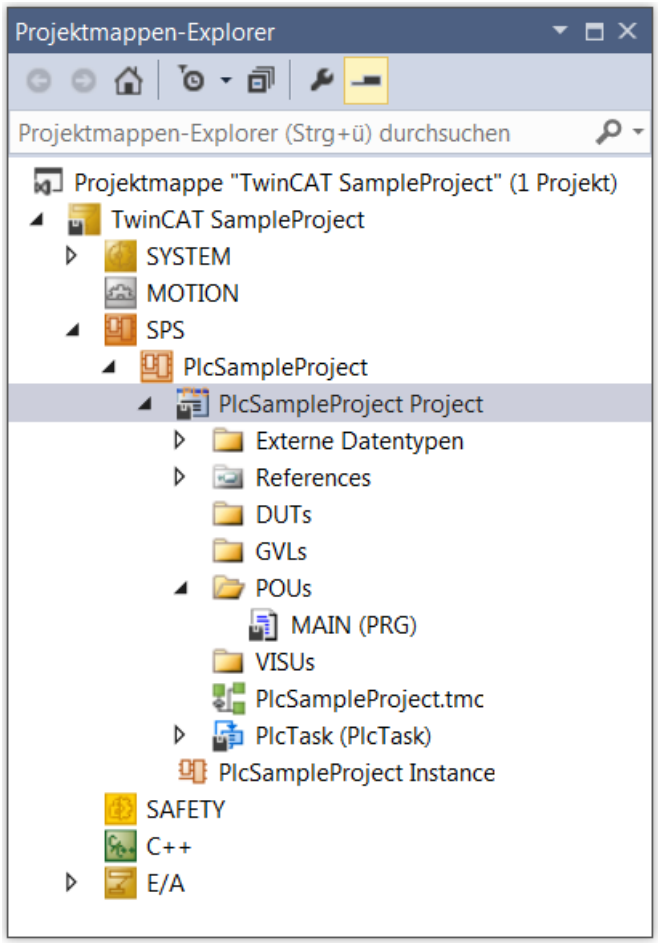
Projektierung:

Die Steuerung der Kühltätigkeit erfolgt im Hauptprogramm des SPS-Projekts, die Signalverwaltung in einem weiteren Programmbaustein. Die benötigten Standardfunktionsbausteine sind in der Bibliothek Tc2_Standard verfügbar. Da in diesem Beispielprojekt keine echten Temperatursensoren und keine echten Aktoren angeschlossen werden, schreiben Sie zusätzlich ein Programm zur Simulation von Temperaturanstieg und Temperatursenkung. Damit können Sie das Arbeiten der Kühlschranksteuerung im Onlinebetrieb beobachten. Variablen, die von allen Bausteinen verwendet werden sollen, definieren Sie in einer globalen Variablenliste.

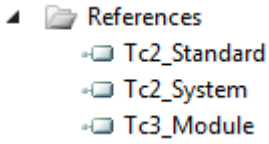
Anlegen des SPS-Projekts

1. Wählen Sie im Menü **Datei** den Befehl **Neu > Projekt**, um eine neue TwinCAT-Projektdatei anzulegen.
   ⇨ Eine neue Projektmappe mit dem TwinCAT-Projektbaum öffnet sich im **Projektmappen-Explorer**.
2. Klicken Sie mit rechts auf den Knoten **SPS** im **Projektmappen-Explorer** und wählen Sie den Befehl **Neues Element hinzufügen**, um dem TwinCAT-Projekt ein SPS-Projekt hinzuzufügen.
   ⇨ Der Dialog **Neues Element hinzufügen – TwinCAT <Projektname>** öffnet sich.
3. Selektieren Sie in der Kategorie **Plc Templates** die Vorlage **Standard PLC Project**.
4. Geben Sie einen Namen und einen Speicherort für das Projekt an und klicken Sie auf die Schaltfläche **Hinzufügen**.

⇨ Mit dem gewählten Template wird automatisch ein Programm MAIN angelegt, das von einer Task aufgerufen wird. Als Programmiersprache wird automatisch „Strukturierter Text (ST)“ ausgewählt.



Unter References ist automatisch der Bibliotheksverwalter mit einigen wichtigen Standardbibliotheken angewählt. Die Bibliothek Tc2_Standard enthält alle Funktionen und Funktionsbausteine, die von der Norm IEC 61131-3 beschrieben werden.



Deklarieren der globalen Variablen

Deklarieren Sie zunächst die Variablen, die Sie im gesamten SPS-Projekt verwenden wollen. Dazu legen Sie eine globale Variablenliste an:

1. Selektieren Sie den Unterordner **GVLs** im SPS-Projektbaum.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Globale Variablenliste**.
3. Ändern Sie den automatisch eingetragenen Namen „GVL“ zu „GVL_Var“.
4. Bestätigen Sie mit **Öffnen**.

   ⇨ Im SPS-Projektbaum im Unterordner **GVLs** erscheint das Objekt „GVL_Var“ ( ). Der GVL-Editor öffnet sich.

   ⇨ Wenn die textuelle Ansicht erscheint, sind die Schlüsselwörter VAR_GLOBAL und END_VAR bereits enthalten.

5. Aktivieren Sie für das Beispiel mit einem Klick auf die Schaltfläche in der rechten Randleiste des Editors die tabellarische Ansicht.

   ⇨ Eine leere Zeile erscheint. Der Cursor befindet sich in der Spalte **Name**.

6. Tippen Sie „fTempActual“ im Feld **Name** ein.
   ⇨ Gleichzeitig werden in der Zeile automatisch der Gültigkeitsbereich VAR_GLOBAL und der Datentyp BOOL eingetragen.
7. Doppelklicken Sie in das Feld in der Spalte **Datentyp**.
   ⇨ Das Feld ist jetzt editierbar und die Schaltfläche  erscheint.
8. Klicken Sie auf die Schaltfläche und wählen Sie **Eingabehilfe**.
   ⇨ Der Dialog **Eingabehilfe** öffnet sich.
9. Wählen Sie den Datentyp REAL aus und klicken Sie auf die Schaltfläche **OK**.
10. Geben Sie einen numerischen Wert in der Spalte **Initialisierung** ein, beispielsweise „8.0“.

⇨ Deklarieren Sie die folgenden Variablen auf gleiche Weise:

Name	Datentyp	Initialisierung	Kommentar
fTempActual	REAL	1.0	Isttemperatur
fTempSet	REAL	8.0	Solltemperatur
bDoorOpen	BOOL	FALSE	Status der Tür
tImAlarmThreshold	TIME	T#30s	Kompressorlaufzeit, nach der ein Signal ertönt.
tDoorOpenThreshold	TIME	T#10s	Zeit ab Türöffnung, nach der ein Signal ertönt.
xCompressor	BOOL	FALSE	Steuersignal
xSignal	BOOL	FALSE	Steuersignal
xLamp	BOOL	FALSE	Statusmeldung

Hauptprogramm zur Kühlungssteuerung im CFC-Editor erstellen

Im standardmäßig angelegten Hauptprogrammbaustein MAIN beschreiben Sie die Hauptfunktion des SPS-Programms: Der Kompressor wird aktiv und kühlt, wenn die Isttemperatur höher ist als die Solltemperatur zuzüglich einer Hysterese. Der Kompressor wird ausgeschaltet, sobald die Isttemperatur niedriger ist als die Solltemperatur abzüglich der Hysterese.

Um diese Funktionalität in der Implementierungssprache „Continous Function Chart (CFC)“ zu beschreiben, führen Sie folgende Schritte aus:

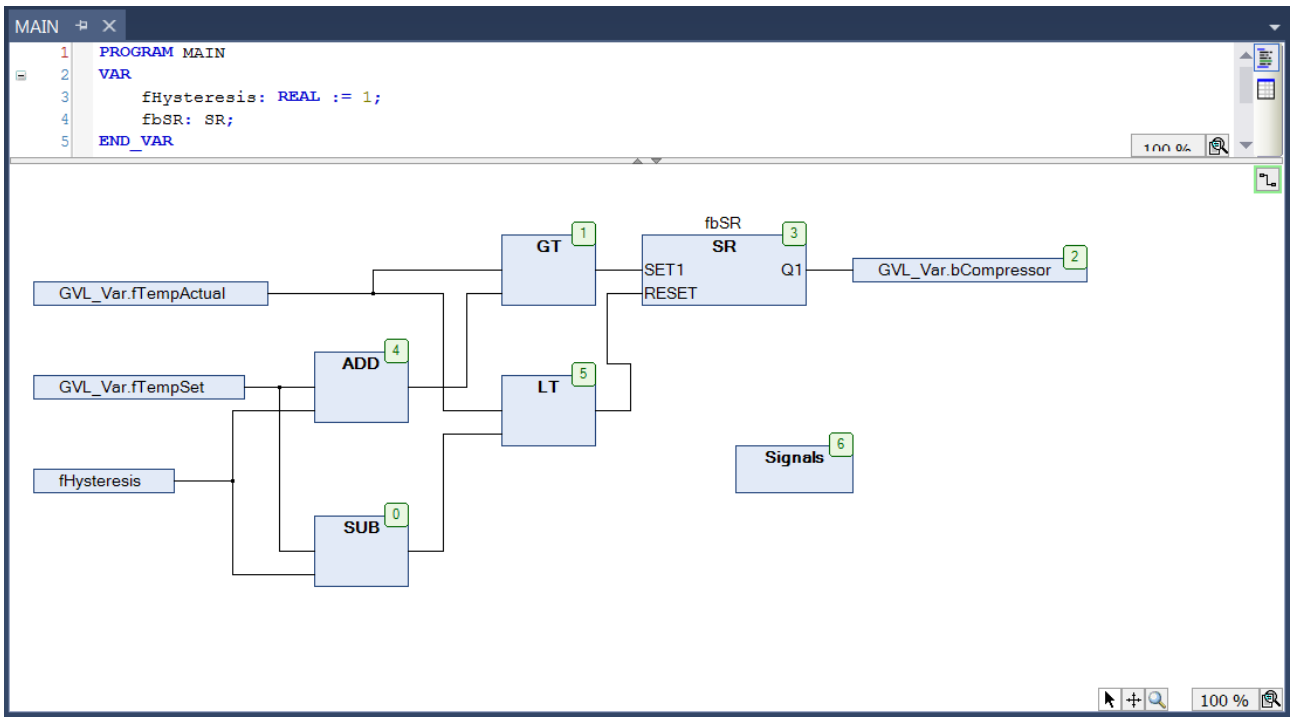
✓ Da der automatisch angelegte Hauptprogrammbaustein MAIN standardmäßig in der Implementierungssprache „Strukturierter Text (ST)“ angelegt wird, müssen Sie dieses Programm zunächst löschen. Mit dem Kontextmenübefehl **Hinzufügen > POU…** legen Sie ein neues Programm MAIN in der Implementierungssprache „Continous Function Chart (CFC)“ an.

1. Doppelklicken Sie auf das Programm **MAIN** im SPS-Projektbaum (Unterordner **POUs**).
   ⇨ Der CFC-Editor öffnet sich mit der Registerkarte **MAIN**. Oberhalb des grafischen Editorbereichs erscheint der Deklarationseditor in textueller oder tabellarischer Darstellung. Rechts ist die Ansicht **Werkzeugkasten**. Sollte der Werkzeugkasten nicht erscheinen, können Sie ihn über den Befehl **Werkzeugkasten** im Menü **Ansicht** aufrufen und auf der Arbeitsfläche platzieren.
2. Klicken Sie in der Ansicht **Werkzeugkasten** auf das Element **Eingang** und ziehen Sie es mit der Maus an eine Stelle im CFC-Editor.
   ⇨ Der namenlose Eingang **???** wurde eingefügt.
3. Klicken Sie im CFC-Editor auf den Eingang **???** und öffnen Sie mit einem Klick auf  die **Eingabehilfe**.
4. Wählen Sie aus der Kategorie **Variablen** unter **Projekt > GVLs** die Variable `fTempActual` aus.
5. Bestätigen Sie den Dialog mit **OK**, um die globale Variable `fTempActual` zu referenzieren.
6. Legen Sie wie bei Schritt 3 einen weiteren Eingang mit dem Namen der globalen Variable `rTempSet` an.
7. Legen Sie einen weiteren Eingang an.
8. Klicken Sie auf **???** und ersetzen Sie diese mit dem Namen `fHysteresis`.
   ⇨ Da dies nicht der Name einer bereits bekannten Variablen ist, erscheint der Dialog **Variable deklarieren**. Der Name ist bereits in den Dialog übernommen.

9. Füllen Sie die Felder im Dialog **Variable deklarieren** mit dem Datentyp REAL und dem Initialisierungswert „1“ aus.
10. Klicken Sie auf die Schaltfläche **OK**.
    ⇨ Die Variable `fHysteresis` erscheint im Deklarationseditor.
11. Nun fügen Sie einen Additionsbaustein ein: Klicken Sie in der Ansicht **Werkzeugkasten** auf das Element **Baustein** und ziehen Sie es mit der Maus an eine Stelle im CFC-Editor.
    ⇨ Der Baustein erscheint im CFC-Editor.
12. Ersetzen Sie **???** mit ADD.
    ⇨ Der Baustein ADD (Addition) addiert alle Eingänge, die mit ihm verbunden sind.
13. Verbinden Sie den Eingang GVL_Var.fTempSet mit dem Baustein ADD: klicken Sie dazu auf den Ausgangsverbinder des Eingangs und ziehen Sie Ihn bis zum oberen Eingang des Bausteins ADD.
14. Verbinden Sie auf die gleiche Weise den Eingang `fHysteresis` mit dem unteren Eingang des Bausteins ADD.
    ⇨ Die beiden Eingänge `fHysteresis` und `fTempSet` werden nun von ADD addiert.
15. Wenn Sie ein Element im Editor verschieben möchten, klicken Sie auf eine freie Stelle im Element oder auf den Rahmen, sodass das Element selektiert ist (roter Rahmen, rot schattiert).
16. Halten Sie die Maustaste gedrückt und ziehen Sie das Element an die gewünschte Position.
17. Legen Sie einen weiteren Baustein rechts vom Baustein ADD an.
    ⇨ Er soll „GVL_Var.fTempActual“ mit der Summe aus „GVL_Var.fTempSet“ und fHysteresis vergleichen.
18. Geben Sie dem Baustein die Funktion GT (Greater Than).
    ⇨ Der GT-Baustein arbeitet folgendermaßen:
    ```
    IF (oberer Eingang > unterer Eingang) THEN Ausgang := TRUE;
    ```
19. Verbinden Sie den Eingang „GVL_Var.fTempActual“ mit dem oberen Eingang des Bausteins GT.
20. Verbinden Sie den Ausgang des Bausteins ADD mit dem unteren Eingang des Bausteins GT.
21. Legen Sie einen weiteren Funktionsbaustein rechts vom Baustein GT an, der den Kühlkompressor je nach Eingangsbedingung startet oder stoppt (Set - Reset).
22. Tippen Sie im Feld **???** den Namen „SR“ ein.
23. Schließen Sie das geöffnete Eingabefeld oberhalb des Bausteins (SR_0) mit der Eingabetaste.
    ⇨ Der Dialog **Variable deklarieren** erscheint.
24. Deklarieren Sie die Variable mit dem Namen „fbSR“ und dem Datentyp SR.
25. Klicken Sie auf die Schaltfläche **OK**.
    ⇨ Der Baustein SR, ebenfalls in der Bibliothek Tc2_Standard definiert, bestimmt das THEN am Ausgang des GT-Bausteins. Die Eingänge SET1 und RESET erscheinen.
26. Verbinden Sie die Ausgangsverbindung rechts am Baustein GT mit dem Eingang SET1 des Bausteins fbSR.
    ⇨ SR kann eine boolesche Variable von FALSE auf TRUE und wieder zurücksetzen. Wenn die Bedingung am Eingang SET1 zutrifft, wird die boolesche Variable auf TRUE gesetzt. Trifft die Bedingung an RESET zu, wird die Variable wieder zurückgesetzt. Die boolesche (globale) Variable ist in unserem Beispiel „GVL_Var.bCompressor“.
27. Legen Sie ein Element **Ausgang** an und weisen Sie ihm die globale Variable „GVL_Var.bCompressor“ zu. Ziehen Sie eine Verbindungslinie zwischen „GVL_Var.bCompressor“ und der Ausgangsverbindung Q1 von SR.

Jetzt geben Sie an, unter welcher Bedingung sich der Kompressor wieder abschalten soll, also der RESET-Eingang des SR-Bausteins ein TRUE-Signal erhält. Dazu formulieren Sie die gegenteilige Bedingung wie oben. Verwenden Sie dazu die Bausteine SUB (Subtract) und LT (Less Than).

Folgender CFC-Plan entsteht:



Erstellen eines Programmbausteins zur Signalverwaltung im Kontaktplan-Editor

Sie implementieren nun in einem weiteren Programmbaustein die Signalverwaltung für den Alarmtongeber und für das Ein- und Ausschalten der Lampe. Dafür eignet sich die Implementierungssprache „Kontaktplan (KOP)“.

Behandeln Sie die folgenden Signale jeweils in einem eigenen Netzwerk:

- Wenn der Kompressor zu lange läuft, weil die Temperatur zu hoch ist, macht ein durchgehendes akustisches Signal darauf aufmerksam.
- Wenn die Türe zu lange geöffnet ist, macht ein getaktetes Signal darauf aufmerksam.
- Solange die Türe geöffnet ist, brennt das Licht.

1. Legen Sie im SPS-Projektbaum (Unterordner **POUs**) ein POU-Objekt des Typs **Programm** mit der Implementierungssprache „Kontaktplan (KOP)“ an.
2. Nennen Sie das POU-Objekt „Signals“.
   ⇨ „Signals“ erscheint im SPS-Projektbaum unterhalb von MAIN. Der Kontaktplan-Editor öffnet sich mit der Registerkarte **Signals**. Im oberen Teil erscheint der Deklarationseditor, rechts die Ansicht **Werkzeugkasten**. Der KOP enthält ein leeres Netzwerk.
3. Im Netzwerk programmieren Sie, dass ein akustisches Signal ertönt, wenn der Kühlkompressor zu lange läuft, ohne die Solltemperatur zu erreichen. Fügen Sie dazu einen Timer-Baustein TON ein.
   ⇨ Er schaltet ein boolesches TRUE-Signal erst nach einer vorgegebenen Zeit auf TRUE.
4. Wählen Sie in der Ansicht **Werkzeugkasten** unter **Funktionsbausteine** einen TON aus und ziehen Sie ihn mit der Maus in das leere Netzwerk auf das erscheinende Rechteck **Hier starten**.
5. Wenn das Feld grün wird, lassen Sie die Maustaste los.
   ⇨ Der Baustein erscheint als Rechteck mit Ein- und Ausgängen und erhält automatisch den Instanznamen TON_0. Der Zeileneditor ist geöffnet und der Cursor blinkt.
6. Bestätigen Sie den Instanznamen mit der Eingabetaste.
   ⇨ Der Dialog **Variable deklarieren** öffnet sich.
7. Deklarieren Sie die Variable mit dem Namen „fbTimer1“ und dem Datentyp TON.
8. Klicken Sie auf die Schaltfläche **OK**.

Wenn Sie die Hilfe zum Funktionsbaustein lesen wollen, markieren Sie den vollständigen Namen des Bausteins mit dem Cursor und drücken Sie [F1].

9. Um zu programmieren, dass der Baustein aktiviert wird, sobald der Kühlkompressor zu laufen beginnt, benennen Sie den Kontakt am oberen Eingang des Bausteins mit „GVL_Var.bCompressor“.
   ⇨ Diese boolesche Variable haben Sie bereits in der globalen Variablenliste „GVL_Var“ definiert.

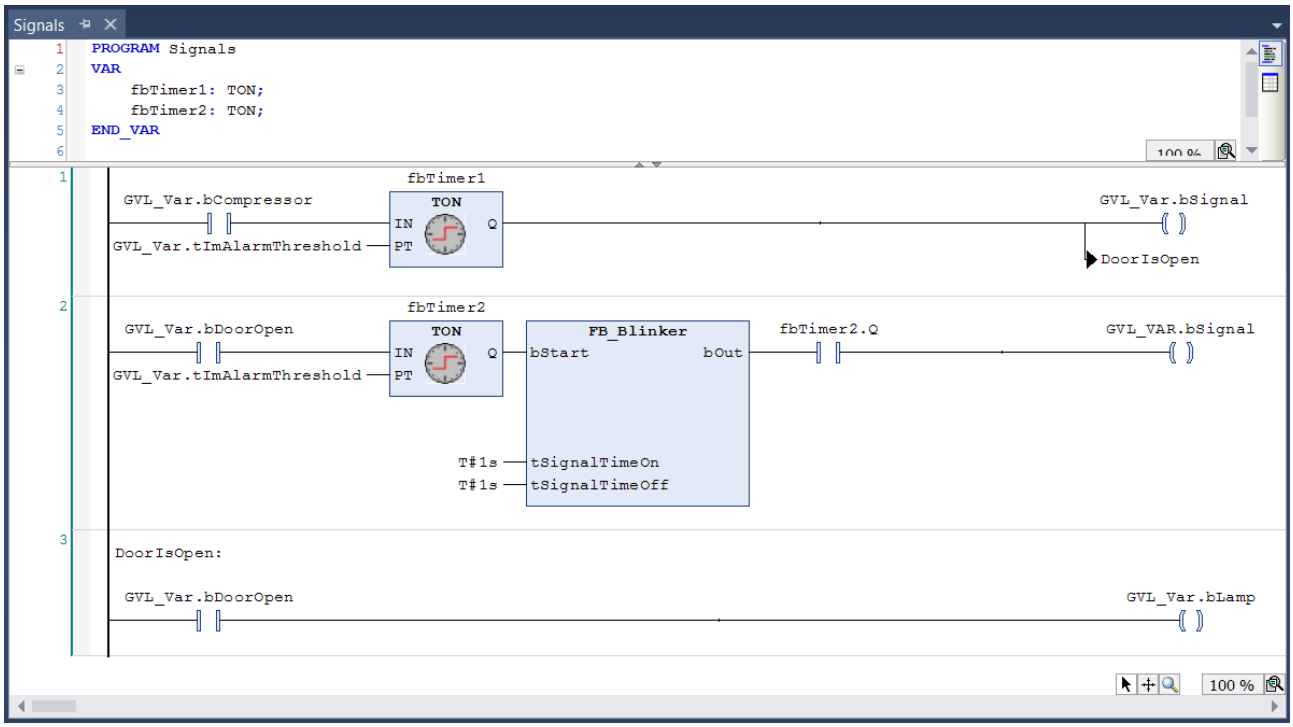
Wenn Sie beginnen, an der Eingabeposition einen Variablennamen einzugeben, erhalten Sie immer automatisch eine Liste aller Variablen, deren Namen mit den eingetippten Zeichen beginnen und die an dieser Stelle verwendbar sind. Diese Unterstützung ist eine Standardeinstellung in den TwinCAT-Optionen für Intelligentes Kodieren („Smart Coding“).

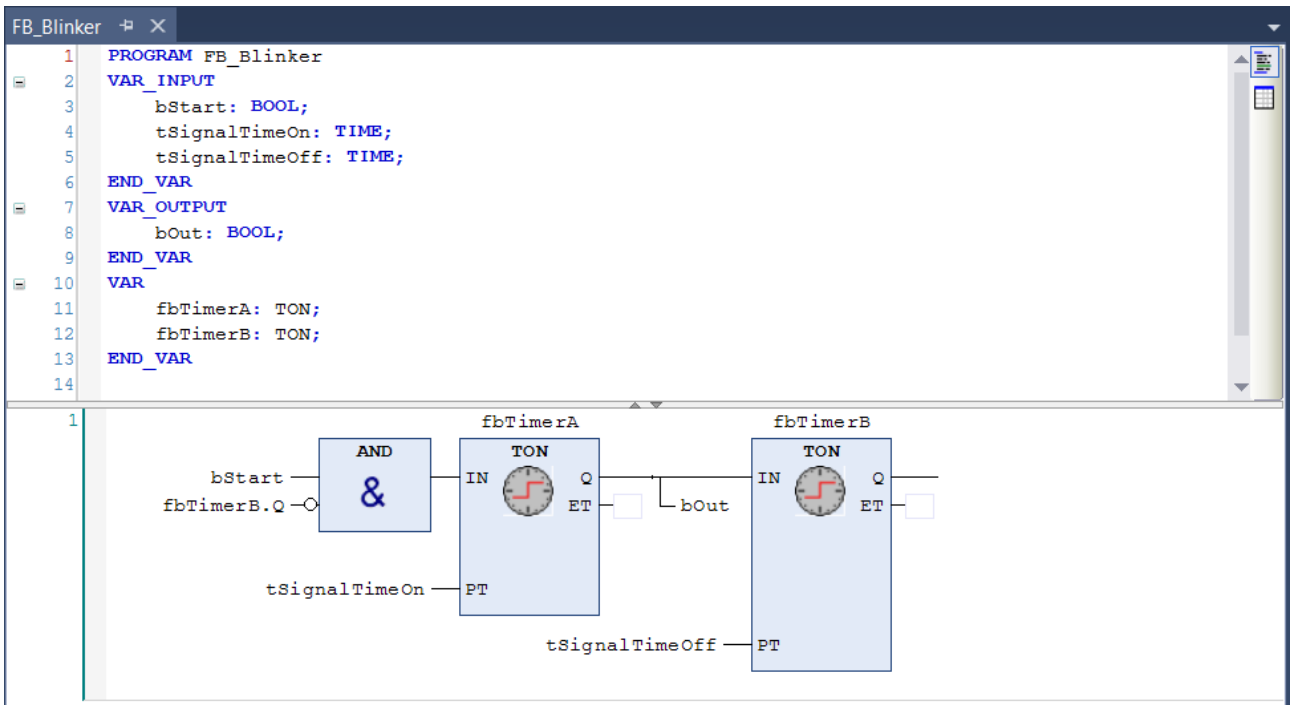
10. Fügen Sie das Signal ein, das aktiviert werden soll: Ziehen Sie dazu aus der Ansicht **Werkzeugkasten** Kategorie **Kontaktplan-Elemente** eine **Spule** an den Ausgang Q des TON-Bausteins. Benennen Sie die Spule mit „GVL_Var.bSignal“.
11. Fügen Sie die Variable „GVL_Var.tImAlarmThreshold“ am Eingang PT von „fbTimer1“ ein, um die Zeit ab Aktivierung des TON-Bausteins, nach der das Signal ertönen soll, zu definieren. Klicken Sie dazu auf das fein umrandete Rechteck rechts der Eingangsverbindung.
12. Geben Sie den Variablennamen ein.
13. Klicken Sie auf den TON-Baustein und wählen Sie im Kontextmenü den Befehl **Nicht verwendete FB-Aufruf-Parameter entfernen**.
    ⇨ Der nicht verwendete Ausgang ET wurde entfernt.
14. Im zweiten Netzwerk des KOP programmieren Sie, dass das Signal getaktet ertönen soll, wenn die Türe zu lange geöffnet ist.
15. Legen Sie dazu zunächst im SPS-Projektbaum, Unterordner **POUs**, ein neues POU-Objekt vom Typ **Funktionsbaustein** in der Implementierungssprache „Funktionsplan (FUP)“ an.
16. Nennen Sie es „FB_Blinker“.
    ⇨ Der Funktionsbaustein FB_Blinker erscheint im SPS-Projektbaum oberhalb von MAIN. Der FUP-Editor öffnet mit der Registerkarte **FB_Blinker**. Im oberen Teil erscheint der Deklarationseditor, rechts die Ansicht **Werkzeugkasten**. Der FUP enthält ein leeres Netzwerk.
    ⇨ Der Blinker soll durch einen AND-Operator und zwei TON-Bausteine realisiert werden.
17. Wählen Sie in der Ansicht **Werkzeugkasten** unter **Allgemeine** einen **Baustein** aus und ziehen Sie ihn mit der Maus ins leere Netzwerk auf das erscheinende Rechteck **Hier starten**.
18. Wenn das Feld grün wird, lassen Sie die Maustaste los.
    ⇨ Der Baustein erscheint als Rechteck mit Ein- und Ausgängen.
19. Klicken Sie auf **???** innerhalb des Bausteins und geben Sie in das nun editierbare Feld das Schlüsselwort AND ein.
20. Bestätigen Sie mit der Eingabetaste.
    ⇨ Da es sich um eine Funktion handelt, ist keine Instanziierung notwendig.
21. Wählen Sie in der Ansicht **Werkzeugkasten** unter **Funktionsbausteine** einen TON-Baustein aus und ziehen Sie ihn mit der Maus in das Netzwerk an den Ausgang des AND-Bausteins.
22. Wenn das Feld grün wird, lassen Sie die Maustaste los.
    ⇨ Der Baustein erscheint als Rechteck mit Ein- und Ausgängen und erhält automatisch den Instanznamen TON_0.
23. Schließen Sie das geöffnete Eingabefeld oberhalb des Bausteins (TON_0) mit der Eingabetaste.
    ⇨ Der Dialog **Variable deklarieren** öffnet sich.
24. Deklarieren Sie die Variable mit dem Namen „fbTimerA“ und dem Datentyp TON.
25. Klicken Sie auf die Schaltfläche **OK**.
26. Wählen Sie in der Ansicht **Werkzeugkasten** unter **Funktionsbausteine** einen weiteren TON-Baustein aus und ziehen Sie ihn mit der Maus in das Netzwerk an den Ausgang des TON-Bausteins „fbTimerA“.
27. Wenn das Feld grün wird, lassen Sie die Maustaste los.
    ⇨ Der Baustein erscheint als Rechteck mit Ein- und Ausgängen und erhält automatisch den Instanznamen „TON_0“.
28. Schließen Sie das geöffnete Eingabefeld oberhalb des Bausteins (TON_0) mit der Eingabetaste.
    ⇨ Der Dialog **Variable deklarieren** öffnet sich.

29. Deklarieren Sie die Variable mit dem Namen „fbTimerB“ und dem Datentyp TON. Klicken Sie auf die Schaltfläche **OK**.
    ⇨ Der erste Eingang des AND-Bausteins soll mit einer booleschen Variable „bStart“ verbunden werden.
30. Klicken Sie auf **???** und tragen Sie den Variablennamen ein.
31. Bestätigen Sie mit der Eingabetaste.
    ⇨ Der Dialog **Variable deklarieren** öffnet sich. Der Name und der Datentyp werden automatisch erkannt.
32. Wählen Sie als **Gültigkeitsbereich** den Eintrag VAR_INPUT aus.
33. Bestätigen Sie den Dialog mit **OK**.
    ⇨ Der zweite Eingang des AND-Bausteins soll mit dem Ausgang Q des zweiten TON-Bausteins „fbTimerB“ verbunden werden.
34. Klicken Sie auf **???** und öffnen Sie über … die **Eingabehilfe**.
35. Wählen Sie in der Kategorie **Variablen** den Funktionsbaustein „fbTimerB“ und den Ausgang Q aus.
36. Bestätigen Sie den Dialog mit **OK**.
    ⇨ Am zweiten Eingang wird die Variable „fbTimerB.Q“ ergänzt.
37. Markieren Sie den zweiten Eingang und öffnen Sie das Kontextmenü mit einem Rechtklick.
38. Wählen Sie den Befehl **Negation**, um den Eingang zu negieren.
    ⇨ Am entsprechenden Eingang erscheint ein Kreis.
39. Über die Eingänge PT der TON-Bausteine geben Sie die Zeit vor bis der Ausgang Q gesetzt wird.
40. Deklarieren Sie für die TON-Bausteine „fbTimerA“ und „fbTimerB“ über den Dialog **Variable deklarieren** die Eingangsvariablen „tSignalTimeOn“ und „tSignalTimeoff“.
41. Wählen Sie als **Gültigkeitsbereich** den Eintrag VAR_INPUT.
    ⇨ Der erzeugte Taktimpuls soll am Ausgang Q des TON-Bausteins „fbTimerA“ ausgegeben werden.
42. Wählen Sie dazu in der Ansicht **Werkzeugkasten** unter **Allgemeine** eine **Zuweisung** aus und ziehen Sie sie mit der Maus in das Netzwerk an den Ausgang des TON-Bausteins „fbTimerA“.
43. Wenn das Feld grün wird, lassen Sie die Maustaste los.
    ⇨ Die Zuweisung wird zwischen den Bausteinen „fbTimerA“ und „fbTimerB“ ergänzt.
44. Klicken Sie auf **???** und tragen Sie den Variablennamen „bOut“ ein.
45. Bestätigen Sie mit der Eingabetaste.
    ⇨ Der Dialog **Variable deklarieren** öffnet sich.
46. Wählen Sie den **Gültigkeitsbereich** VAR_OUTPUT und den Datentyp BOOL.
47. Bestätigen Sie den Dialog.
48. Entfernen Sie zuletzt die **???** an den nicht benutzten Ein- und Ausgängen der Bausteine.
    ⇨ Der fertig gestellte Funktionsbaustein FB_Blinker kann nun instanziiert und aufgerufen werden.
49. Öffnen Sie das Programm „Signals“ im FUP/KOP/AWL-Editor.
50. Klicken Sie unterhalb des ersten Netzwerks in das Editorfenster.
51. Wählen Sie im Kontextmenü den Befehl **Netzwerk einfügen**.
    ⇨ Ein leeres Netzwerk mit Nummer 2 erscheint.
52. Implementieren Sie wie im ersten Netzwerk einen TON-Baustein zur zeitgesteuerten Aktivierung des Signals, diesmal getriggert durch die globale Variable „GVL_Var.bDoorOpen“ am Eingang IN.
53. Am Eingang PT fügen Sie die globale Variable „GVL_Var.tImDoorOpenThreshold“ hinzu.
54. Zusätzlich fügen Sie in diesem Netzwerk am Ausgang Q des TON-Bausteins den Funktionsbaustein FB_Blinker ein.
    ⇨ Der Baustein FB_Blinker taktet die Signalweiterleitung Q und damit „GVL_Var.bSignal“.
55. Ziehen Sie hierfür ein Element **Kontakt** aus der Ansicht **Werkzeugkasten** an den Ausgang OUT des Bausteins.
56. Weisen Sie dem Kontakt die Variable „fbTimer2.Q“ zu.

57. Fügen Sie hinter dem Kontakt ein Element **Spule** ein und weisen Sie ihr die globale Variable „GVL_Var.bSignal“ zu.
58. Weisen Sie den beiden Eingangsvariabeln „tSignalTimeOn“ und „tSignalTimeOff“ des Funktionsbausteins FB_Blinker den Wert T#1s zu.
    ⇨ Die Taktdauer ist somit jeweils 1 Sekunde für TRUE und 1 Sekunde für FALSE.
59. Klicken Sie auf den TON-Baustein.
60. Wählen Sie im Kontextmenü den Befehl **Nicht verwendete FB-Aufruf-Parameter entfernen**.
    ⇨ Der nicht verwendete Ausgang ET wird entfernt.
    ⇨ Im dritten Netzwerk des KOP programmieren Sie, dass die Lampe leuchtet, solange die Tür geöffnet ist.
61. Fügen Sie dazu ein weiteres Netzwerk ein und darin links einen Kontakt „GVL_Var.bDoorOpen“, der direkt auf eine eingefügte Spule „GVL_Var.bLamp“ leitet.
    ⇨ TwinCAT arbeitet die Netzwerke eines KOP nacheinander ab.
62. Um zu erreichen, dass nur Netzwerk 1 oder nur Netzwerk 2 ausgeführt wird, bauen Sie am Ende von Netzwerk 1 einen Sprung zu Netzwerk 3 ein.
63. Selektieren Sie Netzwerk 3 durch einen Mausklick ins Netzwerk oder in das Feld mit der Netzwerknummer.
64. Wählen Sie aus dem Kontextmenü den Befehl **Sprungmarke einfügen**.
65. Ersetzen Sie den Text **Label:** der Sprungmarke im linken oberen Bereich des Netzwerks durch „DoorIsOpen“.
66. Selektieren Sie Netzwerk 1.
67. Ziehen Sie aus der Ansicht **Werkzeugkasten**, Kategorie **Allgemeine**, das Element **Sprung** ins Netzwerk.
68. Platzieren Sie es auf dem erscheinenden Rechteck **Ausgang** oder **Sprung hier einfügen**.
    ⇨ Das Sprungelement erscheint. Das Sprungziel ist noch mit **???** angegeben.
69. Selektieren Sie **???** und klicken Sie auf die Schaltfläche ... .
70. Wählen Sie aus den möglichen Bezeichnern von Sprungmarken „DoorIsOpen“ aus.
71. Bestätigen Sie mit **OK**.
    ⇨ Die Sprungmarke zu Netzwerk 3 ist implementiert.

⇨ Das KOP-Programm sieht nun folgendermaßen aus:





Aufrufen des Programms „Signals“ im Hauptprogramm

In unserem Programmbeispiel soll das Hauptprogramm MAIN das Programm „Signals“ zur Signalverarbeitung aufrufen.

1. Doppelklicken Sie im SPS-Projektbaum auf das Programm MAIN.
   ⇨ Das Programm MAIN öffnet sich im Editor.
2. Ziehen Sie ein Element **Baustein** aus der Ansicht **Werkzeugkasten** in den Editor von MAIN.
3. Fügen Sie diesem Baustein über die **Eingabehilfe** aus der Kategorie **Bausteinaufrufe** den Aufruf des Programms „Signals“ hinzu.

Erstellen eines ST-Programmbausteins für eine Simulation

Da dieses Beispielprojekt nicht mit realen Sensoren und Aktoren verknüpft ist, schreiben Sie ein Programm zur Simulation von Temperaturanstieg und Temperatursenkung. Damit können Sie nachher das Arbeiten der Kühlschranksteuerung im Onlinebetrieb beobachten.

Sie erstellen das Simulationsprogramm in „Strukturiertem Text (ST)“.

Das Programm erhöht die Temperatur so lange, bis das Hauptprogramm MAIN feststellt, dass die Solltemperatur überschritten ist und den Kühlkompressor aktiviert. Daraufhin senkt das Simulationsprogramm die Temperatur wieder, bis das Hauptprogramm den Kompressor wieder deaktiviert.

1. Fügen Sie dem SPS-Projektbaum einen POU-Baustein namens „Simulation“, des Typs **Programm** und der Implementierungssprache „Strukturierter Text (ST)“ein.
2. Implementieren Sie Folgendes im ST-Editor:

```
PROGRAM Simulation
VAR
    fbT1                : TON;              //
The temperature is decreased on a time delay, when the compressor has been activated
    tCooling            : TIME := T#500MS;
    bReduceTemp         : BOOL;             //Signal for dereasing the temperature
    fbT2                : TON;              //
The temperature is increased on a time delay, when the compressor has been activated
    tEnvironment        : TIME := T#2S;     //Delay time when the door is closed
    tEnvironmentDoorOpen : TIME := T#1s;    //Delay time when the door is open
    bRaiseTemp          : BOOL;             //Signal for increasing the temperature
    tImTemp             : TIME;             //Delay time
    nCounter            : INT;
END_VAR

// After the compressor has been activated due to fTempActual being too high, the temperature de
```

```
creases.
// The temperature is decremented by 0.1°C per cycle after a delay of P_Cooling
IF GVL_VAR.bCompressor THEN
    fbT1(IN:= GVL_Var.bCompressor, PT:= tCooling, Q=>bReduceTemp);
    IF bReduceTemp THEN
        GVL_Var.fTempActual := GVL_Var.fTempActual-0.1;
        fbT1(IN:=FALSE);
    END_IF
END_IF

//If the door is open, the warming will occur faster; SEL selects tEnvironmentDoorOpen
tImTemp:=SEL(GVL_Var.bDoorOpen, tEnvironment, tEnvironmentDoorOpen);

//If the compressor is not in operation, then the cooling chamber will become warmer.
//The  temperature is incremented by 0.1°C per cycle after a delay of tImTemp
fbT2(IN:= TRUE, PT:= tImTemp, Q=>bRaiseTemp);
IF bRaiseTemp THEN
    GVL_Var.fTempActual := GVL_Var.fTempActual + 0.1;
    fbT2(IN:=FALSE);
END_IF

nCounter := nCounter+1; // No function, just for demonstration purposes.
```

Visualisierung

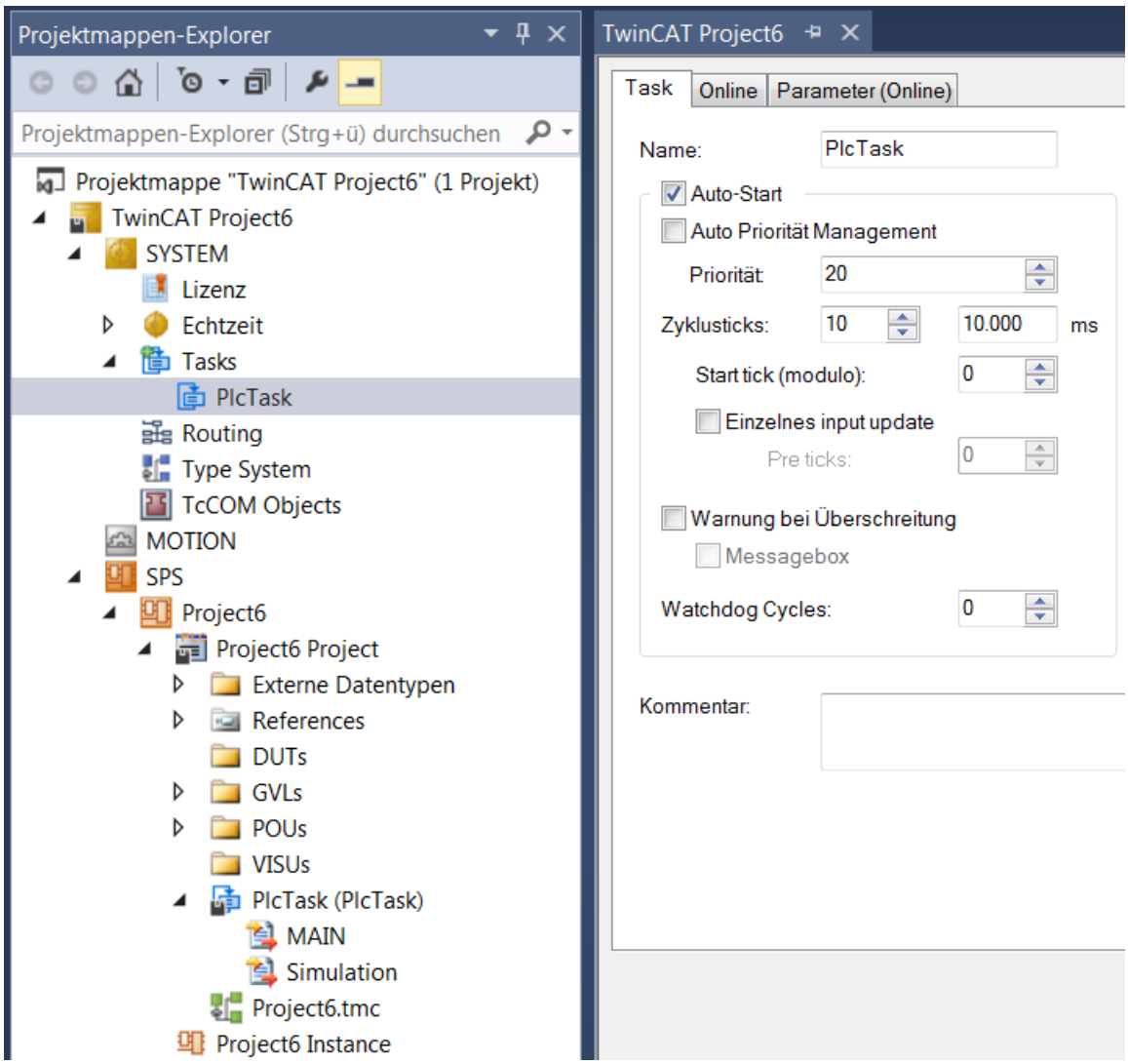
Für ein komfortables Bedienen und Beobachten des gesamten Steuerungsprogramms kann eine Visualisierung eingesetzt werden, die den Kühlschrank darstellt und das Arbeiten des Simulationsprogramms wiedergibt. Eine Visualisierung können Sie in TwinCAT auch im SPS-Bereich programmieren. Beim Starten des Projekts auf der Steuerung startet die Visualisierung, ohne dass Sie eine Eingabe vornehmen müssen. Je nach Programmierung können Sie z. B. über einen Mausklick auf einen Ein/Aus-Schalter das Öffnen und Schließen der Tür herbeiführen oder über die Nadel eines Drehreglers die Temperaturvorwahl verstellen. Das Erstellen einer Visualisierung wird hier nicht weiter beschrieben.

Festlegen der auszuführenden Programme in der Taskkonfiguration

Die voreingestellte Taskkonfiguration enthält den Aufruf für das Hauptprogramm MAIN. Für unser Beispielprojekt müssen Sie den Aufruf für das Programm „Simulation“ hinzufügen.

1. Ziehen Sie im SPS-Projektbaum den Eintrag „Simulation“ mit der Maus auf die Taskreferenz (PlcTask).
   ⇨ Das Programm „Simulation“ wird in der Taskkonfiguration eingefügt.
2. Wenn Sie sich die Taskkonfiguration ansehen möchten, doppelklicken Sie auf den Eintrag **PlcTask** im SPS-Projektbaum.
   ⇨ Im **Projektmappen-Explorer** wird der referenzierte Task (**PlcTask**) unter **SYSTEM > Tasks** aktiviert.
3. Doppelklicken Sie auf den Task, um die Konfiguration des Tasks in einem Editor zu öffnen.

⇨ Sie sehen im SPS-Projektbaum unter **PlcTask** die POUs, die von dem Task aufgerufen werden: MAIN (standardmäßig eingetragen) und Simulation. Im Editor des Knotens **PlcTask** im SYSTEM-Bereich sehen Sie für den referenzierten PlcTask die entsprechende Zykluszeit. Das Intervall beträgt in diesem Beispiel 10 Millisekunden. Im Onlinebetrieb wird der Task die beiden Bausteine pro Zyklus 1x abarbeiten.



Definieren des aktiven SPS-Projekts

Auf einer TwinCAT-Steuerung können mehrere SPS-Projekte ausgeführt werden. Der erste Eintrag der Drop-down-Liste **Active PLC Project** in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** zeigt das gerade aktive SPS-Projekt an. Wenn mehrere SPS-Projekte vorhanden sind, können Sie über die Drop-down-Liste ein SPS-Projekt auswählen.

Überprüfen des SPS-Projekts auf Fehler

Während der Eingabe von Code weist TwinCAT Sie sofort durch rote Unterschlängelung auf Syntaxfehler hin.

1. Um eine Syntaxprüfung über das gesamte SPS-Projekt zu erhalten, markieren Sie das SPS-Projektobjekt „<SPS-Projektname> Project“.
2. Wählen Sie den Befehl **Überprüfe alle Objekte** im Kontextmenü oder im Menü **Erstellen**.
   ⇨ Die Ergebnisse der Prüfung sehen Sie in der Ansicht **Fehlerliste**.
3. Wenn nötig, öffnen Sie die Ansicht **Fehlerliste** mit dem Befehl **Fehlerliste** im Menü **Ansicht**.
4. Sie können dann mit einem Doppelklick auf die Meldung zu der entsprechenden Codestelle springen.

Mit dem Befehl **Überprüfe alle Objekte** werden alle im SPS-Knoten vorhandenen Bausteine geprüft und übersetzt. Wenn bei der Übersetzung Bausteine im Baum auftauchen, die nicht übersetzbar sind, weil sie vielleicht nur für einen Test eingefügt wurden, wird ein Fehler erzeugt. Deshalb empfiehlt sich der Befehl **Überprüfe alle Objekte** besonders für die Überprüfung von Bibliotheksbausteinen.

Mit dem Befehl **Erstellen** oder **Neu Erstellen** werden nur die Bausteine geprüft und übersetzt, die im SPS-Projekt auch wirklich genutzt werden.

Weitere Prüfungen des SPS-Projekts werden durchgeführt, wenn dieses auf die Steuerung geladen wird.

Sie können nur ein fehlerfreies SPS-Projekt auf die Steuerung laden.

Kompilieren des SPS-Moduls

Mit dem Befehl **Erstellen** oder **Neu erstellen** wird der von Ihnen im SPS-Projekt verwendete Code kompiliert und damit auf syntaktische Richtigkeit geprüft.

Auswahl des Zielsystems

Wählen Sie nun in der Drop-down-Liste **Choose Target System** der Symbolleiste **TwinCAT XAE Base Symbolleistenoptionen** das Zielgerät für Ihr Steuerungsprogramm aus:

- Wenn der Steuerungscode direkt in Ihre lokale Laufzeit Ihres Programmiergeräts geladen werden soll, wählen Sie den Eintrag **<Lokal>**. (Wählen Sie diese Optionen für das vorliegende Beispiel.)
- Wenn Sie ein anderes Zielgerät auswählen wollen, wählen Sie in der Drop-down-Liste den Eintrag **Zielsystem wählen**. Wählen Sie dann ein schon konfiguriertes Zielgerät aus oder suchen Sie im Netzwerk nach einem Zielgerät, konfigurieren Sie dieses und wählen Sie dieses dann aus.

Aktivierung der Konfiguration

1. Klicken Sie auf die Schaltfläche  in den **TwinCAT XAE Base Symbolleistenoptionen**.
   ⇨ Ein Dialog erscheint mit der Abfrage, ob die Konfiguration aktiviert werden soll.
2. Klicken Sie auf **Ok**.
   ⇨ Ein Dialog erscheint mit der Abfrage, ob TwinCAT im Run-Modus neu gestartet werden soll.
3. Klicken Sie auf **Ok**.

⇨ Die Konfiguration wird aktiviert und TwinCAT in den Run-Modus gesetzt. In der Taskleiste erscheint der aktuelle Status:  . Durch die Aktivierung wird auch das SPS-Projekt auf die Steuerung übertragen.

Laden des SPS-Projekts auf die SPS

✓ Das SPS-Projekt wurde fehlerfrei übersetzt. Siehe Schritt Überprüfen des SPS-Programms auf Fehler [▶ 35].

1. Wählen Sie im Menü **PLC** den Befehl **Einloggen** oder klicken Sie auf die Schaltfläche  in den **TwinCAT SPS Symbolleistenoptionen**.
   ⇨ Ein Dialog erscheint mit der Abfrage, ob die Applikation angelegt und geladen werden soll.
2. Klicken Sie auf **Yes**.

⇨ Das SPS-Projekt wird auf die Steuerung geladen. Die Engineering-Umgebung befindet sich nun im Onlinebetrieb. Die SPS-Module befinden sich noch nicht im Run-Modus. Im **Projektmappen-Explorer** erscheint vor dem SPS-Projektobjekt folgendes Symbol:  . Während des Ladevorgangs, werden in der Ansicht **Fehlerliste** unter anderem Informationen zur generierten Codegröße, zur Größe der globalen Daten, zum resultierenden Speicherbedarf auf der Steuerung ausgegeben.

Starten des Programms

Wenn Sie das Tutorial bis zu diesem Punkt vollständig befolgt haben, können Sie nun das SPS-Projekt auf dem SPS-Gerät verwenden.

1. Wählen Sie im Menü **PLC** den Befehl **Start** oder klicken Sie auf die Schaltfläche  in den (online) **TwinCAT SPS Symbolleistenoptionen** (**[F5]**).

⇨ Das Programm läuft. Die SPS-Module befinden sich im Run-Modus. Im **Projektmappen-Explorer** erscheint vor dem SPS-Projektobjekt folgendes Symbol:  .

Monitoring und einmaliges Schreiben von Variablenwerten zur Laufzeit

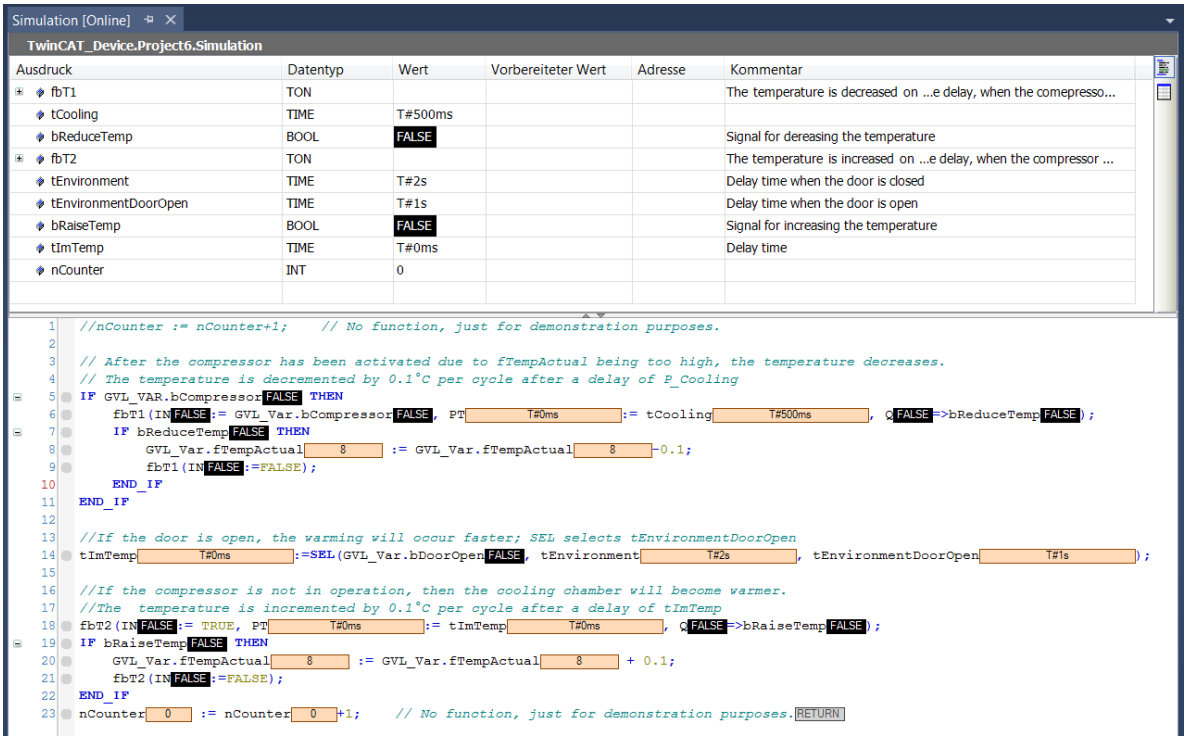
Im Folgenden sehen Sie sich das Monitoring der Variablenwerte in den verschiedenen Programmbausteinen an und setzen aus TwinCAT heraus einmalig einen bestimmten Variablenwert auf der Steuerung.

Die Istwerte der Programmvariablen sehen Sie in den Online-Ansichten der Bausteineditoren oder in Überwachungslisten. Im Beispiel hier beschränken wir uns auf das Monitoring im Bausteineditor.

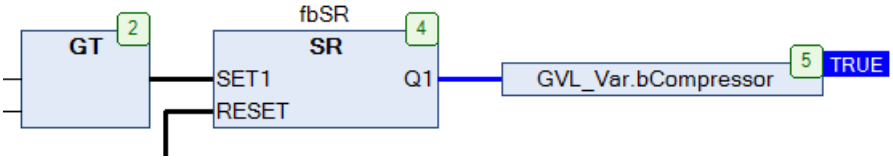
✓ Das SPS-Programm läuft auf der Steuerung.

1. Öffnen Sie mit Doppelklicks auf die Objekte MAIN, „Signals“, „Simulation“ und „GVL_Var“ im SPS-Projektbaum die Online-Ansichten der Editoren.

   ⇨ Im Deklarationsteil jeder Ansicht erscheint in der Tabelle der Ausdrücke in der Spalte **Wert** der Istwert der Variablen auf der Steuerung.



   ⇨ Das Monitoring im Implementierungsteil hängt von der Implementierungssprache ab: Bei nicht-booleschen Variablen steht der Wert immer in einem rechteckigen Feld rechts des Bezeichners. Im ST-Editor gilt dies auch für boolesche Variablen. Diese Anzeige wird „Inline-Monitoring“ genannt. In den grafischen Editoren wird der Wert einer booleschen Variablen durch die Farbe der Ausgangsverbindungslinie angezeigt: schwarz für FALSE, blau für TRUE:



⇨ Betrachten Sie die Veränderung der Variablenwerte in den verschiedenen Bausteinen. Beispielsweise sehen Sie in der globalen Variablenliste „GVL_Var“, wie sich durch die Abarbeitung des Simulationsprogramms die Werte von „fTempActual“ und „bCompressor“ ändern.

Einmaliges Setzen von Variablenwerten auf der Steuerung:

1. Setzen Sie den Fokus in die Online-Ansicht der globalen Variablenliste „GVL_Var“.
2. Um einen neuen Sollwert vorzugeben, doppelklicken Sie bei Ausdruck „fTempSet“ in die Spalte **Vorbereiteter Wert**.

   ⇨ Ein Eingabefeld öffnet sich.
3. Tragen Sie den Wert „9“ ein und verlassen Sie das Eingabefeld.
4. Um ein Offenstehen der Tür vorzugeben, klicken Sie beim Ausdruck „bDoorOpen“ einmal in das Feld **Vorbereiteter Wert**.

⇨ Der Wert TRUE wird eingetragen.

5. Klicken Sie weitere drei Male, um zu sehen, dass Sie damit den vorbereiteten Wert auf FALSE, dann wieder auf leer und dann wieder auf TRUE schalten können.
6. Um den vorbereiteten Wert TRUE einmalig auf die Variable zu schreiben, wählen Sie den Befehl **Werte schreiben** im Menü **PLC** oder klicken Sie auf die Schaltfläche in den **TwinCAT SPS Symbolleistenoptionen**.

⇨ Die beiden Werte werden jeweils in die Spalte **Wert** übertragen. Die Variable „bDoorOpen“ verändert ihren Wert jetzt nicht mehr und die Solltemperatur ist jetzt 9 Grad. Die Variable „tImTemp“ wechselt auf den Wert 1 s, da nun die Kühlschranktür „geöffnet ist“ und dadurch das Erwärmen durch Simulation schneller als vorher (2 s) erfolgen soll.

Setzen von Haltepunkten und schrittweise Ausführung zur Laufzeit

Debuggen: Für die Fehlersuche wollen Sie die Variablenwerte an bestimmten Codestellen überprüfen. Dazu können Sie Haltepunkte für die Abarbeitung definieren und eine schrittweise Ausführung der Anweisungen veranlassen.

✓ Das SPS-Programm ist auf die Steuerung geladen und läuft.

1. Öffnen Sie mit einem Doppelklick auf das Objekt „Simulation“ das Programm im Editor.
2. Setzen Sie den Cursor in die Codezeile `nCounter:=nCoutner+1;` und drücken Sie **[F9]**.

   ⇨ Vor der Codezeile erscheint das Symbol . Es zeigt an, dass an dieser Zeile ein Haltepunkt gesetzt ist. Das Symbol wechselt sofort zu . Der gelbe Pfeil zeigt immer auf die nächste abzuarbeitende Anweisung.
3. Betrachten Sie den Wert der Variablen „nCounter“ im Inline-Monitoring oder im Deklarationsteil des Programms **Simulation**.

   ⇨ Der Variablenwert verändert sich nicht mehr. Die Abarbeitung wurde am Haltepunkt gestoppt.
4. Drücken Sie **[F5]**, was die Abarbeitung wieder startet.

   ⇨ Das Programm stoppt nach einem Zyklus erneut am Haltepunkt. „nCounter“ wurde um 1 hochgezählt.
5. Drücken Sie **[F11]**, um den nächsten Abarbeitungsschritt auszuführen

   ⇨ RETURN am Ende der Zeile `nCounter:=nCounter+1;` Anweisung wird gelb markiert
6. Drücken Sie erneut **[F11]**, um den nächsten Abarbeitungsschritt auszuführen.

   ⇨ Die Abarbeitung springt in den Editor von MAIN. Wiederholtes Drücken von **[F11]** zeigt, wie das Programm Schritt für Schritt ausgeführt wird. Die auszuführende Anweisung wird wieder jeweils mit einem gelben Pfeil gekennzeichnet.
7. Um den Haltepunkt zu deaktivieren und zur normalen Abarbeitung zurückzukehren, setzen Sie den Cursor erneut in die Codezeile und drücken **[F9]**. Drücken Sie dann **[F5]**, um die Programmausführung wieder zu starten.

⇨ Schrittweise könne Sie das Programm durchlaufen und Variablenwerte an bestimmten Codestellen überprüfen.

Ausführen eines Einzelzyklus zur Laufzeit

✓ Das SPS-Programm ist auf die Steuerung geladen und läuft.

1. Beobachten Sie wieder die Zeile `nCounter:=nCounter+1;` im Programm **Simulation**.
2. Klicken Sie auf die Schaltfläche in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen**, um einen Einzelzyklus auszuführen.

   ⇨ Die Abarbeitung durchläuft einen Zyklus und bleibt wieder auf dem Haltepunkt stehen. „nCounter“ wurde um 1 hochgezählt.
3. Drücken Sie weitere Male auf die Schaltfläche, um Einzelzyklen zu sehen. Drücken Sie dann wieder **[F5]**.

⇨ Das Programm läuft wieder ohne Halt und ohne geforcte Werte. Die Variable „tImTemp“ hat wieder den Wert 1 s.

3 Tipps und Tricks

Auf den folgenden Seiten finden Sie einige nützliche Informationen, die bei der Projektierung eines TwinCAT-3-Projekts hilfreich sein können. Diese Tipps und Tricks umfassen die folgenden Punkte:

- Empfohlene Schritte vor der Projektauslieferung [▶ 39]
- Neue Eigenschaften und Features [▶ 43] von TwinCAT 3-Versionen

3.1 Projektauslieferung

Nachdem die Entwicklung eines SPS-Projekts abgeschlossen ist und bevor das Projekt ausgeliefert wird, empfehlen wir, die folgenden Schritte zu prüfen und, wenn sie bezogen auf das Projekt sinnvoll sind, durchzuführen.

Befehl/Schritt	Wo zu finden	Zweck	Weiterführende Informationen
Option „Pin Version“ aktivieren	TwinCAT 3 Projekt > Doppelklick auf den SYSTEM-Knoten > Reiter „Allgemein“	Beim Öffnen des Projekts wird automatisch die im Projekt festgesetzte TwinCAT 3 Engineering-Version verwendet, wenn diese auf dem Engineering-Rechner vorhanden ist. Zur Vermeidung von Sourcecode-Änderungen beim späteren Aktivieren/ Einloggen, welche durch die Verwendung einer anderen Engineering-Version ausgelöst werden könnten	TE1000 XAE\ Technologien\ Remote Manager\ TwinCAT Integration\ Pin Version Projekteinstellung
Compilerversion des SPS-Applikationsprojekts auf „neueste“ setzen	SPS-Projekt > Rechtsklick auf das SPS-Projektobjekt > Befehl „Eigenschaften“ auswählen > Kategorie „Übersetzen“ öffnen > Einstellung „Compilerversion“ auf „neueste“ setzen	Die Compilerversion für Applikationsprojekte wird dann automatisch passend zu der Engineering-Version verwendet, die im Remote Manager geladen ist (s. Punkt 1: Pin Version). Dies ist die vorgesehene Verwendungsweise der Compilerversion.	Kategorie Übersetzen [▶ 951]

Befehl/Schritt	Wo zu finden	Zweck	Weiterführende Informationen
SPS-Bibliotheksreferenzen auf eine feste Version setzen (keine Verwendung von „immer neueste“/„*“)	SPS-Projekt Für alle Bibliotheks-/Platzhalterreferenzen gleichzeitig mit nur einem Befehl: > Rechtsklick auf den References-Knoten > Befehl „Effektive Version verwenden“ auswählen oder für einzelne bzw. für mehrere Bibliotheken/Platzhalter (per Multi-Selektion): > Rechtsklick auf eine Bibliotheksreferenz unterhalb des References-Knotens > Befehl „Effektive Version verwenden“ auswählen oder für einzelne Bibliotheken/Platzhalter: > Bibliotheksreferenz unterhalb des References-Knotens markieren > im Eigenschaftenfenster eine feste Version auswählen oder für einzelne Platzhalter: > Rechtsklick auf den References-Knoten > Platzhalterdialog öffnen > in der Spalte „Bibliothek“ eine feste Version auswählen	Zur Vermeidung einer Online-Change-Abfrage beim späteren Einloggen, welche durch die automatische Verwendung einer neueren Bibliotheksversion ausgelöst werden würde, falls eine Bibliotheksreferenz auf „immer neueste“/„*“ aufgelöst ist und im Bibliotheks-Repository für diese Bibliothek eine neuere Bibliotheksversion verfügbar ist	Empfehlungen und Hinweise [▶ 280]

Befehl/Schritt	Wo zu finden	Zweck	Weiterführende Informationen
Einstellungen des Target-Archivs kontrollieren Hinweis: Falls Sie die Einstellungen ändern sollten, ist es zur Übernahme dieser Einstellungen erforderlich, die Konfiguration oder das Bootprojekt erneut zu aktivieren.	SPS-Projekt > Doppelklick auf das SPS-Projekt > Reiter „Settings“	Zur Vermeidung einer ungewollten Weitergabe der Projekt-Sourcen oder von Bibliotheken durch die Speicherung auf dem Zielsystem • Potenzielle Vorteile, wenn sich die Projekt-Sourcen auf dem Target befinden: ◦ Das im XAE geöffnete SPS-Projekt kann detailliert mit dem Stand auf dem Target verglichen werden (Befehl „Compare Project with Target“). ◦ Das SPS-Projekt kann vom Target geladen und geöffnet werden (Befehl „Open Project from Target“). • Potenzieller Nachteil, wenn sich die Projekt-Sourcen auf dem Target befinden: Wer Zugriff auf das Zielsystem hat, kann die Sourcen lesen/kopieren.	Registerkarte Settings [▶ 969]

Für Serienmaschinen ist außerdem i.d.R. die Verwendung/Aktivierung der folgenden Optionen hilfreich. Die beiden Optionen sind zudem erforderlich, wenn ein gesamtes Maschinen-Update auf Dateiebene durchgeführt werden soll.

Option	Wo zu finden	Zweck	Weiterführende Informationen
Option „Benutze relative NetIds“ in den Routen-Einstellungen	TwinCAT 3 Projekt > SYSTEM-Knoten aufklappen > Doppelklick auf den Routen-Knoten > Reiter „NetId Management“	Wenn die Option „Benutze relative NetIds“ aktiviert ist, dann entsprechen die ersten vier Stellen der Projekt-NetIds einem Platzhalter und werden entsprechend der tatsächlichen NetId des Zielsystems gesetzt. Dadurch kann ein Projekt bzw. eine Konfiguration ohne manuelle Anpassung der Projekt-NetIds auf einem baugleichen anderen Zielsystem verwendet werden. Beispiel: • Zielsystem mit der AMS NetId 172.17.35.70.1.1 • NetId des ersten Geräts (z. B. EtherCAT-Master) ◦ falls die Option „Benutze relative NetIds“ nicht aktiviert ist: 172.17.35.70.2.1 ◦ falls die Option „Benutze relative NetIds“ aktiviert ist: [172.17.35.70].2.1	TE1000 XAE\ System\ System Knoten\ System Unterknoten\ Routen und TE1000 XAE\ System\ Maschinen-Update auf Dateiebene\ Gesamtes Maschinen-Update durchführen
Option „Virtuelle Gerätenamen“ in den Adapter-Einstellungen aller Netzwerk- und USB-Geräte	TwinCAT 3 Projekt > I/O-Knoten aufklappen > Geräte-Knoten aufklappen > Doppelklick auf das Gerät, z. B. den EtherCAT-Master > Reiter „Adapter“	Wenn die Option „Virtuelle Gerätenamen“ aktiviert ist, dann wird der sprechende Name bzw. Anzeigename als Referenz auf das Gerät verwendet. Das System schaut dann auf den Gerätenamen und nicht auf die MAC-Adresse.	TE1000 XAE\ I/O\ EtherCAT\ EtherCAT Master\ Adapter und TE1000 XAE\ System\ Maschinen-Update auf Dateiebene\ Gesamtes Maschinen-Update durchführen

3.2 Neue Eigenschaften und Features

3.2.1 TwinCAT 3.1 Build 4024

Mit TwinCAT 3.1 Build 4024 stehen die folgenden neuen Eigenschaften und Features zur Verfügung.

Kategorie	Feature	Weiterführende Informationen	Verfügbar ab Build-Revision
Entwicklungsunterstützung	Befehl „Gehe zur Definition“ im SPS-Prozessabbild verfügbar	Befehl Gehe zur Definition [▶ 920]	4024.0
	Menübefehl und Tastaturkürzel zum Auskommentieren bzw. Aufheben einer Auskommentierung	Befehl Auswahl auskommentieren [▶ 922] Befehl Auskommentierung der Auswahl aufheben [▶ 922]	4024.0
	Identifizierung von nicht verknüpften allokierten Variablen (AT%I, AT%Q)	siehe unten	4024.10
Speicherformat	Optionales Speicherformat Base64	Befehl Eigenschaften (Objekt) [▶ 942]	4024.0
Compiler	Bedingte Kompilierung - zusätzlich zum Implementierungseditor - nun auch im Deklarationseditor verfügbar	Bedingte Pragmas [▶ 875]	4024.0
Objektorientierte Programmierung	Erweitertes Monitoring einer Schnittstellenvariablen: Symbolpfad und Onlinedaten der aktuell zugewiesenen FB-Instanz werden unter der Schnittstellenvariablen im Monitoring-Bereich angezeigt (Deklarationseditor, Überwachungsliste).	Objekt Schnittstelle [▶ 107]	4024.0
	Mini-Icons zeigen Zugriffsmodifizierer: Mini-Icons auf den Objektsymbolen im Projektbaum stellen dar, ob das Objekt private, public oder protected ist.	Objekt Methode [▶ 93]	4024.0
	Schlüsselwort ABSTRACT	ABSTRACT-Konzept [▶ 212]	4024.0

Kategorie	Feature	Weiterführende Informationen	Verfügbar ab Build-Revision
Online-Features	{attribute 'to_string'} für Enums: Dieses Attribut wird verwendet, um mittels TO_STRING/ TO_WSTRING den Namen eines Enum-Elements textuell abzufragen.	Attribut 'to_string' [▶ 874]	4024.0
	Exception-Handling via __TRY/__CATCH (für 32 Bit Laufzeitsysteme)	__TRY, __CATCH, __FINALLY, __ENDTRY [▶ 774]	4024.0
	Speicherreserve für Online-Change	Speicherreserve für Online-Change konfigurieren [▶ 139]	4024.0
	Details-Button im Meldungsfenster beim Einloggen eines geänderten SPS-Projekts	SPS-Projekt auf der SPS aktualisieren [▶ 262]	4024.0
	Befehl „Gehe zur Instanz“	Befehl Gehe zur Instanz [▶ 921]	4024.0
	Fehleranalyse mit Core Dump	Fehleranalyse mit Core Dump [▶ 259]	4024.11
Sonstiges	Option „Generiere tpy-Datei“	Kategorie Übersetzen [▶ 951]	4024.0

Identifizierung von nicht verknüpften allokierten Variablen (AT%I, AT%Q)

Mit Hilfe der nachfolgend beschriebenen Übersicht können Sie sich einen Überblick über die allokierten Variablen (AT%I, AT%Q) verschaffen, die nicht verknüpft sind. Dadurch können Sie beispielsweise einen Überblick über die Verlinkungen bzw. über den Projektstatus erhalten.

Vorgehen:

1. Doppelklicken Sie auf das Eingangs- oder Ausgangsprozessabbild einer SPS-Task im Projektmappen-Explorer (z.B. auf das Objekt „PlcTask Inputs“ oder „PlcTask Outputs“).
   ⇨ Es öffnet sich eine Übersicht, die die allokierten Ein- bzw. Ausgänge dieses SPS-Prozessabbilds anzeigt (siehe unten, erster Screenshot).
2. Klicken Sie auf den Kopf der zweiten Tabellenspalte (Spaltenüberschrift: „[X]“).
   ⇨ Es werden nur noch die allokierten Ein- bzw. Ausgänge angezeigt, die nicht verknüpft sind (Spaltenüberschrift wechselt auf: „[ ]“) (siehe unten, zweiter Screenshot).
3. Klicken Sie erneut auf den Kopf der zweiten Tabellenspalte.
   ⇨ Es werden wieder alle allokierten Ein- bzw. Ausgänge angezeigt (Spaltenüberschrift wechselt zurück auf: „[X]“).

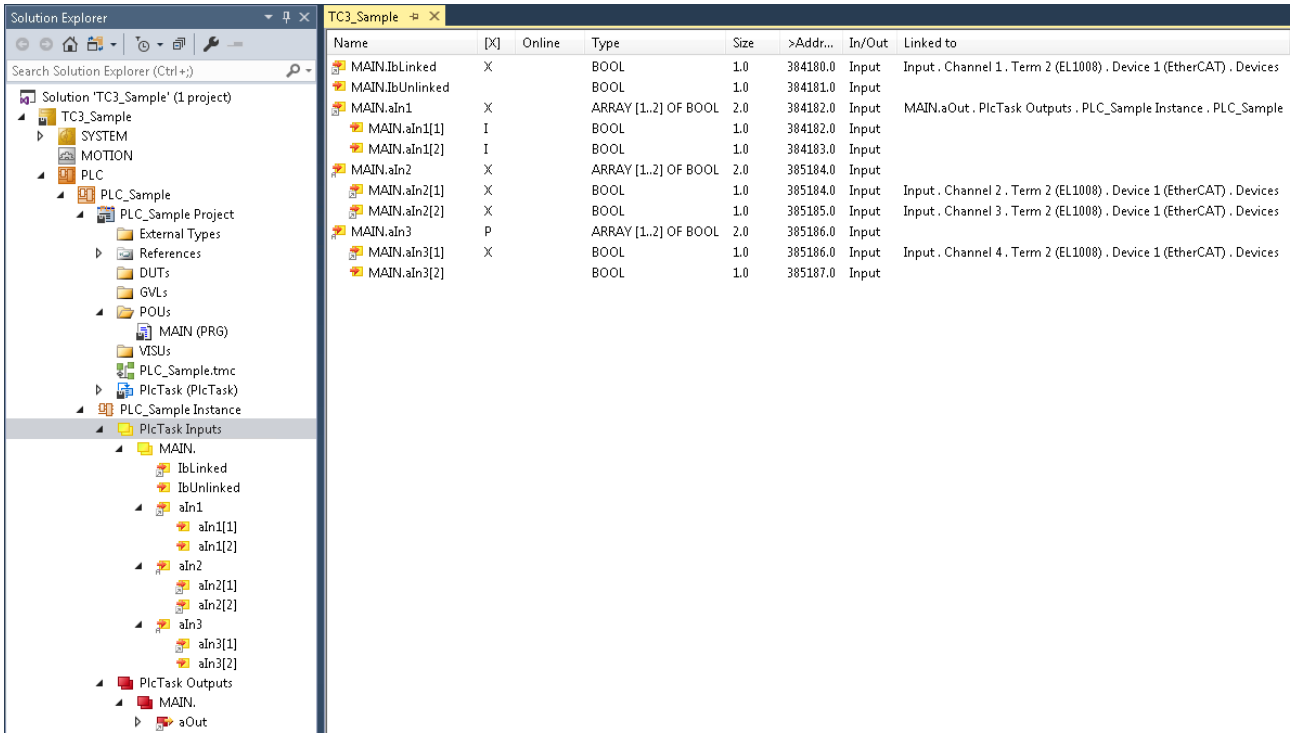
Hinweise:

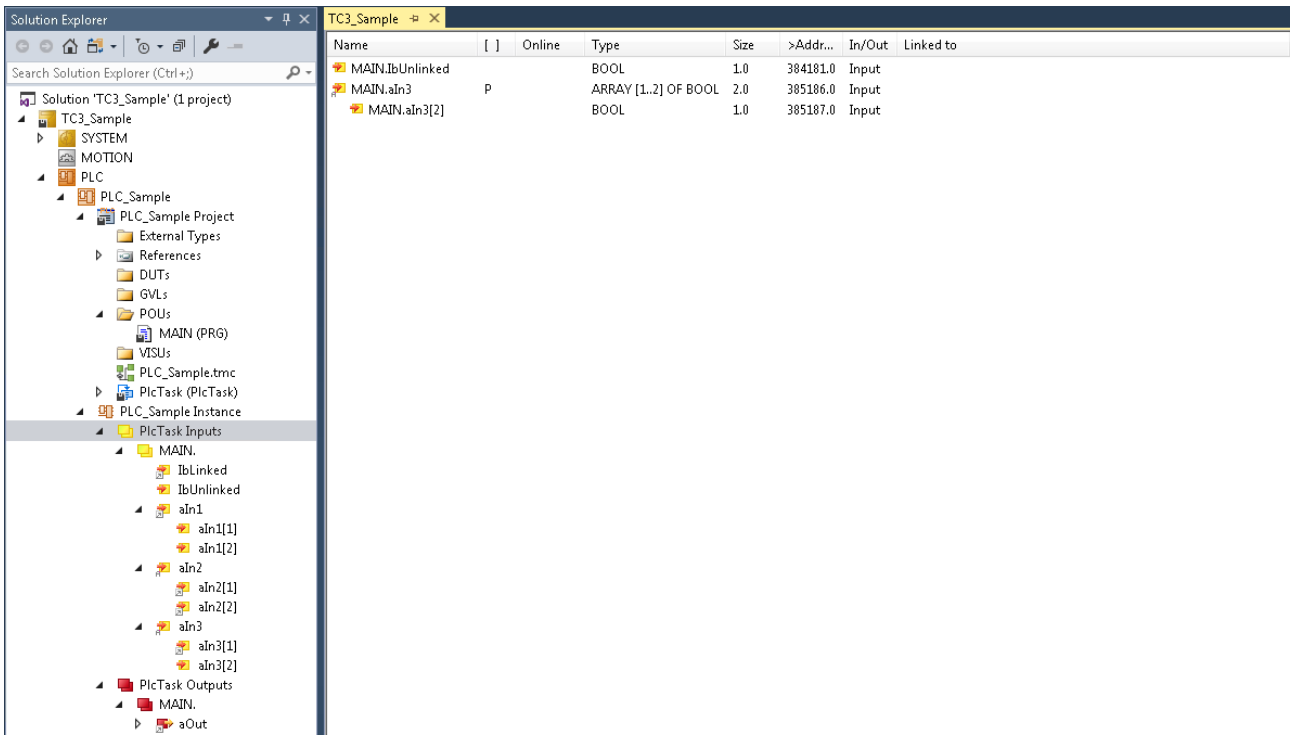
- Mit Hilfe dieser Übersicht können Sie auch neue Links erstellen oder bestehende Links ändern. Klicken Sie hierfür mit der rechten Maustaste auf eine Variable/Zeile und wählen Sie den Befehl **Verknüpfung ändern**.
- Um die Unterelemente einer strukturierten Variablen oder eines Arrays anzuzeigen, klicken Sie auf den Button **Zeige Unterelemente** innerhalb der **TwinCAT XAE Base** Toolbar.

Werte der Spalte „[X]“ bzw. „[ ]“:

Spaltenwert	Bedeutung	Beispiel (siehe *)	Beispielvariable in den Screenshots (s.u.)
Leer	Nicht verknüpft		IbUnlinked
X	Verknüpft	Einzelne verknüpfte Variable	IbLinked MAIN.aIn3[1]
		Array, wobei das Array selbst verknüpft ist	MAIN.aIn1
		Array, wobei alle Elemente des Arrays separat verknüpft sind	MAIN.aIn2
P	Partiell/teilweise verknüpft	Array, wobei mindestens eins, aber nicht alle Elemente des Arrays separat verknüpft sind	MAIN.aIn3
I	Indirekt (über eine Elternvariable) verknüpft	Elemente eines Arrays, wobei das Array selbst verknüpft ist, sodass die Arrayelemente indirekt mit verknüpft sind	MAIN.aIn1[1]

* Die genannten Array-Beispiele sind auch auf strukturierte Variablen (z.B. Strukturinstanz) und ihre Unterelemente übertragbar.





3.2.2 TwinCAT 3.1 Build 4026

Mit TwinCAT 3.1 Build 4026 stehen die folgenden neuen Eigenschaften und Features zur Verfügung.

Kategorie	Feature	Weiterführende Informationen	Verfügbar ab Build-Revision
Entwicklungs-unterstützung	Querverweisliste: Neue Spalte „Kontext“	Befehl Querverweisliste [▶ 984]	4026.0
	Komponenten auflisten: Erweiterung der Funktion um Kategorien und Hervorhebung der eingegebenen Zeichen.	Eingabeunterstützung nutzen [▶ 141]	4026.0
	Smart-Tag-Funktion: Einfache Deklaration von Variablen aus dem Implementierungsteil heraus.	Eingabeunterstützung nutzen [▶ 141]	4026.0
	SPS Lesezeichen	Lesezeichen setzen und verwenden [▶ 166]	4026.0
ST-Editor	Dark Theme (offline)	Dialog Optionen - Texteditor [▶ 1030]	4026.0
	Hervorhebung aller Verwendungsstellen der Variable, auf der der Cursor steht.	ST-Editor [▶ 656]	4026.0
	Inkrementelle Suche	ST-Editor [▶ 656]	4026.0
	Rechteckselektion	ST-Editor [▶ 656]	4026.0
CFC-Editor	Einstellung für die Ausführungsreihenfolge Default: Automatischer Datenfluss-Modus	Befehl Eigenschaften (Objekt) [▶ 946]	4026.0
	Temporäre Anzeige der Ausführungsreihenfolge	Befehl Ausführungsreihenfolge anzeigen [▶ 1061]	4026.0
	Einfacheres Einfügen mehrerer Elemente aus der Toolbox	CFC-Editor [▶ 701]	4026.0
	Drag-and-Drop von Variablen und Objekten aus dem Deklarationsbereich und aus dem Solution Explorer	CFC-Editor [▶ 701]	4026.0
	Änderung der Pin-Anordnung via Drag-and-Drop	CFC-Editor [▶ 701]	4026.0
Objektorientierte Programmierung	Für Inputparameter mit Initialwert müssen bei einem Aufruf keine Variablen übergeben werden.	Methodenaufruf [▶ 210]	4026.0
	IntelliSense für zusätzliche FB_init Parameter	FB_init [▶ 888]	4026.0
Online-Features	Strukturierte Darstellung vererbter Variablen	Befehl Darstellung Vererbung – Einfach, Strukturiert [▶ 1008]	4026.0
	Befehle „Gehe zur Implementierung“ und „Gehe zur Referenz“	Befehl Gehe zur Implementierung [▶ 921] Befehl Gehe zum Verweis [▶ 921]	4026.0
	Filtermöglichkeit im Dialog „Onlinestatus auswählen“	Monitoring in Programmierobjekten aufrufen [▶ 234]	4026.0

Kategorie	Feature	Weiterführende Informationen	Verfügbar ab Build-Revision
Bibliotheken	Verwendung eines SPS-Projekts als referenzierte Bibliothek.	SPS-Projekt als referenzierte Bibliothek verwenden	4026.0
	Neue Optionen für Bibliotheken in den Eigenschaften des SPS-Projekts: „Implizite Prüfungen für Compiled Libraries erlauben“ und „Qualified_only für Bibliothekszugriff erzwingen“.	Kategorie Common [▶ 948]	4026.0
	Verbesserung der Navigation im Bibliotheksverwalter durch Verlinkungen.	Bibliotheksverwalter [▶ 289]	4026.0
	Möglichkeit, Repositories im Bibliotheksverwalter zu aktivieren und zu deaktivieren.	Bibliotheksrepository [▶ 285]	4026.0
	Unterscheidung der Befehle zum Hinzufügen von Platzhaltern und von Bibliotheken.	Befehl Bibliothek ohne Platzhalterauflösung hinzufügen [▶ 374]	4026.0
	Mehrfachauswahl im Dialog „Bibliothek hinzufügen“	Befehl Bibliothek hinzufügen [▶ 373]	4026.0
	Export der Werte einer Parameterliste als .csv Datei	Objekt Parameterliste [▶ 75]	4026.0
Operatoren und Variablen	Ermittlung des Namens lokaler Objekte mit dem Operator __POUNAME und der Zeilennummer mit dem Operator __POSITION.	__POUNAME [▶ 779] __POSITION [▶ 779]	4026.0
	Generische Konstanten	Generische konstante Variablen - VAR_GENERIC CONSTANT [▶ 725]	4026.0
Sonstiges	Repräsentation des Typsystems: Bibliothek Tc3_Global_Types ExternalTypes.tmc statt des External Types Ordners	Globale Datentypen verwenden [▶ 63]	4026.0
	Speichern von SPS-Projekten als wiederverwendbare Projekt-Templates.	Befehl Als SPS-Projektvorlage speichern	4026.0

3.3 Weitere hilfreiche Eigenschaften und Features

Zusätzlich zu den Eigenschaften und Features, die mit einer bestimmten TC3.1-Version zur Verfügung gestellt werden, beinhaltet die Engineering-Umgebung u.a. die folgenden hilfreichen Eigenschaften und Features.

Kategorie	Feature	Weiterführende Informationen
Entwicklungsunterstützung	Tastaturkürzel: Für viele Funktionen stehen Tastaturkürzel zur Verfügung, die bei Bedarf individuell angepasst werden können.	Tastaturkürzel [▶ 52]
	Option „Ordneransicht aktivieren“: Für eine optionale Ordneransicht im PLC-Prozessabbild und im Link-Dialog	siehe unten
	Refactoring: Zur komfortablen, nachträglichen Änderung eines Programms	Refactoring [▶ 168]
	Objektorientierte Programmiermöglichkeiten: Kann zur Erstellung von leicht les- und erweiterbarer Software verwendet werden	Objektorientiert programmieren [▶ 178]
Orientieren und Navigieren	Verwendungsstellen mit der Querverweisliste finden	Verwendungsstellen mit der Querverweisliste finden [▶ 165]
	Deklaration finden	Deklaration finden [▶ 166]
Online-Features	Implizite Überwachungsfunktionen: z.B. um während der Laufzeit die Grenzen von Arrays zu überprüfen	Bausteine für implizite Prüfung verwenden [▶ 172]
	Ablaufkontrolle: Zur Verfolgung der Programmabarbeitung	Ablaufkontrolle [▶ 231]
	Überwachungslisten: Zum übersichtlichen Monitoring von Variablenwerten	Überwachungslisten verwenden [▶ 239]
	Haltepunkte und schrittweises Abarbeitung des Programms: Zur Fehlersuche im Programm	Haltepunkte verwenden [▶ 222] Schrittweises Abarbeiten eines Programms (Stepping) [▶ 224]
Bibliotheken	Bibliotheken verwenden: Für die Erstellung und Verwendung von Sammlungen wiederverwendbarer Objekte	Bibliotheken verwenden [▶ 278]
	Bibliotheksplatzhalter: Zur komfortablen Auswahl einer bestimmten Bibliotheksversion für einen Platzhalter, der direkt oder indirekt in dem SPS-Projekt referenziert wird	Bibliotheksplatzhalter [▶ 293]
	Befehl „Effektive Version verwenden“: Zum Festsetzen einer Bibliotheksreferenz auf eine bestimmte Bibliotheksversion	Befehl Effektive Version verwenden [▶ 380] Projektauslieferung [▶ 39]

Optionale Ordneransicht im PLC-Prozessabbild und im Link-Dialog

Option: Checkbox **Ordneransicht aktivieren**

- Aktiviert: Die Variablen im SPS-Prozessabbild und im SPS-Bereich des Link-Dialogs werden als aufklappbare Ordnerebenen angezeigt.

- Deaktiviert: Die Variablen im SPS-Prozessabbild und im SPS-Bereich des Link-Dialogs werden als flache Liste angezeigt.

Dialog: Doppelklick auf den SPS-Knoten im **Projektmappen-Explorer** > Registerkarte **SPS Einstellungen**

Beispiel:

Funktionsbaustein FB_Sample

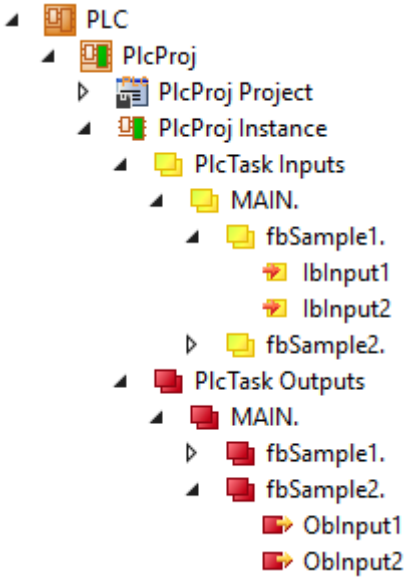
```
FUNCTION_BLOCK FB_Sample
VAR
    IbInput1  AT%I* : BOOL;
    IbInput2  AT%I* : BOOL;
    ObInput1  AT%Q* : BOOL;
    ObInput2  AT%Q* : BOOL;
END_VAR
```

Programm MAIN

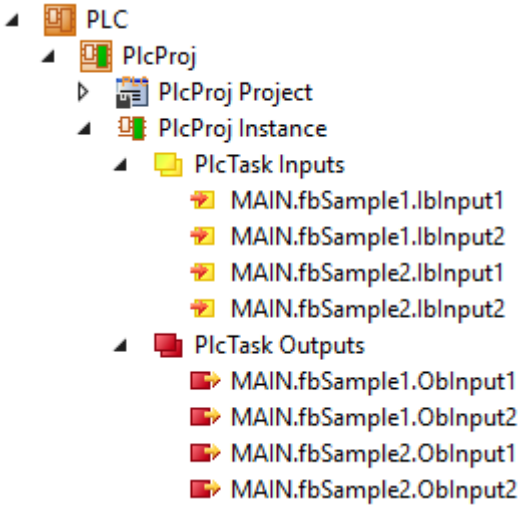
```
PROGRAM MAIN
VAR
    fbSample1       : FB_Sample;
    fbSample2       : FB_Sample;
END_VAR
```

PLC-Prozessabbild:

- Fall 1: Checkbox **Ordneransicht aktivieren** ist aktiviert.

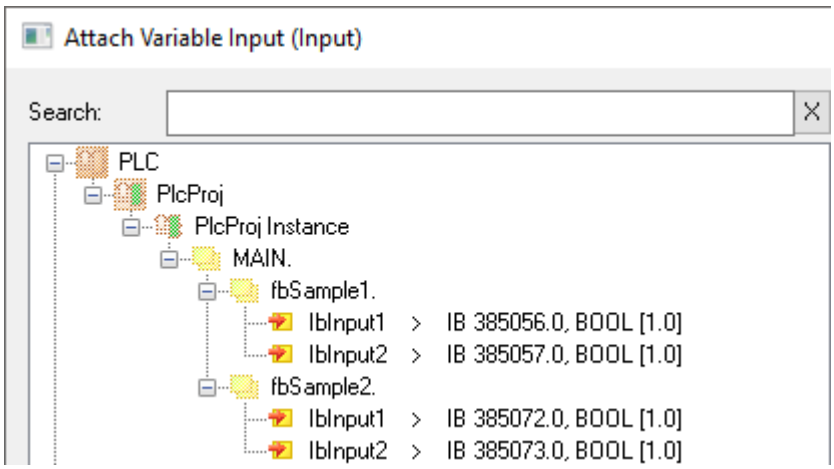


- Fall 2: Checkbox **Ordneransicht aktivieren** ist deaktiviert.

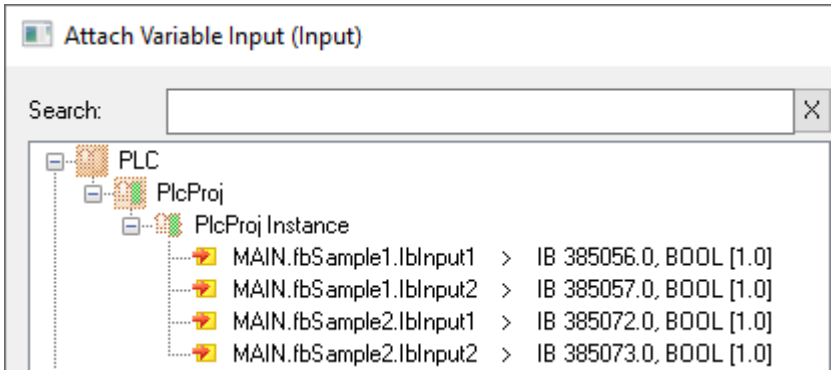


Link-Dialog:

- Fall 1: Checkbox **Ordneransicht aktivieren** ist aktiviert



- Fall 2: Checkbox **Ordneransicht aktivieren** ist deaktiviert



3.3.1 Tastaturkürzel

In der TwinCAT 3 Entwicklungsumgebung sind einige Befehle per Tastaturkürzel ausführbar. Nachfolgend finden Sie eine Übersicht dieser Befehle mit den standardmäßig eingestellten Tastaturkürzeln. Bei Bedarf können Sie die Konfiguration der Tastaturkürzel anpassen und/oder erweitern.

Kategorie	Befehl	Standardmäßiges Tastaturkürzel	Weiterführende Informationen
Standardbefehle	Rückgängig	Strg+Z	Standardbefehle [▶ 912]
	Wiederholen	Strg+Y	
	Kopieren	Strg+C	
	Ausschneiden	Strg+X	
	Einfügen	Strg+V	
	Alles auswählen	Strg+A	Befehl Alles auswählen [▶ 913]
	Löschen	Entf	Befehl Entfernen [▶ 913]
Entwicklungsunterstützung	Gehe zur Definition	F12	Befehl Gehe zur Definition [▶ 920]
	Verweise suchen	Umschalt+F12	Befehl Verweise suchen [▶ 921]
	Eingabehilfe	F2	Befehl Eingabehilfe [▶ 913]
	Fokus zwischen Deklarations- und Implementierungseditor wechseln	F6	
Kommentierung	Auskommentieren	[Ctrl+K] + [Ctrl+C]	Befehl Auswahl auskommentieren [▶ 922]
	Auskommentierung aufheben	[Ctrl+K] + [Ctrl+U]	Befehl Auskommentierung der Auswahl aufheben [▶ 922]
Online-Befehle	Start	F5	Befehl Start [▶ 1003]
	Stop	Umschalt+F5	Befehl Stop [▶ 1003]

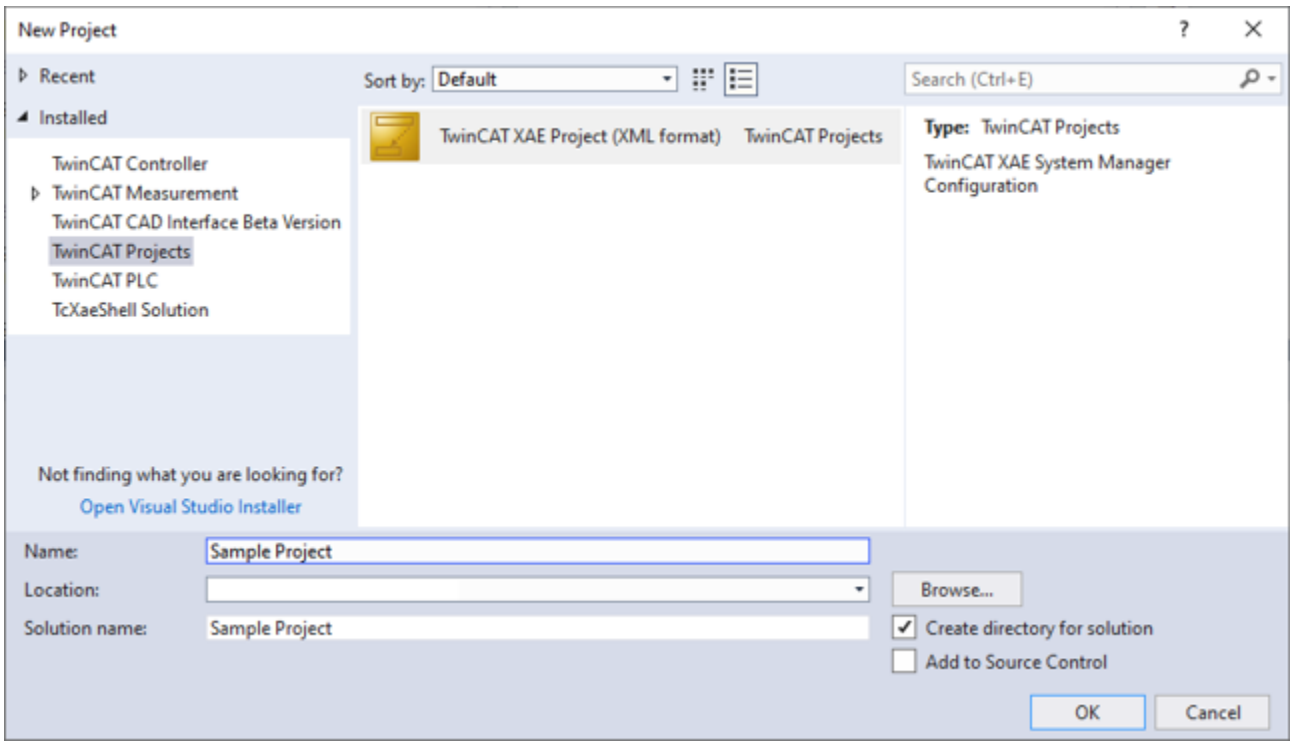
3.4 Umbenennen eines Projekts

Szenario

Ein bestehendes Projekt soll für eine neue Maschine wiederverwendet werden. Dazu soll das Projekt den Namen der neuen Maschine erhalten.

Herausforderung

Beim Anlegen eines Projekts wird abgefragt, ob ein Projektmappen-Verzeichnis erstellt werden soll. Diese Option ist das Default-Verhalten. Ist sie ausgewählt, wird das TwinCAT-3-Projekt in diesem Verzeichnis angelegt.



In der Projektmappen-Datei (*.sln) des Projekts ist ein Verweis auf alle Unterprojekte enthalten. Wird ein TwinCAT-3-Projekt umbenannt, erfolgt nur eine Umbenennung der Projektdatei, nicht aber des Ordners. Somit wäre nach einer Umbenennung des Projekts auf der Festplatte immer noch der alte Ordnername des TwinCAT-3-XAE-Projekts vorhanden, auch wenn dieser nicht in der Projektmappe steht.

Lösungsmöglichkeiten

- Wählen Sie generische Ordnernamen, die keinen Projektbezug haben.
- Deaktivieren Sie die Option **Create directory for solution**, dann wird die Projektmappen-Datei neben der Projektdatei des TwinCAT-3-XAE-Projekts abgelegt.
- Ist nur ein TwinCAT-3-Projekt in der Projektmappen-Datei enthalten, dann können Sie das TwinCAT-3-Projekt manuell ohne die *.sln-Datei und den Ordner, in dem das TwinCAT 3-Projekt abgelegt ist, in einen anderen Ordner kopieren. Durch Doppelklicken auf die *.tsproj-Datei öffnet sich ebenfalls die TwinCAT 3 XAE Shell. Sie können das Projekt in der XAE Shell entsprechend umbenennen. Beim Speichern des Projekts wird automatisch gefragt, ob eine neue Projektmappen-Datei erstellt werden soll.
- Kopieren Sie das gesamte Verzeichnis und passen Sie den Ordnernamen der Unterprojekte manuell an. Anschließend müssen diese Ordnernamen ebenfalls manuell in der *.sln-Datei getauscht werden.

4 SPS-Projekt anlegen und konfigurieren

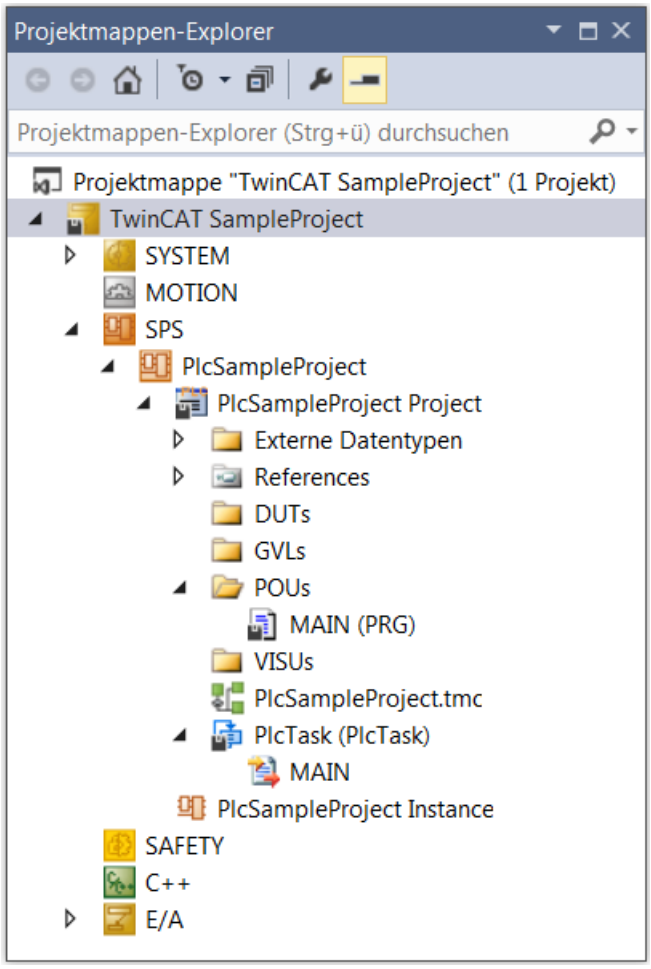
Was ist ein SPS-Projekt?

- Ein Projekt enthält die Objekte, die zur Erstellung eines Steuerungsprogramms nötig sind:
  - Reine Programmierbausteine, beispielsweise Programme, Funktionsbausteine, Funktionen, GVLs.
  - Objekte, die zusätzlich benötigt werden, um das Programm auf einer SPS ausführen zu können. Beispielsweise referenzierte Tasks, Bibliotheksverwalter und Visualisierungen.
- In einem TwinCAT-Projekt können Sie mehrere SPS-Projekte programmieren und auf einem Zielgerät ausführen.
- Projektspezifische Bausteine verwaltet TwinCAT in der Ansicht Projektmappen-Explorer unter dem SPS-Knoten.
- Für das Erstellen von Projekten gibt es Vorlagen, die bereits bestimmte Objekte enthalten.
- In den Projekteinstellungen und Projektinformationen sind Grundkonfigurationen und Informationen zum Projekt definiert. Beispielsweise:
  - Compiler-Einstellungen
  - Autor
- Sie legen ein Projekt als Datei im Dateisystem ab. Optional können Sie es zusammen mit projektrelevanten Dateien und Informationen in einem Projektarchiv verpacken. Möglich ist auch die Ablage in einem Quellcodeverwaltungssystem wie Microsoft Team Foundation Server (TFS) oder SVN.
- Jedes Projekt enthält die Information, mit welcher TwinCAT-Version es erstellt wurde. Wenn Sie es in einer anderen Version öffnen, weist TwinCAT auf mögliche oder nötige Aktualisierungen hin.
- Sie können Projekte vergleichen und einzelne Objekte exportieren, importieren.
- Sie können ein Projekt gegen Veränderung und komplett, auch gegen Lesen schützen. Durch das Verwenden einer Benutzerverwaltung können Sie den Zugriff auf das Projekt und sogar auf einzelne Objekte im Projekt gezielt steuern.

4.1 Standardprojekt anlegen

1. Wählen Sie im Menü **Datei** den Befehl **Neu > Projekt**.
   ⇨ Der Dialog **Neues Projekt** öffnet sich.
2. Wählen Sie die Vorlage **TwinCAT Projekte > TwinCAT XAE Projekt** und geben Sie einen Namen, hier als Beispiel „SampleProject“, und einen Ablageort im Dateisystem ein.
3. Beenden Sie den Dialog mit **OK**.
   ⇨ Im **Projektmappen-Explorer** öffnet sich eine neue Projektmappe. In der Titelleiste des Hauptfensters erscheint der Projektname „TwinCAT SampleProject“.
4. Markieren Sie im Projektbaum das SPS-Objekt und wählen Sie im Menü **Projekt** oder im Kontextmenü den Befehl **Neues Element hinzufügen…** aus.
   ⇨ Der Dialog **Neues Element hinzufügen <TwinCAT Projektname>** öffnet sich.
5. Wählen Sie in der Kategorie **Plc Templates** das **Standard PLC Projekt** aus und geben Sie einen Namen ein (hier auch „SampleProject“).
6. Beenden Sie den Dialog mit **Hinzufügen**.

⇨ In der Ansicht **Projektmappen-Explorer** ist folgende Struktur angelegt:



⇨ Mit dem gewählten Template erscheinen unterhalb des SPS-Projektobjekts (**PlcSampleProject**) automatisch folgende Basisobjekte: ein SPS-Projekt (**PlcSampleProject Project**) und eine Projektinstanz (**PlcSampleProject Instance**). Das SPS-Projekt enthält einen Bibliotheksverwalter (**References**), den Standardprogrammbaustein **MAIN** und eine Taskreferenz (**PlcTask**). Der dort referenzierte Task (**PlcTask**) definiert die Abarbeitung des Programmbausteins MAIN. Zudem erscheinen automatisch die Strukturordner **Externe Datentypen**, **DUTS**, **GVLs**, **POUS** und **VISUs**. Der Bibliotheksverwalter enthält bereits die Standardbibliotheken mit Basisbausteinen wie Zählern, Timern, String-Funktionen, die nachher zur Programmierung verwendet werden können.
Wenn Sie nun MAIN mit fehlerfreiem Code füllen, können Sie diesen ohne weitere Programmierobjekte zu benötigen auf die Steuerung laden und zur Ausführung bringen.

Siehe auch:

- SPS-Projekt programmieren [▶ 68]
- Bibliotheken verwenden [▶ 278]

4.2 Objekte hinzufügen

Die folgende Anleitung zeigt einige Möglichkeiten beim Anlegen von Objekten im SPS-Projekt.

✓ Ein SPS-Projekt ist geöffnet.

1. Selektieren Sie einen Eintrag im SPS-Projektbaum, beispielsweise den Ordner **POUs**.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen**.
   ⇨ Abhängig vom ausgewählten Eintrag im Baum bietet TwinCAT die passenden Objekte zur Auswahl an. Folgende Objekte stehen Ihnen zur Verfügung:

- Objekt POU [▶ 80]
  - Objekt Methode [▶ 93]

    - Objekt Eigenschaft [▶ 100]
    - Objekt Aktion [▶ 89]
    - Objekt Transition [▶ 90]
  - Objekt POUs für implizite Prüfungen [▶ 172]
  - Objekt DUT [▶ 77]
  - Objekt Globale Variablenliste [▶ 75]
  - Objekt Taskreferenz [▶ 137]
  - Objekt Rezepturverwalter [▶ 240]
    - Objekt Rezepturdefinition [▶ 244]
  - Objekt Bildersammlung [▶ 153]
  - Objekt Schnittstelle [▶ 107]
  - Objekt Parameterliste [▶ 75]
  - Objekt "Textliste" [▶ 151]
  - Objekt "Class Diagram [▶ 139]"
  - Objekt "Visualization" Manager" [▶ 405]
  - Objekt "Visualization" [▶ 422]
3. Wählen Sie beispielsweise das Objekt **POU** und im daraufhin erscheinenden Dialog **POU hinzufügen** den Typ **Programm** mit Implementierungssprache „Strukturierter Text (ST)“ und Namen „Prog“ aus. Klicken Sie auf **Öffnen**.
   ⇨ TwinCAT fügt ein Programm-Objekt **Prog** im SPS-Projektbaum unterhalb des gewählten Eintrags ein.
4. Selektieren Sie ein Objekt im Baum und wählen im Kontextmenü den Befehl **Eigenschaften** (Objekt).
   ⇨ Die Ansicht **Eigenschaften** mit objektrelevante Kategorien wird aktiv. Wenn Sie eine Benutzerverwaltung verwenden, könnten Sie hier beispielsweise den Zugriff auf das Objekt einschränken.
5. Selektieren Sie einen Eintrag im Baum, unterhalb dessen Sie einen Ordner anlegen möchten, um in ihm bestimmte Objekte zu sammeln.
6. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Neuer Ordner.**
   ⇨ Der Ordner erscheint im SPS-Projektbaum.
7. Geben Sie dem Ordner einen Namen und bestätigen Sie.
8. Selektieren Sie ein Objekt im SPS-Projektbaum und verschieben Sie es durch Ziehen mit der Maus innerhalb des Projektbaums an eine andere Position, beispielsweise in den Ordner.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Vorhandenes Elements hinzufügen (Objekt)
- Dokumentation TC3 User Interface: Befehl Eigenschaften (Objekt) [▶ 942]
- Dokumentation TC3 User Interface: Befehl Ordner hinzufügen

4.3 Compilerversion ändern

Die Version des Compilers, mit der im aktuellen Projekt Code für die Verwendung auf dem Zielgerät erzeugt wird, ist in den Projekteigenschaften definiert.

Die Compilerversion ist unabhängig von der TwinCAT-Version. Also wird bei gleicher eingestellter Compilerversion aus dem Quellcode konstanter Programmcode erzeugt, auch wenn dies aus unterschiedlichen TwinCAT-Versionen heraus erfolgt.

Wenn Sie ein Projekt öffnen, in dem nicht die neueste Compilerversion eingestellt ist, erscheint der Dialog **Projektumgebung** mit einem entsprechenden Hinweis und der Möglichkeit, direkt zu aktualisieren.

✓ Ein SPS-Projekt ist geöffnet.

1. Markieren Sie das SPS-Projekt und wählen Sie im Menü **Projekt** oder im Kontextmenü den Befehl **Eigenschaften**.
   ⇨ Die SPS-Projekteigenschaften öffnen sich in einem Editorfenster.
2. Wählen Sie die Kategorie **Übersetzen**.
3. Wählen Sie im Gruppenfeld **Solution options** die gewünschte feste Version.

⇨ Die Änderung ist sofort wirksam.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Eigenschaften (Projekt) > Kategorie Übersetzen [▶ 951]

4.4 TwinCAT-3-SPS-Projekt öffnen

✓ TwinCAT XAE ist gestartet. Ein TwinCAT Projekt ist geöffnet.

1. Markieren Sie das SPS-Objekt in der Ansicht **Projektmappen-Explorer** und wählen Sie im Menü **Projekt** oder im Kontextmenü den Befehl **Vorhandenes Element hinzufügen.**
2. Wählen Sie im Dialog **Öffnen** das gewünschte TwinCAT-3-SPS-Projekt oder -Projektarchiv oder auch eine Bibliothek aus dem Dateisystem. Zur Suche können Sie den Dateifilter in der rechten unteren Ecke des Dialogs setzen.
3. Klicken Sie auf **Öffnen**.

⇨ Folgende Fälle sind möglich:

- Sie haben ein Projekt ausgewählt, das mit einer neueren TwinCAT-Version gespeichert wurde:
  Ein solches Projekt enthält eventuell Daten, welche nicht geladen werden können. Sie können das Projekt dennoch öffnen, müssen jedoch Folgendes beachten: Das Projekt kann ein unerwartetes Verhalten zeigen, weil es nicht vollständig interpretiert werden kann. Objekte, welche TwinCAT 3 nicht oder nur unvollständig laden konnte, werden in der Ansicht **Projektmappen-Explorer** rot und mit dem Text „[unbekannt]“ oder „[unvollständig]“ gekennzeichnet. Objekte, die der aktuellen Version unbekannt sind, kann TwinCAT nicht im Editor anzeigen. Unvollständige Objekte zeigt TwinCAT im Editor mit einer Warnung an, dass der angezeigte Inhalt möglicherweise nicht dem Original entspricht. Unvollständige Projekte kann TwinCAT nicht unter ihrem Originalnamen abspeichern. Dies wird durch einen Schreibschutz-Hinweis in der oberen rechten Ecke angezeigt. Sie können die Datei unter einem anderen Namen abspeichern.
- Sie haben ein Projekt ausgewählt, nach dessen letzter Änderung TwinCAT nicht regulär beendet wurde, aber die Projektoption **Automatisch Speichern** aktiviert war:
  Sie erhalten nun den Dialog **Auto Save Backup** zur Handhabung der Sicherungskopie.

Siehe auch:

- TwinCAT-2-SPS-Projekt öffnen [▶ 58]
- Dokumentation TC3 User Interface: Befehl Projekt/Projektmappe (Projekt/Projektmappe öffnen) [▶ 901]
- Dokumentation TC3 User Interface: Befehl Vorhandenes Element hinzufügen (Projekt) [▶ 903]

4.5 TwinCAT-2-SPS-Projekt öffnen

✓ TwinCAT XAE ist gestartet. Ein TwinCAT-Projekt ist geöffnet. Ihnen sollten die unterhalb der folgenden Anleitung beschriebenen Einschränkungen bewusst sein.

1. Markieren Sie das SPS-Objekt in der Ansicht **Projektmappen-Explorer** und wählen Sie im Menü **Projekt** oder im Kontextmenü den Befehl **Vorhandenes Element hinzufügen**.
2. Wählen Sie im Dialog **Öffnen** das gewünschte Plc 2.x-Projekt oder Bibliothek aus dem Dateisystem. Zur Suche können Sie den Dateifilter in der rechten unteren Ecke des Dialogs setzen.
   ⇨ Danach startet automatisch der TwinCAT 2.x-Konverter.

3. Der TwinCAT 2.x-Konverter prüft, ob das Projekt fehlerfrei kompilierbar ist. Wenn ja, bearbeitet er das Projekt automatisch.
   Wenn das Projekt Visualisierungsobjekte mit Platzhaltervariablen enthält, die der Konverter nicht auflösen kann, erfolgt ein direktes Einbinden der jeweiligen Visualisierungen als Gruppierung an Stelle der Visualisierungsreferenzen.
4. Bibliothekskonvertierung: Wenn im zu öffnenden Projekt eine Bibliothek referenziert ist, für die noch keine Konvertierungsregel definiert ist, erscheint der Dialog **Konvertierung einer Bibliotheksreferenz**. Legen Sie hier fest, ob und wie der Konverter die bisherige Bibliotheksreferenz durch eine aktuelle ersetzen soll. Falls Sie dabei eine Bibliothek auswählen, für die Projektinformationen fehlen, erscheint der Dialog **Projektinformationen**, den Sie ausfüllen müssen.

⇨ Der Konverter lädt das angepasste Projekt.

Einschränkungen beim Weiterverwenden eines TwinCAT-2.x-Projekts in TwinCAT 3.1

Automatische Syntaxanpassungen nur bei vorheriger Übersetzbarkeit

Zwischen TwinCAT 2 (TC2) und TwinCAT 3 (TC3) unterscheidet sich die Syntax an einigen Stellen (z. B. bei der Initialisierung von Arrays). Bitte beachten Sie, dass der Konverter die Syntax an diesen Codestellen nur dann anpasst, wenn das zu konvertierende TC2-Projekt „TC2-seitig“ kompiliert werden kann.

Im Detail bedeutet das:

Bei dem Konvertierungsprozess übersetzt der Konverter, der in TC3 enthalten ist, das ausgewählte TC2-Projekt zunächst mit einem TC2-Compiler. Nur wenn diese „TC2-seitige“ Übersetzung erfolgreich ist, werden bei der Erstellung des TC3-SPS-Projekts Syntaxanpassungen des TC2-Codes vorgenommen. Voraussetzung für eine erfolgreiche „TC2-seitige“ Übersetzung ist, dass dem Konverter alle benötigten TC2-Bibliotheken zur Verfügung gestellt werden. Dies kann z. B. erreicht werden, indem die Bibliotheken, die im TC2-Projekt referenziert werden, in den Ordner C:\TwinCAT\3.1\Components\Plc\Converter\Lib eingefügt werden, oder indem dem Konverter die Speicherorte der TC2-Bibliotheken über die angezeigten Konvertierungsdialoge mitgeteilt werden.

Kompilierung	Das Projekt muss ohne Übersetzungsfehler in TwinCAT 2.x PLC Control kompilierbar sein. TwinCAT gibt bei der Übersetzung dennoch Warnungen aus. Diese werden durch implizite Konvertierungen hervorgerufen, die zu Informationsverlust führen können (beispielsweise durch Vorzeichenwechsel). TwinCAT 3.1 prüft case-Anweisungen gegen die Switch-Variable: CASE USINT OF INT wird in TwinCAT 2.x nicht überprüft, es gibt jedoch eine Fehlermeldung beim Import in TwinCAT 3.1.
Bibliotheken	Alle Variablen und Konstanten, die in einer Bibliothek verwendet werden, müssen auch in dieser Bibliothek deklariert sein. Die Bibliothek muss in TwinCAT 2.x fehlerfrei übersetzbar sein.

Syntaktische und semantische Einschränkungen	• FUNCTIONBLOCK ist nicht länger ein gültiges Schlüsselwort anstelle von FUNCTION_BLOCK
	• Nach TYPE (Deklaration einer Struktur) muss ein ":" folgen.
	• ARRAY-Initialisierung muss mit runden Klammern versehen sein.
	• Eine lokale Deklaration einer Enumeration ist nicht länger möglich, außer innerhalb von TYPE; bei einem TwinCAT 2.x-Import wird eine lokale Enumerationsdeklaration automatisch in eine explizite Type-Definition konvertiert.
	• INI wird nicht mehr unterstützt (Sie müssen dies im Code durch die Init-Methode ersetzen).
	• In Funktionsaufrufen ist es nicht länger möglich, explizite Parameterzuweisungen mit impliziten zu mischen. Deshalb kann die Reihenfolge der Parametereingangs-Zuweisungen verändert werden:
	`fun(formal1 := actual1, actual2); // → Fehlermeldung`
	`fun(formal2 := actual2, formal1 := actual1); // gleiche Semantik wie folgende Zeile:`
	`fun(formal1 := actual1, formal2 := actual2);`
	• TwinCAT 2.x-Pragmas werden nicht konvertiert. Sie erzeugen in TwinCAT 3.1 eine Warnung.
	• Der TRUNC-Operator konvertiert nun in den Datentyp DINT, anstelle von INT; bei einem TwinCAT 2.x-Import fügt TwinCAT 3.1 automatisch eine entsprechende Typkonvertierung hinzu.

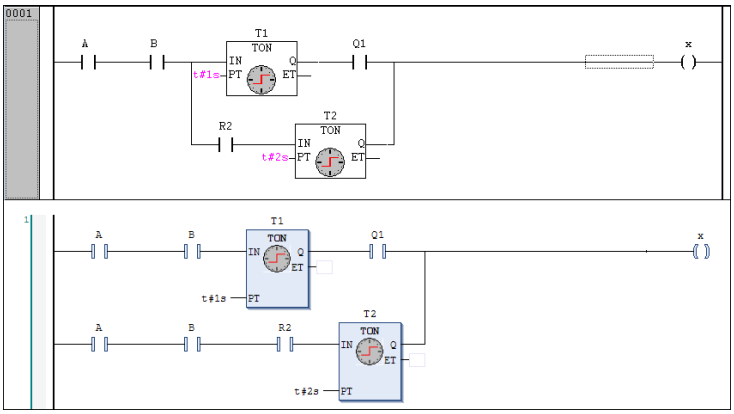
Speicher	• TwinCAT 3.1 besitzt, anders als TwinCAT 2, ein 8-Byte-Alignment. Für weitere Details und Beispiele siehe Referenz Programmierung > Alignment [▶ 826]. • Instanzen vom Datentyp STRING werden in TwinCAT 2 vorinitialisiert, indem der komplette Speicherbereich genullt wird. In TwinCAT 3.1 wird dagegen nur das erste Byte einer solchen Instanz genullt. Aufgrund der Nullterminierung ist die Instanz ebenso leer.

Visualisierung

Platzhalter und deren Ersetzung	Platzhalter	VAR_INPUT	Verwendung	Ersetzung
	MAIN. $LocalVar$.aArr[0]	localVar: MyStruct;	localVar.aArr[0]	localVar := MAIN.myStructVar
	Var.aArr[0]	Var : MyStruct;	Var.aArr[0]	Var := MAIN.myStructVar
	MAIN.myStructVar. aArr[$Index$]	Index : INT;	MAIN.myStructVar. aArr[Index]	Index := 0

Problematische Platzhalter	• Platzhalter innerhalb eines Textes:
	Text: $axle$-Axis
	Behebung:
	`localVar : STRING;`
	Text: %s-Axis
	`TextVariable : localVar`
	• Platzhalter beschreibt nur einen Teil eines Variablennamens:
	axis$axis$spur$spur$.fActPosition
	Behebung:
	Definieren Sie für den Platzhalter axis$axis$spur$spur$ nur einen Platzhalter.
	`axis_spur : MyFunctionBlock;`
	Übergeben Sie dann direkt die entsprechende Instanz des Funktionsbausteins.
	`axis_spur := MAIN.axis1spur2;`
	• Platzhalter wird durch einen Ausdruck ersetzt:
	$Expression$ → MAIN.var1 + MAIN.var2
	Behebung:
	Sie müssen den Ausdruck an eine Hilfsvariable übergeben und diese Hilfsvariable dann als Instanz übergeben.
	• Platzhalter beschreibt einen Programmnamen: $Program$.bToggle → MAIN.bToggle D
	Diese Form der Platzhalterersetzung kann der Konverter nicht in TwinCAT 3.1 übertragen. Sie werden sie in der Praxis aber selten verwenden.
	• Platzhalter wird durch verschiedene Typen ersetzt:
	Var
	→ Ersetzung 1 : MAIN.n (INT)
	→ Ersetzung 2 : MAIN.st (STRING)
	Behebung:
	Definieren Sie hierfür zwei verschiedene Platzhalter in der Schnittstelle.
	• Die Visualisierung liegt in einer Bibliothek. Sie ersetzen die Platzhalter erst später aus einem beliebigen Projekt, wenn Sie die Visualisierung dort verwenden.
	Behebung:
	Hier müssen Sie die TYPE_NONE-Datentypen manuell ersetzen. Es gibt aber auch die Möglichkeit, dass Sie die Bibliothek in ein Projekt einbinden und der Platzhalter richtig ersetzt wird. Wenn Sie nun dieses Projekt importieren, dann wird der Datentyp auch in der Bibliothek richtig ermittelt.			

Nicht importierbare Elemente	Trend, ActiveX - Der Import ist nicht möglich, weil die Implementierung sehr unterschiedlich ist. In TwinCAT 3.1 wird eine entsprechende Warnung ausgegeben und ein entsprechender manueller Nachbau ist nötig.			

Programmiersprachen	ST, AWL, FUP	Keine Einschränkungen
	KOP	TwinCAT 3.1 importiert Funktionsbausteine mit Parallelverzweigungen so, dass der Teil vor der Verzweigung für jeden Zweig wiederholt wird. Das entspricht dem generierten Code, den TwinCAT 2.x für Parallelverzweigungen erzeugt. 
	AS	• Explizit vom Anwender deklarierte Schrittvariablen müssen lokal im AS-Editor deklariert werden. Sie dürfen sie nicht als VAR_INPUT, VAR_OUTPUT oder VAR_INOUT deklarieren, da TwinCAT 3.1 die Aufrufe nicht automatisch anpassen kann. Erklärung: Schritte verwenden zur Verwaltung der internen Zustände in TwinCAT 3.1 keine booleschen Variablen mehr, sondern auch Strukturen vom Typ SFCStepType. • Kennzeichner: Nicht mit einem Unterstrich beginnen dürfen folgende Kennzeichner: ◦ Namen von IEC-Aktionen im Baum ◦ Variablen, die in einer IEC-Assoziationsliste aufgerufen werden ◦ Namen von ausprogrammierten Transitionen Erklärung: In TwinCAT 3.1 erhalten die impliziten Variablen, welche TwinCAT 3.1 für Aktionen anlegt, einen Unterstrich als Präfix. Es würde ein ungültiger Kennzeichner mit doppeltem Unterstrich entstehen.
	CFC	• Große Bausteine: Das Layout großer Bausteine kann durch den Import an Qualität verlieren; Die Baustein-Boxen überlappen sich möglicherweise stark. • Makros: Makros können nicht importiert werden.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Vorhandenes Element hinzufügen (Projekt) [▶ 903]

4.6 SPS-Projekt konfigurieren

Sie können Ihr TwinCAT-SPS-Projekt in den folgenden Dialogen und Ansichten konfigurieren:

- TwinCAT-Optionen: Allgemeine projektunabhängige Einstellungen zum Verhalten von Editoren
- SPS-Projekteigenschaften: Projektbezogene Eigenschaften, Informationen und Einstellungen bezüglich Compiler, etc.
- SPS-Projekteinstellungen: Allgemeine Projekteinstellungen

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Eigenschaften (SPS-Projekt) [▶ 947]

- Dokumentation TC3 User Interface: Befehl Optionen [▶ 1010]
- Dokumentation TC3 User Interface: SPS-Projekteinstellungen [▶ 967]

4.7 Globale Datentypen verwenden

Wenn Datentypen nicht nur innerhalb eines SPS-Projekts, sondern auch SPS-Projektübergreifend verwendet werden sollen, können diese Datentypen zu globalen Datentypen konvertiert werden. Globale Datentypen werden im Typsystem des System Managers verwaltet und dem SPS-Projekt automatisch zur Verfügung gestellt.

Bis zum Build 4026 werden die globalen Datentypen mithilfe eines „super global“ Flags an das SPS-Projekt übergeben. Auf diese Weise sind sie nicht nur in dem SPS-Projekt selbst, sondern auch in den darin verwendeten Bibliotheken verfügbar.

Mit dem Build 4026 werden virtuelle Datentypbibliotheken automatisch erzeugt und im SPS-Projekt sowie den darin verwendeten Bibliotheken hinzugefügt. Auf oberster Ebene ist es die Tc3_GlobalTypes, die darunterliegend die eigentlichen Datentypbibliotheken mitbringt. Alle Datentypen mit Namespace werden in eine entsprechende Bibliothek mit diesem Namespace gruppiert, alle anderen werden unter der Bibliothek Tc3_GlobalTypes_Global hinzugefügt.

5 SPS-Projekt exportieren und transferieren

Für den Austausch der Daten von TwinCAT-Projekten mit anderen Programmen stehen Ihnen Export- und Importfunktionen zur Verfügung.

Ein Austausch von TwinCAT-Projekten zwischen TwinCAT-Entwicklungssystemen erfolgt durch eine Kopie der Projektdatei (*.project) oder des Projektarchivs (*.projectarchive).

5.1 SPS-Projekt exportieren und importieren

TwinCAT bietet Befehle zum Exportieren und Importieren von Objekten in eine oder aus einer Datei. Hier stehen mehrere Möglichkeiten zur Verfügung:

- Export in oder Import aus eine(r) ZIP-Datei (*.zip)
  Sie können dieses Format verwenden, um mehrere Dateien (auch nicht TwinCAT-Projektdateien) inklusive der Pfade zu exportieren und importieren.
- Export in ein Projektarchiv (*.zip)
  Sie können dieses Format verwenden, um ein ganzes Projekt (inklusive der verwendeten Bibliotheken) in ein ZIP-Archiv zu exportieren.
- Export in oder Import aus eine(r) XML-Datei im PLCopen-Format (*.xml)
  Sie können dieses Format verwenden, um Informationen mit anderen Programmen (zum Beispiel Programmeditoren oder Dokumentations-Tools) auszutauschen. PLCopenXML definiert eine Untermenge der in TwinCAT bekannten Elemente. Eine 100%-Kompatibilität ist somit nicht sichergestellt.

Der ZIP-Export bzw. ZIP-Import erfolgt analog zum hier beschriebenen PLCopenXml-Export bzw. PLCopen-Import.

Exportieren eines Projekts

✓ Ein Projekt ist in TwinCAT geöffnet.

1. Markieren Sie die SPS-Objekte oder das SPS-Projekt, welches Sie exportieren wollen.
2. Wählen Sie im Kontextmenü den Befehl **PLCopenXML exportieren…**
3. Wählen Sie im Dialog **PLCopenXML-Datei exportieren** den Speicherort bzw. den Dateinamen aus, an dem Sie die exportierte Datei speichern wollen.
4. Bestätigen Sie mit **Speichern**.

⇨ Die exportierte Datei steht Ihnen am gewählten Speicherort zur Verfügung.

Importieren eines Projekts

✓ Ein Projekt ist in TwinCAT geöffnet.

1. Markieren Sie das SPS-Projekt bzw. den Ordner im SPS-Projekt, an welchem die Datei importiert werden soll.
2. Wählen Sie im Kontextmenü den Befehl **PLCopenXML importieren…**
3. Wählen Sie im Dialog **PLCopenXML Datei importieren** die Datei aus, die Sie importieren wollen.
   ⇨ Ein Dialog öffnet sich und zeigt die Objekte in einer Baumstruktur an, die an dieser Stelle eingefügt werden können.
4. Wählen Sie die Objekte aus und klicken Sie auf **Öffnen**.

⇨ Die Objekte werden zum bestehenden Projektbaum hinzugefügt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Als ZIP exportieren
- Dokumentation TC3 User Interface: Befehl Aus ZIP importieren
- Dokumentation TC3 User Interface: Befehl PLCopenXML exportieren
- Dokumentation TC3 User Interface: Befehl PLCopenXML importieren

- Dokumentation TC3 User Interface: Dialog Optionen - PLCopenXML [▶ 1021]
- Dokumentation TC3 User Interface: Dialog Optionen - ZIP Export/Import [▶ 1040]

5.2 SPS-Projekt transferieren

Wenn Sie ein Projekt auf einen anderen Rechner übertragen und von dort mit derselben SPS verbinden wollen, ohne dass ein Online-Change oder Download erforderlich ist, beachten Sie folgende Punkte:

- Stellen Sie sicher, dass das Projekt nur feste Versionen von Bibliotheken (Ausnahme Schnittstellenbibliotheken), Visualisierungsprofil und Compiler verlangt.
- Stellen Sie sicher, dass das Boot-Projekt aktuell ist.

Erzeugen Sie ein Projektarchiv und entpacken Sie es auf dem anderen Rechner oder checken Sie das Projekt in Ihrem Quellcodeverwaltungssystem ein.

Übertragen eines Projekts auf ein anderes System

✓ Auf einem Rechner „PC1“ ist das Projekt geöffnet, das Sie auf einen anderen Rechner „PC2“ übertragen und von dort wieder mit derselben Steuerung verbinden wollen.

✓ Im Projekt sind nur Bibliotheken mit festen Versionen eingebunden (Ausnahme: reine Schnittstellenbibliotheken). Um dies zu prüfen, öffnen Sie den Bibliotheksverwalter. Wenn an einem Bibliothekseintrag ein „*“ anstelle einer festen Versionsangabe steht, ist die Bibliothek nicht mit einer festen Version eingebunden. (Siehe Abschnitt „Bibliotheken verwenden [▶ 278]“)

✓ Das geöffnete SPS-Projekt ist das gleiche wie das, welches gerade auf der SPS verwendet wird. Das heißt, das „Boot-Projekt“ ist identisch mit dem Projekt im Programmiersystem. Wenn im SPS-Projektbaum die Disketten-Symbole des SPS-Projekts und der SPS-Objekte rot sind, wurde das Projekt oder ein SPS-Objekt geändert, aber noch nicht gespeichert. In diesem Fall stimmen SPS-Projekt und Boot-Projekt möglicherweise nicht überein. Speichern Sie die Änderungen und erzeugen Sie ein neues Boot-Projekt. Um ein Boot-Projekt explizit zu erzeugen, wählen Sie im Kontextmenü des SPS-Knotens den Befehl **Activate Boot Project**. Loggen Sie sich mit dem Befehl **Einloggen** in die SPS ein und starten Sie mit dem Befehl **Start** die Ausführung. Das Projekt läuft nun auf der SPS, mit der Sie sich später von PC2 vom selben Projekt aus wieder verbinden möchten.

1. Wählen Sie im Kontextmenü des SPS-Projekts den Befehl **Sichern <TwinCAT-Projektname> als Archiv**…, um ein Projektarchiv zu erzeugen oder checken Sie das Projekt vollständig in Ihrem Quellcodeverwaltungssystem ein.
2. Übertragen Sie das Projektarchiv auf PC2 und extrahieren Sie es oder laden Sie auf PC2 die letzte Version aus Ihrem Quellcodeverwaltungssystem.
3. Öffnen Sie nun das TwinCAT-Projekt bzw. binden Sie die SPS in ein neues TwinCAT-3-Projekt ein.
4. Übersetzen Sie das Projekt.
5. Loggen Sie sich wieder in die SPS ein.

⇨ TwinCAT fordert keinen Online-Change oder Download. Das Projekt läuft.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Activate Boot Project
- Dokumentation TC3 User Interface: Befehl Einloggen [▶ 1001]
- Dokumentation TC3 User Interface: Befehl Sichern <TwinCAT-Projektname> als Archiv... [▶ 1113]
- Dokumentation TC3 User Interface: Befehl Start [▶ 1003]

6 SPS-Projekt lokalisieren

Sie können Ihr Projekt in unterschiedlichen Landessprachen darstellen, wenn Sie entsprechende Lokalisierungsdateien erstellen und einbinden. Die Lokalisierungsdateien entsprechen denen des GNU gettext-Systems. Die Lokalisierungsvorlagendateien sind also *pot-Dateien (Portable Object Template), aus denen nach der Übersetzung Lokalisierungsdateien im *.po-Dateien (Portable Object) erzeugt werden.

Das Projekt kann in verschiedenen Landessprachen dargestellt werden. Editieren ist jedoch nur in der Originalversion möglich.

Sie konfigurieren, welche Kategorien von Textinformationen im Projekt Sie lokalisieren möchten. Dann exportieren Sie diese Texte in eine Übersetzungsvorlage. Diese Vorlage ist eine Datei des Formats pot (beispielsweise „project_1.pot“). Mit einem entsprechenden externen Übersetzungstool oder auch händisch mithilfe eines einfachen Texteditors erzeugen Sie daraus Lokalisierungsdateien des Formats po (beispielsweise „de.po“, „en.po“, „es.po“). Die po-Dateien können Sie dann wieder in TwinCAT importieren und zur Lokalisierung verwenden. Die Befehle zum Handhaben der Projektlokalisierung finden Sie im Menü **PLC > Project Localization**.

Lokalisierungsvorlage erzeugen

✓ Ein Projekt ist geöffnet.

1. Wählen Sie im Menü **PLC > Project Localization** den Befehl **Lokalisierungsvorlage erzeugen**.
   ⇨ Der Dialog **Lokalisierungsvorlage erzeugen…** öffnet sich.
2. Aktivieren Sie die Kategorien von Textinformationen, die Sie in die Lokalisierungsvorlage aufnehmen möchten.
3. Auch „Positionsinformationen“ können in die Vorlage mit aufgenommen werden. Sie geben für jeden zu übersetzenden Text an, wo er im Projekt vorliegt. Wählen Sie hier aus, ob nur die erste gefundene Position des Texts, alle gefundenen Positionen oder gar keine in der Übersetzungsvorlage angezeigt werden sollen.
4. Klicken Sie auf die Schaltfläche **Erzeugen**.

⇨ Der Dialog zum Speichern einer Datei des Formats pot im Dateisystem öffnet sich. Speichern Sie die Lokalisierungsvorlage. Danach können Sie die Datei in einem entsprechenden Übersetzungstool bearbeiten und Lokalisierungsdateien *<Sprache>.po* in den gewünschten Sprachen erzeugen.

Format der Lokalisierungsvorlage (Datei *.pot))

In der ersten Zeile ist angegeben, welche Textkategorien beim Erzeugen der Vorlage für die Übersetzung ausgewählt wurden:

Beispiel:

#: Content:Comments|Identifiers|Names|Strings: Alle 4 Kategorien wurden ausgewählt.

Danach folgt für jeden zu übersetzenden Text ein Abschnitt in der Form wie im folgenden Beispiel zu sehen:

Beispiel:

```
#: D:\Projects\p1.project\Project_Settings:1
msgid "Project Settings"
msgstr ""
```

Zeile 1: Positionsinformation als Quellcode-Referenz: Wird nur angezeigt, wenn dies beim Erzeugen der Übersetzungsdatei so konfiguriert wurde.

Zeile 2: Unübersetzter Text als Eintrag msgid. Beispiel: `msgid "Project Settings"`.

Zeile 3: Platzhalter für die Übersetzung: `msgstr ""`. Zwischen den Hochkommas muss dann in der po-Datei die Übersetzung in der jeweiligen Sprache eingefügt werden.

Format der Lokalisierungsdatei (Datei *-<Sprache>.po)

Eine po-Datei können Sie mithilfe eines Übersetzungstools oder händisch mithilfe eines neutralen Texteditors auf Basis der pot-Datei erzeugen. Sie könnten also die pot-Datei in eine po-Datei umbenennen und entsprechend des po-Standardformats bearbeiten. Geben Sie unbedingt der Landessprache in Form der üblichen Kulturkürzel in den Metadaten der Datei an.

Beispiel: `"Language: de"` für Deutsch.

Die Übersetzungen der einzelnen Texte fügen Sie jeweils bei den msgstr ""-Einträgen zwischen den Hochkommas ein.

Beispiel:

```
"Language: de\n"
#: Content:Names
#: D:\projects\p1.project\Project_Settings:1
msgid "Project Settings"
msgstr "Projekteinstellungen"
```

Importieren der Lokalisierungsdateien (Lokalisieren des Projekts)

✓ Für Ihr Projekt wurden auf Basis der Übersetzungsvorlage *.pot Lokalisierungsdateien *<Sprache>.po* erzeugt. Das Projekt ist geöffnet.

1. Wählen Sie im Menü **PLC > Project Localization** den Befehl **Lokalisierungen verwalten…**
2. Klicken Sie auf die Schaltfläche **Hinzufügen**.
   ⇨ Der Dialog **Lokalisierungsdatei öffnen** zum Auswählen einer Datei des Formats po aus dem Dateisystem erscheint.
3. Wählen Sie eine der Lokalisierungsdateien, beispielsweise „<Projektname>-de.po“.
4. Der Dialog schließt sich und im Projekt erscheinen nun die betroffenen Texte in der entsprechenden Landessprache. Wenn Sie beispielsweise für den Bausteinnamen MAIN in der deutschen Lokalisierungsdatei die Übersetzung `msgstr "Hauptprogramm"` eingetragen haben, erscheint nun im SPS-Projektbaum der Objektname **Hauptprogramm**.
5. Importieren Sie auf gleiche Weise die Lokalisierungsdateien für die anderen Sprachen, in die übersetzt wurde.

Lokalisierung wechseln (Lokalisierungsdateien hinzufügen und entfernen)

✓ Alle gewünschten Landessprachen sind durch Importieren der entsprechenden po-Dateien im Projekt hinterlegt.

✓ Das Projekt ist geöffnet.

1. Wählen Sie im Menü **Projekt > Project Localization** den Befehl **Lokalisierung verwalten…**.
   ⇨ Der Dialog **Lokalisierung verwalten** öffnet sich. Unter **Dateien** erscheinen alle hinterlegten Lokalisierungsdateien *-<Sprache>.po*, sowie der Eintrag **<Originalversion>**.
2. Wählen Sie die gewünschte Sprache und klicken auf die Schaltfläche **Lokalisierung wechseln**.

⇨ Das Projekt erscheint in der gewählten Sprache. Wenn Sie **<Originalversion>** gewählt haben, erscheint das Projekt in der nicht-lokalisierten ursprünglichen Fassung und kann wieder editiert werden.

Optional eine Standardlokalisierung festlegen (Lokalisierung umschalten)

Selektieren Sie eine der verfügbaren Lokalisierungen und aktivieren Sie die Option **Standardlokalisierung**.

Mit dem Befehl **Lokalisierung umschalten** im Menü **PLC > Project Localization** wechseln Sie nun die Lokalisierung immer zwischen der Standardlokalisierung und der Originalversion. Der Befehl ist standardmäßig auch über die Schaltfläche  in der Werkzeugleiste verfügbar.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Lokalisierungsvorlage erzeugen [▶ 1008]
- Dokumentation TC3 User Interface: Befehl Lokalisierungen verwalten [▶ 1009]
- Dokumentation TC3 User Interface: Befehl Lokalisierung umschalten [▶ 1009]

7 SPS-Projekt programmieren

Um ein auf der Steuerung ein lauffähiges Anwendungsprogramm zu erstellen, füllen Sie POUs mit Deklarationen und Implementierungscode (Quellcode), stellen die Verknüpfung der E/As der Steuerung mit Programmvariablen her und konfigurieren eine Taskzuordnung. Nach Prüfung und Fehlerbeseitigung erstellt der Compiler dann den Programmcode, der auf die Steuerung geladen werden kann.

Die Programmierung der Projektbausteine (POUs) wird unterstützt durch die Programmiersprachen editoren und einigen weiteren Funktionalitäten wie beispielsweise Pragmas, Refactoring und Verwendung von fertigen Bausteinen aus TwinCAT-3-SPS-Bibliotheken.

Es gibt Mittel zur Syntaxprüfung und Codeanalyse, zum Herstellen von Datenpersistenz und zur Verschlüsselung des Programmcodes, der auf die Steuerung geladen wird.

Siehe auch:

- Ihr erstes TwinCAT-3-SPS-Projekt [▶ 25]

7.1 Bezeichner vergeben

Bezeichner sind die Namen von Variablen und Programmierobjekten, zum Beispiel Programme, Funktionsbausteine, Methode etc. und Namen anderer Objekte des SPS-Projekts. Es gibt Regeln, die Sie bei der Vergabe von Bezeichner einhalten müssen. Darüber hinaus gibt es Empfehlungen, die dazu dienen, die Bezeichner einheitlich und aussagekräftig zu gestalten.

Die Bezeichner von Variablen vergeben Sie bei der Variablendeklaration. Sie können diese Bezeichner im Deklarationsteil des Programmierobjekts ändern. Die Bezeichner für Programmierobjekte und andere Objekte vergeben Sie im Dialog beim Hinzufügen des jeweiligen Objekts. Sie können den Bezeichner eines bestehenden Objekts des SPS-Projekts im Eigenschaftenfenster des Objekts ändern. Die Bezeichner von Objekten, die nur einmal pro SPS-Projekt vorhanden sein können, können Sie nicht ändern, zum Beispiel die Bezeichner Bibliotheksverwalter (Refererences) und Rezepturverwalter (RecipeManager).

Siehe auch:

- Referenz Programmierung > Bezeichner [▶ 882]
- TwinCAT-3-Programmierkonventionen > Bezeichner/Namen

7.2 Variablen deklarieren

Variablendeklaration

Sie können Variablen an folgenden Stellen deklarieren:

- Deklarationsteil eines Programmierobjekts
- GVL-Editor

Der Dialog **Variable deklarieren unterstützt Sie bei der Variablendeklaration.**

Syntax

```
( <pragma> )*
<scope> ( <type qualifier> )?
      <identifier> (AT <address> )? : <data type> ( := <initial value> )? ;
END_VAR
```

<pragma> (optional)	Pragma (keinmal, einmal oder mehrmals) Durch das Hinzufügen eines Pragmas können Sie das Verhalten und die Eigenschaften einer oder mehrerer Variablen beeinflussen.	Siehe auch • Pragmas verwenden [▶ 143]
<scope>	Gültigkeitsbereich • VAR • VAR_CONFIG • VAR_EXTERNAL • VAR_GLOBAL • VAR_INPUT • VAR_INST • VAR_IN_OUT • VAR_OUTPUT • VAR_STAT • VAR_TEMP	Siehe auch • Variablen [▶ 718]
<type qualifier> (optional)	Typqualifizierer • CONSTANT • RETAIN • PERSISTENT	Siehe auch • Variablentypen - Attribut-Schlüsselwörter [▶ 731]
<identifier>	Bezeichner, Variablenname Die im Abschnitt „Bezeichner“ aufgeführten Regeln müssen Sie bei der Vergabe eines Bezeichners zwingend beachten. Zusätzlich finden Sie im Abschnitt „Bezeichner/Namen“ Konventionen, die der Vereinheitlichung bei der Namensvergabe dienen.	Siehe auch • Bezeichner [▶ 882] • Bezeichner/Namen
AT <address> (optional)	Zuweisung einer Adresse im Eingangs-, Ausgangs- oder Merkerspeicherbereich (I, Q oder M) Beispiel: AT %I*, AT %Q*	Siehe auch • AT-Deklaration [▶ 71] • Adressen [▶ 790]
<data type>	Datentyp • <elementary data type> • <user defined data type> • <function block >	Siehe auch • Datentypen [▶ 793]
<initial value> (optional)	Initialwert • <literal value> • <identifier> • <expression>	Siehe auch • Operanden [▶ 780] • ST-Ausdrücke [▶ 657]
(...)?	Optional	
(…)*	Optionale Wiederholung	

Variableninitialisierung

Der Standard-Initialisierungswert für alle Deklarationen ist 0. Im Deklarationsteil können Sie für jede Variable und jeden Datentyp auch benutzerdefinierte Initialisierungswerte angeben.

Die benutzerdefinierte Initialisierung beginnt mit dem Zuweisungsoperator := und besteht aus einem beliebigen, gültigen Ausdruck der Programmiersprache ST (strukturierter Text). Somit definieren Sie den Initialisierungswert mithilfe von Konstanten, anderen Variablen oder Funktionen. Wenn Sie eine Variable verwenden, müssen Sie diese ebenfalls initialisieren.

Beispiel 1:

```
VAR
    nVar1   : INT := 12;                    // initialization value 12
    nVar2   : INT := 13 + 8;                //
initialization value defined by an expression of constants
    nVar3   : INT := nVar2 + F_Fun(4);      //
initialization value defined by an expression that contains a function call; notice the order!
    pSample : POINTER TO INT := ADR(nVar1); //
not described in the standard IEC61131-3: initialization value defined by an adress function; Notice
: the pointer will not be initialized during an Online Change
END_VAR
```

Beispiel 2:

Im folgenden Beispiel werden eine Eingangsvariable und eine Eigenschaft von einem Funktionsbaustein initialisiert, welcher über eine FB_init-Methode [▶ 888] mit einem zusätzlichen Parameter verfügt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

Methode FB_Sample.FB_init:

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

Eigenschaft FB_Sample.nMyProperty und die zugehörige Set-Funktion:

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

Initialisierungsergebnis:

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8

- nLocalInitParam = 7
- nLocalProp = 9

Siehe auch:

- Referenz Programmierung > Datentypen [▶ 793]
- Referenz Programmierung > Variablentypen und spezielle Variablen [▶ 718]
- Referenz Programmierung > Operanden > Adressen [▶ 790]
- Referenz Programmierung > Bezeichner [▶ 882]
- TwinCAT-3-Programmierkonventionen > Bezeichner/Namen

7.2.1 AT-Deklaration

Um eine Projektvariable an eine flexible (*) oder direkte Adresse zu binden, können Sie die Adresse bei der Deklaration der Variablen angeben. Durch Verwendung einer entsprechenden Variablenbezeichnung können Sie der Adresse einen aussagekräftigen Namen geben.

Automatische Adressierung

Es wird empfohlen, keine direkte Adressierung für allokierte Variablen zu verwenden, sondern stattdessen den Platzhalter * zu nutzen.
Mit dem Platzhalter * (%I*, %Q* bzw. %M*) wird eine flexible und optimierte Adressierung von TwinCAT automatisch durchgeführt.

Syntax:

```
<identifier> AT <address> : <data type>;
```

Bei der Angabe einer Adresse werden die Position im Speicher und die Größe mittels spezieller Zeichenfolgen ausgedrückt. Eine Adresse ist gekennzeichnet mit dem Prozentzeichen %, dann folgt der Speicherbereichspräfix, der optionale Größenpräfix und die Speicherposition.

```
%<memory area prefix> ( <size prefix> )? <memory position>
```

```
<memory area prefix> : I | Q | M
<size prefix>        : X | B | W | D
<memory position>    : * | <number> ( .<number> )*
```

Speicherbereichspräfix

I	Eingangsspeicherbereich für Eingänge ("Inputs") Für physikalische Eingänge über Eingangstreiber ("Sensoren")
Q	Ausgangsspeicherbereich für Ausgänge ("Outputs") Physikalische Ausgänge über Ausgangstreiber ("Aktoren")
M	Merkerspeicherbereich

Größenpräfix

X	Single Bit
B	Byte (8 Bits)
W	Word (16 Bits)
D	Double word (32 Bits)

Beispiele:

```
IbSensor1 AT%I*    : BOOL;
IbSensor2 AT%IX7.5 : BOOL;
```

Wenn Sie nicht explizit eine Einzelbitadresse angeben, werden boolesche Variablen byteweise alloziert. Beispiel: Eine Wertänderung von bVar AT %QB0 betrifft den Bereich von QX0.0 bis QX0.7.

Wenn Sie eine Variable einer Adresse zuweisen, müssen Sie Folgendes beachten:

- Auf Variablen, die auf einen Eingang mit direkter Adressierung gelegt sind, können Sie im Implementierungseditor nicht schreibend zugreifen. Dies führt zu einem Compiler-Fehler.
- Wenn Sie AT-Deklarationen mit direkter Adressierung bei Struktur- oder Funktionsbaustein-Komponenten anwenden, verwenden alle Instanzen denselben Speicher. Dies entspricht der Verwendung von statischen Variablen [▶ 723], wie sie z. B. auch aus klassischen Programmiersprachen wie "C" bekannt sind.
- Das Speicher-Layout von Strukturen ist ebenfalls abhängig vom Zielsystem.

Gültige Adressen

Dem Schlüsselwort AT muss eine gültige Adresse folgen. Weitere Informationen hierzu finden Sie im Abschnitt Referenz Programmierung > Operanden > Adressen [▶ 790]. Beachten Sie mögliche Überlappungen im Fall vom Byte-Adressierungsmodus.

Beispiele

Variablendeklarationen:

IbSensor AT%I* : BOOL;	Bei der Adressangabe ist statt der Speicherposition der Platzhalter * angegeben. Dadurch wird eine flexible und optimierte Adressierung von TwinCAT automatisch durchgeführt.
InInput AT%IW0 : WORD;	Variablendeklaration mit Adressangabe eines Eingangsworts
ObActuator AT%QB0 : BOOL;	Boolesche Variablendeklaration Hinweis: Für boolesche Variable wird intern ein Byte alloziert, wenn keine Einzelbitadresse angegeben ist. Eine Wertänderung von ObActuator betrifft folglich den Bereich von QX0.0 bis QX0.7.
IbSensor AT%IX7.5 : BOOL;	Boolesche Variablendeklaration mit expliziter Angabe einer Einzelbitadresse. Beim Zugriff wird nur das Eingangsbit 7.5 gelesen.

Weitere Adressen:

%QX7.5 %Q7.5	Einzelbitadresse des Ausgangsbits 7.5
%IW215	Wortadresse des Eingangsworts 215
%QB7	Byteadresse des Ausgangsbytes 7
%MD48	Adresse eines Doppelworts an der Speicherstelle 48 im Merkerbereich
%IW2.5.7.1	Interpretation abhängig von der aktuellen Steuerungskonfiguration (siehe unten)

Siehe auch:

- Referenz Programmierung > Operanden > Adressen [▶ 790]

7.2.2 Deklarationseditor verwenden

Der Deklarationseditor dient der Deklaration von Variablen in Variablenlisten und POUs.

Wenn der Deklarationseditor in Verbindung mit einem Programmiersprachen-Editor verwendet wird, erscheint er als Deklarationsteil im oberen Teil des Fensters einer POU.

Der Deklarationseditor bietet zwei mögliche Ansichten: textuell ( ) und tabellarisch ( ). Im Dialog **Extras > Optionen > TwinCAT > SPS-Programmierumgebung > Deklarationseditor** definieren Sie, ob entweder nur die textuelle oder nur die tabellarische Ansicht verfügbar ist, oder ob Sie über die Schaltflächen am rechten Rand des Editorfensters zwischen den beiden Ansichten wechseln können.

Siehe auch:

- Referenz Programmierung: Deklarationseditor [▶ 654]

Deklarieren im textuellen Deklarationseditor

✓ Ein Programmierobjekt (POU oder GVL) eines Projekts ist geöffnet. Der Fokus ist im textuellen Deklarationseditor.

1. Geben Sie die Variablendeklarationen in korrekter Syntax ein. Über **[F2]** oder den Befehl **Eingabehilfe** im Kontextmenü erhalten Sie den Dialog **Eingabehilfe** zur Auswahl des Datentyps oder eines Schlüsselworts. Über den Befehl **Variable deklarieren** im Kontextmenü erhalten Sie den Dialog **Variable deklarieren**.

⇨ Bei der Deklaration der Variablen werden Schlüsselwörter automatisch korrigiert und farblich hervorgehoben.

Siehe auch:

- Dialog Variable deklarieren verwenden [▶ 73]
- Dokumentation TC3 User Interface: Befehl Eingabehilfe [▶ 913]
- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]

Deklarieren im tabellarischen Deklarationseditor

✓ Ein Programmierobjekt (POU oder GVL) eines Projekts ist geöffnet. Der Fokus liegt im tabellarischen Deklarationseditor.

1. Klicken Sie auf Schaltfläche  im Deklarationskopf oder wählen Sie im Kontextmenü den Befehl **Variable deklarieren**, um den Dialog **Variable deklarieren** zu öffnen.
   ⇨ TwinCAT fügt eine neue Zeile für eine Variablendeklaration ein und das Eingabefeld für den Variablennamen öffnet sich.
2. Geben Sie einen gültigen Variablenbezeichner ein.
3. Öffnen Sie nach Bedarf die anderen Felder der Deklarationszeile mit einem Doppelklick und wählen Sie die gewünschten Angaben aus den Auswahllisten oder mithilfe der erscheinenden Dialoge.

⇨ Bei der Deklaration der Variablen wird automatisch die korrekte Syntax hergestellt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]

7.2.3 Dialog Variable deklarieren verwenden

✓ Ein Programmierobjekt (POU oder GVL) eines Projekts ist geöffnet.

1. Wählen Sie im Menü **Bearbeiten** oder im Kontextmenü des Editors den Befehl **Variable deklarieren**.
   ⇨ Der Dialog **Variable deklarieren** öffnet sich.
2. Wählen Sie aus der Auswahlliste **Gültigkeitsbereich** den gewünschten Gültigkeitsbereich für die Variable aus.
3. Geben Sie einen Variablennamen in das Eingabefeld **Name** ein.
4. Wählen Sie aus der Auswahlliste **Datentyp** den gewünschten Datentyp aus.
5. Wenn der Initialisierungswert vom Standard-Initialisierungswert abweichen soll, geben Sie einen Initialisierungswert für die Variable ein.
6. Schließen Sie Ihre Eingaben mit einem Klick auf **OK** ab.
   ⇨ TwinCAT listet die neu deklarierte Variable im Deklarationsteil Ihres Programmierobjekts.

Mithilfe von Pragmas im Deklarationsteil können Sie die Verarbeitung der Deklaration durch den Compiler beeinflussen. (Siehe Abschnitt „Pragmas verwenden [▶ 143]“)

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]

7.2.4 Array deklarieren

✓ Ein Programmierobjekt (POU oder GVL) eines Projekts ist geöffnet.

1. Wählen Sie im Menü **Bearbeiten** oder im Kontextmenü des Editors den Befehl **Variable deklarieren**.
   ⇨ Der Dialog **Variable deklarieren** öffnet sich.
2. Wählen Sie aus der Auswahlliste **Gültigkeitsbereich** den gewünschten Gültigkeitsbereich für das Array aus.
3. Geben Sie einen Bezeichner für das Array in das Eingabefeld **Name** ein.
4. Klicken Sie auf die Schaltfläche neben dem Eingabefeld **Datentyp** und wählen Sie im Auswahlmenü den Eintrag **Array-Assistent** aus.
5. Geben Sie in die Eingabefelder **Dimension 1** die untere und die obere Indexgrenze der 1. Dimension des Arrays ein, zum Beispiel: 1 und 3.
   ⇨ Das Feld **Ergebnis** zeigt die 1. Dimension des Arrays an, zum Beispiel: ARRAY [1..3] OF ?.
6. Geben Sie im Eingabefeld **Basistyp** den Datentyp des Arrays direkt oder mithilfe der **Eingabehilfe** oder des **Array-Assistent** (Schaltfläche) ein, zum Beispiel: DINT.
   ⇨ Das Feld **Ergebnis** zeigt jetzt auch den Datentyp des Arrays an, zum Beispiel: ARRAY [1..3] OF DINT.
7. Definieren Sie entsprechend der Schritte 5 und 6 die Dimensionen 2 und 3 des Arrays, zum Beispiel: Dimension 2: 1 und 4, Dimension 3: 1 und 2.
   ⇨ Das Feld **Ergebnis** zeigt das Array mit den definierten Dimensionen an: Array [1..3, 1..4, 1..2] OF DINT. Das Array besteht aus 3 * 4 * 2 = 24 Elementen.

Bei einem Array variabler Länge deklarieren Sie die Dimensionsgrenzen mit dem Sternchen-Platzhalter *. Arrays variabler Länge sind nur in VAR_IN_OUT-Deklarationen von Funktionsbausteinen, Methoden oder Funktionen erlaubt.

Beispiel für ein zweidimensionales Array variabler Länge: aVariableLength : ARRAY [* , *] OF INT;

8. Klicken Sie auf **OK**.
   ⇨ Im Dialog **Variable deklarieren** zeigt das Feld **Datentyp** das Array an.
9. Wenn Sie die Initialisierungswerte des Arrays ändern wollen, klicken Sie auf die Schaltfläche neben dem Eingabefeld **Initialisierungswert**.
   ⇨ Der Dialog **Initialisierungswert** öffnet sich.
10. Selektieren Sie die Zeile des Array-Elements, dessen Initialisierungswert Sie ändern wollen. Beispiel: Arrayelement [1, 1, 1] auswählen.
11. Geben Sie im Eingabefeld unterhalb der Auflistung den gewünschten Initialisierungswert ein und klicken Sie auf die Schaltfläche **Wert auf ausgewählte Zeilen anwenden**, zum Beispiel: „Wert 4“.
    ⇨ TwinCAT zeigt den geänderten Initialisierungswert der selektierten Zeile an.
12. Klicken Sie auf **OK**.
    ⇨ Im Feld **Initialisierungswert** des Dialogs **Variable deklarieren** zeigt TwinCAT die Initialisierungswerte des Arrays an, zum Beispiel: [4, 23(0)].
13. Optional geben Sie einen Kommentar in das Eingabefeld ein.
14. Klicken Sie auf **OK**, um die Deklaration des Arrays abzuschließen.

⇨ TwinCAT fügt die Deklaration des Arrays im Deklarationsteil des Programmierobjekts hinzu.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]
- Referenz Programmierung > ARRAY [▶ 809]

7.2.5 Globale Variablen deklarieren

Globale Variablen deklarieren und bearbeiten Sie in globalen Variablenlisten. Globale Konstanten in Bibliotheken deklarieren Sie in Parameterlisten.

Siehe auch:

- Referenz Programmierung > Remanente Variablen - RETAIN, PERSISTENT [▶ 726]

7.2.5.1 Objekt Globale Variablenliste

Symbol: 

Eine Globale Variablenliste dient der Deklaration, der Bearbeitung und der Anzeige von globalen Variablen. Wenn Sie dem Projekt eine GVL hinzufügen, sind die Variablen projektweit gültig.

Objekt Globale Variablenliste anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum den Ordner **GVLs**.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Globale Variablenliste**.
3. Geben Sie einen Namen ein und klicken Sie auf **Öffnen**.
   ⇨ TwinCAT fügt die GVL dem SPS-Projektbaum hinzu und öffnet sie im Editor. Zwischen den Schlüsselwörtern VAR_GLOBAL und END_VAR können Sie die globalen Variablen definieren.

Globale Variablen definieren

✓ Eine GVL ist im Editor geöffnet.

1. Geben Sie die Variablendeklarationen in korrekter Syntax ein oder wählen Sie im Menü **Bearbeiten** oder im Kontextmenü des Editors den Befehl **Variable deklarieren**.
   ⇨ Der Dialog **Variable deklarieren** öffnet sich. In der Auswahlliste **Gültigkeitsbereich** ist der Eintrag VAR_GLOBAL ausgewählt.
2. Geben Sie im Feld **Name** einen Namen für die globale Variable ein.
3. Wählen Sie in der Auswahlliste **Datentyp** einen Datentyp aus.
4. Wenn Ihre Variable einen anderen Initialisierungswert als den Standard-Initialisierungswert haben soll, klicken Sie auf  neben dem Feld Initialisierungswert.
   ⇨ Der Dialog **Initialisierungswert** öffnet sich.
5. Doppelklicken Sie auf die Zelle I**nitialisierungswert** Ihrer Variable und geben Sie den gewünschten gültigen Wert ein.
6. Klicken Sie auf **OK**.
   ⇨ Der Initialisierungswert wird im Dialog **Variable deklarieren** angezeigt.
7. Aktivieren Sie bei Bedarf eines der Flags.
8. Bestätigen Sie Ihre Eingaben mit einem Klick auf die Schaltfläche **OK**.

⇨ TwinCAT fügt die deklarierte Variable in der GVL ein. Die globale Variable ist im gesamten SPS-Projekt verfügbar.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]

7.2.5.2 Objekt Parameterliste

Symbol: 

Wenn Sie globale Konstanten, die durch die Bibliothek bereitgestellt werden, später in einem SPS-Projekt konfigurieren wollen, definieren Sie diese in einer Parameterliste. Eine Parameterliste ist ein spezieller Typ einer globalen Variablenliste.

Objekt Parameterliste anlegen

✓ Ein Bibliotheksprojekt ist geöffnet.

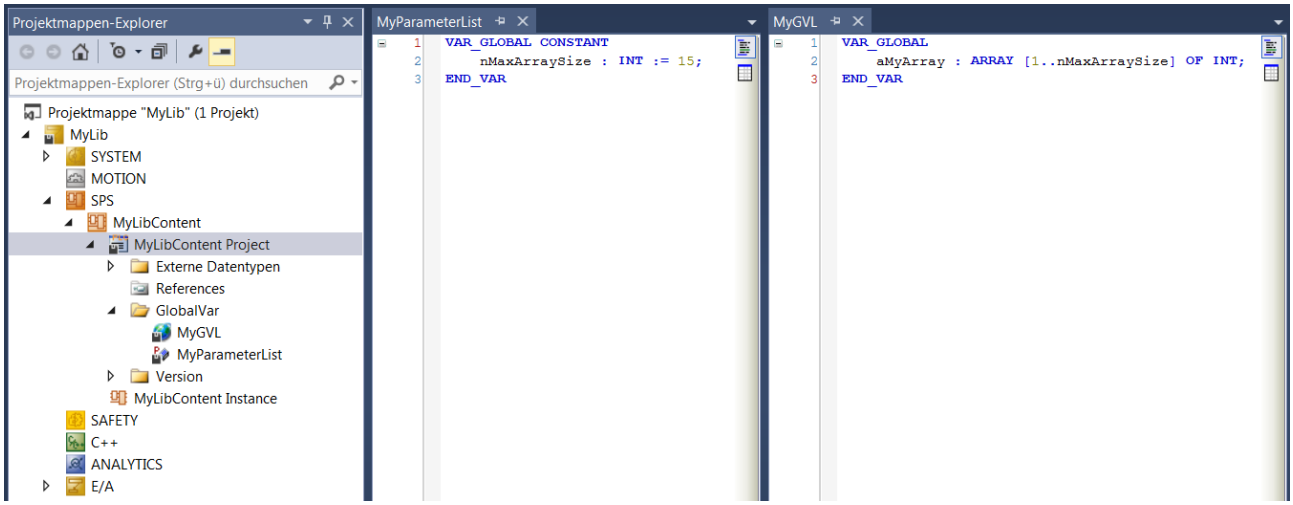
1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum das SPS-Projekt <Projektname.project>.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Parameterliste…**
3. Geben Sie einen Namen ein und klicken Sie auch **Öffnen**.

⇨ TwinCAT für die Parameterliste dem SPS-Projektbaum hinzu und öffnet sie im Editor. Zwischen den Schlüsselwörtern VAR_GLOBAL CONSTANT und END_VAR können Sie die globalen Konstanten definieren.

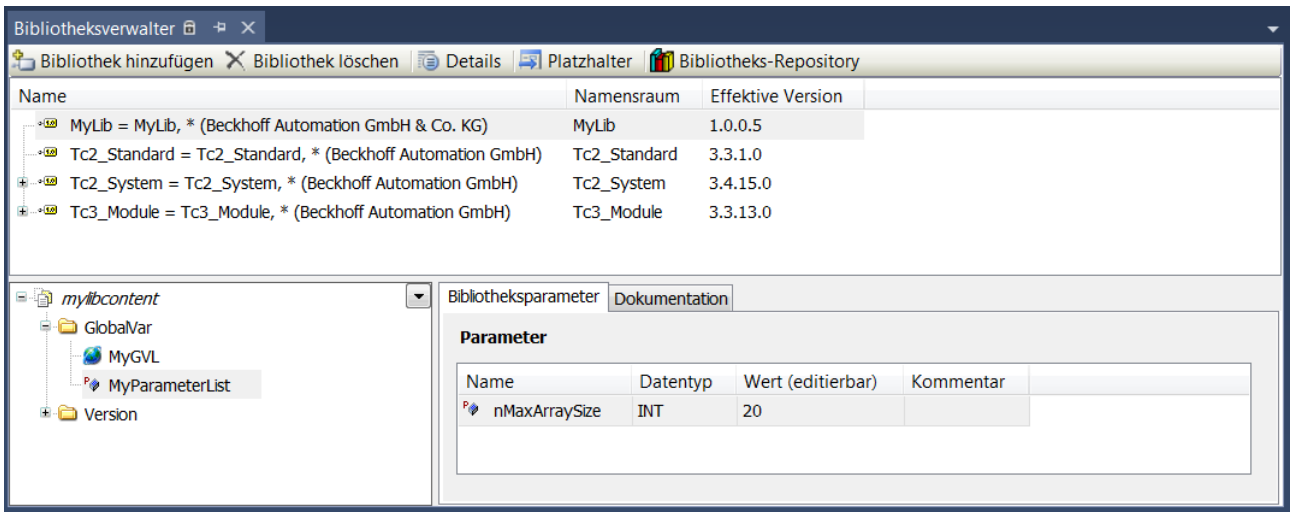
Beispiel:

Eine Bibliothek MyLib stellt eine ARRAY-Variable aMyArray bereit, deren Größe durch die globale Konstante nMaxArraySize definiert ist. Die Bibliothek wird in verschiedene SPS-Projekte eingebunden. Die SPS-Projekte verwenden unterschiedliche Array-Größen und die globale Konstante aus der Bibliothek soll jeweils mit einem projektspezifischen Wert überschrieben werden.

Die globale Konstante nMaxArraySize wird beim Erstellen der Bibliothek MyLib innerhalb einer Parameterliste definiert. Zunächst wird dem SPS-Projekt über den Befehl **Hinzufügen** im Kontextmenü ein Objekt **Parameterliste** mit dem Namen MyParameterList hinzugefügt. Im Editor des Objekts wird die Variable nMaxArraySize deklariert.



Die Bibliothek MyLib wird in ein SPS-Projekt eingebunden. Um den Wert der globalen Konstanten durch einen projektspezifischen Wert zu ersetzen, wird der Bibliotheksverwalter geöffnet. Die Bibliothek wird im oberen Bereich ausgewählt. Im unteren Bereich wird der Bausteinbaum mit der Parameterliste MyParameterList angezeigt. Die Parameterliste wird ausgewählt. Im rechten unteren Bereich öffnet sich die Registerkarte **Bibliotheksparameter** mit den in der Parameterliste enthaltenen Deklarationen. Der zu bearbeitende Wert der globalen Konstanten nMaxArraySize wird in der Spalte **Wert (editierbar)** selektiert. Durch Drücken der Leertaste wird ein Eingabefeld geöffnet, sodass der gewünschte neue Wert für nArraySize eingegeben werden kann. Sobald das Eingabefeld geschlossen wird, wird der Wert im lokalen Gültigkeitsbereich der Bibliothek angewendet.



Parameter exportieren und importieren

Verfügbar ab TwinCAT 3.1 Build 4026

Parameter exportieren:

✓ Eine Bibliothek mit Parameterliste ist im Bibliotheksverwalter ausgewählt.

1. Selektieren Sie im Bibliotheksverwalter die Parameterliste.
2. Wählen Sie im Kontextmenü den Befehl **Export Library Parameters…**
3. Wählen Sie einen Ordner aus, geben Sie einen Dateinamen ein und klicken Sie auf **Speichern**.

⇨ Die editierten Werte aus der Parameterliste werden in einer csv-Datei gespeichert.

Parameter importieren:

✓ Eine Bibliothek mit Parameterliste ist im Bibliotheksverwalter ausgewählt.

1. Selektieren Sie im Bibliotheksverwalter die Parameterliste.
2. Wählen Sie im Kontextmenü den Befehl **Import Library Parameters…**
3. Wählen Sie eine csv-Datei mit gespeicherten Parametern aus und klicken Sie auf **Öffnen**.

⇨ Die in der csv-Datei gespeicherten Werte werden in die Parameterliste eingefügt.

7.3 Anwenderspezifische Datentypen erzeugen

Zusätzlich zu den Standard-Datentypen können Sie eigene Datentypen wie Strukturen, Aufzählungen/ Enumeration, Referenzen und Unions definieren. Diese legen Sie als Datentyp-Objekte (DUT = Data Unit Types) an.

7.3.1 Objekt DUT

Symbol:

- für einen DUT ohne Textlistenunterstützung
- für einen DUT vom Typ Enumeration mit Textlistenunterstützung)

Ein DUT (Data Unit Typ) beschreibt einen anwenderspezifischen Datentyp.

Objekt DUT anlegen

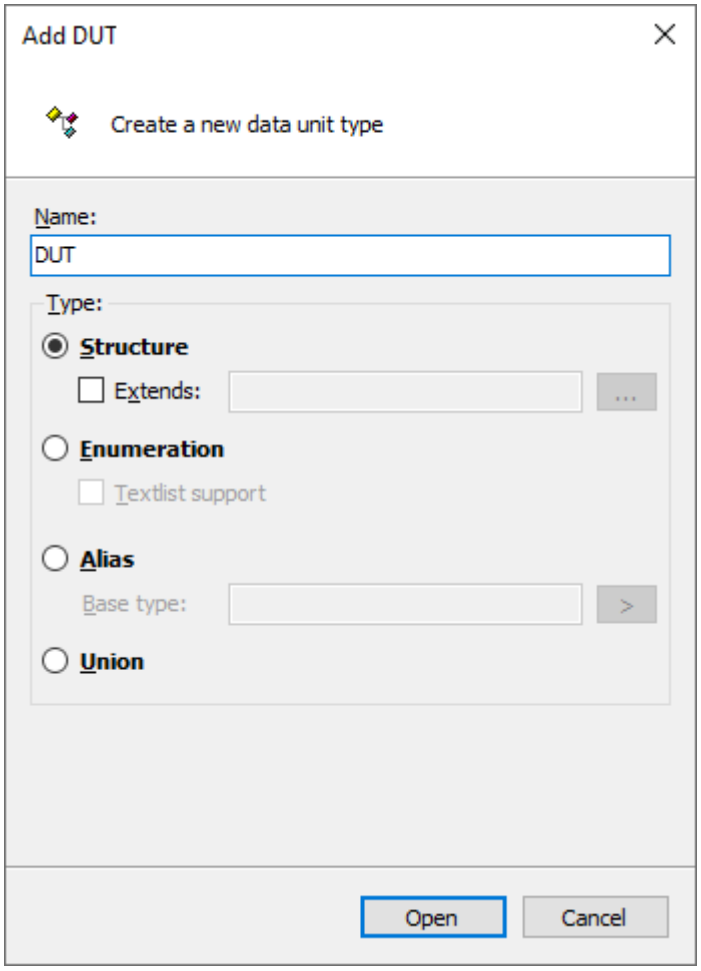
1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > DUT…**

⇨ Der Dialog **DUT hinzufügen** öffnet sich.

3. Geben Sie einen Namen ein und wählen Sie einen Datentyp.
4. Klicken Sie auf **Öffnen**.

⇨ Der DUT wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog DUT hinzufügen



Name	Name des neuen DUT

Datentyp

Struktur	Legt ein Objekt an, das eine Struktur deklariert, die mehrere Variablen mit unterschiedlichen Datentypen zu einer logischen Einheit zusammenfasst. Die innerhalb der Struktur deklarierten Variablen werden als Komponenten bezeichnet. ☑ **Erweitert**: Die Struktur erweitert eine bereits bestehende Struktur um weitere Komponenten. Geben Sie eine bestehende Struktur im Eingabefeld daneben an. Die Komponenten der bestehenden Struktur sind automatisch in der neuen verfügbar. (Siehe auch: Struktur [▶ 814])
Enumeration	Legt ein Objekt an, das eine Enumeration deklariert, die mehrere Integer-Konstanten zu einer logischen Einheit zusammenfasst Die innerhalb einer Enumeration deklarierten Konstanten werden auch als Enumerationswerte bezeichnet. ☐ **Textlistenunterstützung**: Eine Enumeration ohne Textlistenunterstützung wird angelegt. Das DUT-Objekt erscheint im SPS-Projektbaum im **Projektmappen-Explorer** mit dem folgenden Symbol:  ☑ **Textlistenunterstützung**: Die Textliste ermöglicht Ihnen, die Namen der Enumerationswerte zu lokalisieren. Das DUT-Objekt erscheint im SPS-Projektbaum im **Projektmappen-Explorer** mit dem folgenden Symbol:  Die lokalisierten Texte können Sie beispielsweise in einer Visualisierung ausgeben. Dann erscheinen in der Textausgabe eines Visualisierungselements statt der numerischen Enumerationswerte die symbolischen in der aktuellen Sprache. Wenn eine textlistenunterstützte Enumerationsvariable in der Eigenschaft **Textvariable** eines Visualisierungselements eingetragen wird, erhält sie den Zusatz <Enumerationsname>. Mithilfe der Schaltflächen am rechten Editorrand können Sie zwischen Textuelle Ansicht () und Lokalisierungsansicht (Textliste) () wechseln. Beispiel: Sie verwenden die Variable MAIN.eVar vom Typ E_myEnum. E_myEnum ist ein textlistenunterstützter DUT. Dann sieht der Eintrag im Eigenschafteneditor wie folgt aus: MAIN.eVar <E_myEnum>. Wenn der Enumerationstyp im SPS-Projekt geändert wird, erscheint eine Eingabeaufforderung mit der Frage, ob TwinCAT die betroffenen Visualisierungen entsprechend aktualisieren soll. Bei einem bestehenden Enumerationsobjekt kann die Textlistenunterstützung jederzeit nachträglich hinzugefügt oder wieder entfernt werden: Dazu dienen die Befehle **Textlistenunterstützung hinzufügen** oder **Textlistenunterstützung entfernen** im Kontextmenü des Objekts. (Siehe auch: Aufzählungen / Enumerationen [▶ 816])
Alias	Legt ein Objekt an, das ein Alias deklariert, mit dem ein alternativer Name für einen Basistyp, Datentyp oder einen Funktionsbaustein deklariert wird. Sie können den Basistyp direkt eingeben oder über die Eingabehilfe oder den Array-Assistenten auswählen. (Siehe auch: Alias [▶ 819])
Union	Legt ein Objekt an, das eine Union deklariert, die mehrere Komponenten mit meist unterschiedlichen Datentypen zu einer logischen Einheit zusammenfasst. Alle Komponenten haben den gleichen Offset, so dass sie am selben Speicherplatz liegen. Der Speicherplatzbedarf einer Union wird bestimmt durch den Speicherplatzbedarf seiner „größten“ Komponente. (Siehe auch: UNION [▶ 820])

DUT deklarieren

Syntax:

```
TYPE <identifier> : <data type declaration with optional initialization>
END_TYPE
```

Wie die Datentypdeklaration syntaktisch zu erfolgen hat, hängt im Detail vom gewählten Datentyp ab (z. B. Struktur oder Enumeration).

Beispiele:

Deklaration einer Struktur

Im Folgenden sehen Sie zwei DUTs, die die Strukturen ST_Struct1 und ST_Struct2 definieren. Die Struktur ST_Struct2 erweitert die Struktur ST_Struct1, was bedeutet, dass ST_Struct2.nVar1 verwendet werden kann, um auf die Variable nVar1 zuzugreifen.

```
TYPE ST_Struct1 :
STRUCT
    nVar1 : INT;
    bVar2 : BOOL;
END_STRUCT
END_TYPE
```

```
TYPE ST_Struct2 EXTENDS ST_Struct1 :
STRUCT
    nVar3 : DWORD;
    sVar4 : STRING;
END_STRUCT
END_TYPE
```

Deklaration einer Enumeration

```
TYPE E_TrafficSignal :
(
    eRed,
    eYellow,
    eGreen := 10
);
END_TYPE
```

Deklaration eines Alias

```
TYPE T_Message : STRING[50];
END_TYPE
```

Deklaration einer Union von Komponenten mit unterschiedlichen Datentypen

```
TYPE U_Name :
UNION
    fA : LREAL;
    nB : LINT;
    nC : WORD;
END_ UNION
END_TYPE
```

7.4 Programmierobjekte anlegen

Ihrem SPS-Projekt können Sie verschiedene Programmierobjekte hinzufügen, in denen Sie den Quellcode für Ihr Steuerungsprogramm schreiben und strukturieren.

7.4.1 Objekt POU

Symbol: 

Ein Objekt vom Typ **POU** ist eine Programm-Organisationseinheit (Programming Organization Unit) in einem TwinCAT-SPS-Projekt im Sinne der Norm IEC 61131-3.

Folgende Typen von POUs können Sie dem SPS-Projekt hinzufügen:

- Funktion [▶ 83]

- Funktionsbaustein [▶ 86]
- Programm [▶ 88]

In Erweiterung können Sie diesen Objekten wiederum folgende Programmierobjekte hinzufügen:

- Aktion [▶ 89]
- Transition [▶ 90]
- Methode [▶ 93]
- Eigenschaft [▶ 100]

Bestimmte POUs können andere POUs aufrufen. Rekursionen sind nicht erlaubt.

Beim Aufruf von POUs über den Namensraum durchsucht TwinCAT das Projekt nach der aufzurufenden POU gemäß folgender Reihenfolge:

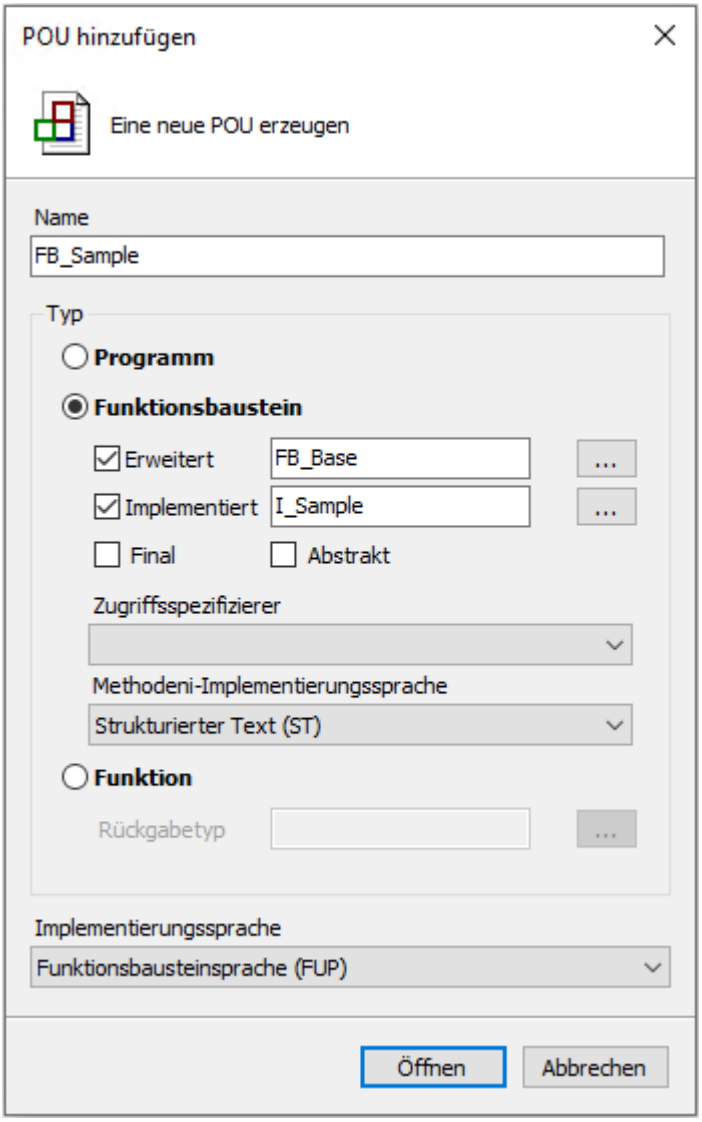
1. Aktuelle Anwendung
2. Bibliotheksverwalter der aktuellen Anwendung

Objekt POU anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU…**
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie einen Typ und die Implementierungssprache aus.
4. Klicken Sie auf **Öffnen**.

⇨ Die POU wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet. Der Editor besteht aus dem Deklarationseditor im oberen Teil und dem Implementierungsteil im unteren Teil. Abhängig von Implementierungssprache wird automatisch auch die Ansicht **Werkzeugkasten** aktiv, in der die passenden Elemente, Operatoren und Funktionsbausteine bereitstehen.

Dialog POU hinzufügen



Name	Name der POU
Implementierungssprache	Auswahlliste für die Implementierungssprache der POU

Typ

Programm	
Funktionsbaustein	• ☑ **Erweitert**: Angabe oder Auswahl eines Basis-Funktionsbausteins im Sinne der objektorientierten Programmierung. Wird mit Schlüsselwort EXTENDS in der Funktionsbausteindeklaration angegeben. • ☑ **Implementiert**: Angabe oder Auswahl einer Schnittstelle im Sinne der objektorientierten Programmierung. Wird mit Schlüsselwort IMPLEMENTS in der Funktionsbausteindeklaration angegeben. Bei der Anlage der POU werden alle Methoden angelegt, die über die Schnittstelle definiert sind. Siehe auch Automatisches Anlegen von Schnittstellenelementen in einem Funktionsbaustein [▶ 87] • ☑ **Final**: Abgeleiteter Zugriff ist nicht erlaubt. Das bedeutet, dass Sie den Funktionsbaustein nicht durch einen anderen Funktionsbaustein erweitern können. Dadurch können Sie kontrollieren, ob weitere Ableitungen erlaubt sein sollen oder nicht. Dies ermöglicht eine optimierte Codegenerierung. • ☑ **Abstrakt**: Kennzeichnet, dass der Funktionsbaustein eine fehlende oder unvollständige Implementierung hat und nicht instanziiert werden kann. Abstrakte Funktionsbausteine dienen ausschließlich als Basis-Funktionsbausteine und die Implementierung erfolgt typischerweise in einem abgeleiteten Funktionsbaustein. Wenn ein nicht-abstrakter Funktionsbaustein angelegt wird, der wiederum einen abstrakten Funktionsbaustein erweitert, werden sämtliche Methoden des abstrakten Basis-Funktionsbausteins als (nicht-abstrakte) Methoden dem neuen Funktionsbaustein hinzugefügt. Siehe auch ABSTRACT-Konzept [▶ 212] • **Zugriffsmodifizierer** ◦ PUBLIC: Entspricht der Angabe keines Zugriffsmodifizierers ◦ INTERNAL: Der Zugriff auf den Funktionsbaustein ist auf den Namensraum (die Bibliothek) beschränkt. • **Methodenimplementierungssprache**: Wenn Sie die Option **Implementiert** ausgewählt haben, können Sie hier eine Implementierungssprache für alle Methodenobjekte auswählen, die TwinCAT über die Implementierung der Schnittstelle erzeugt. Die Methoden-Implementierungssprache ist unabhängig von der Implementierungssprache des Funktionsbausteins.
Funktion	Nicht verfügbar, wenn in der Auswahlliste Implementierungssprache die Sprache Ablaufsprache (AS) ausgewählt ist. **Rückgabetyp**: Auswahlliste für den Datentyp des Rückgabewerts

7.4.1.1 Objekt Funktion

Eine Funktion ist eine POU, die bei der Ausführung genau ein Datenelement liefert und dessen Aufruf in textuellen Sprachen als Operator in Ausdrücken vorkommen kann. Das Datenelement kann auch ein Array oder eine Struktur sein.

Im SPS-Projektbaum haben Funktions-POUs das Suffix (FUN). Der Editor einer Funktion besteht aus dem Deklarationsteil und dem Implementierungsteil.

Alle Daten einer Funktion sind temporäre Daten und sind nur während der Ausführung einer Funktion gültig (Stack-Variablen). Das bedeutet, dass TwinCAT alle Variablen, die Sie in einer Funktion deklariert haben, bei jedem Aufruf der Funktion neu initialisiert.

Aufrufe einer Funktion mit denselben Eingabevariablen-Werten liefern immer denselben Ausgabewert. Deshalb dürfen Funktionen keine globalen Variablen und Adressen verwenden!

Die oberste Zeile des Deklarationsteils enthält folgende Deklaration:

```
FUNCTION <function> : <data type>
```

Darunter deklarieren Sie die Eingabe- und Funktionsvariablen.

Die Ausgabevariable einer Funktion ist der Funktionsname.

Wenn Sie eine lokale Variable in einer Funktion als RETAIN deklarieren, hat dies keinen Effekt. In diesem Fall gibt TwinCAT einen Compilerfehler aus.

In TwinCAT 3 können Sie explizite und implizite Parameterzuweisungen in Funktionsaufrufen nicht mischen. Das bedeutet, dass Sie entweder nur explizite oder nur implizite Parameterzuweisungen in Funktionsaufrufen verwenden. Die Reihenfolge der Parameterzuweisungen beim Funktionsaufruf ist beliebig.

Funktion aufrufen

In ST können Sie den Aufruf einer Funktion als Operand in Ausdrücken verwenden.

In AS können Sie einen Funktionsaufruf nur innerhalb von Schrittaktionen oder Transitionen verwenden.

Beispiele:

Funktion mit Deklarationsteil und einer Zeile Implementierungs-Code:

F_Sample

```
1  FUNCTION F_Sample : INT
2  VAR_INPUT
3      nVar1 : INT;
4      nVar2 : INT;
5      nVar3 : INT;
6  END_VAR
7  VAR
8  END_VAR
```

```
1  F_Sample := nVar1 + nVar2 * nVar3;
```

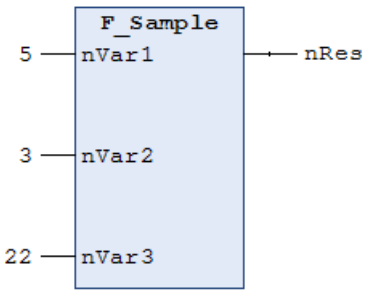
ST:

```
nRes := F_Sample(5,3,22)
```

AWL:

```
1   LD          5
    F_Sample    3
                22
    ST          nRes
```

FUP:



Funktionen mit zusätzlichen Ausgängen

Nach der Norm IEC 61131-3 können Funktionen zusätzliche Ausgänge haben. Die zusätzlichen Ausgänge deklarieren Sie in der Funktion zwischen den Schlüsselwörtern VAR_OUTPUT und END_VAR. Die Funktion rufen Sie gemäß folgender Syntax auf:

<function> (<function output variable1> => <output variable 1>, <function output variable n> => <output variable n>)

Beispiel:

Die Funktion F_Fun ist mit zwei Eingabevariablen nIn1 und nIn2 definiert. Der Ausgabevariablen der Funktion F_Fun wird auf die lokal deklarierten Ausgangsvariablen nLoc1 und nLoc2 geschrieben.

```
F_Fun(nIn1 := 1, nIn2 := 2, nOut1 => nLoc1, nOut2 => nLoc2);
```

Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf

Um direkt bei einem Methoden-, Funktions- oder Eigenschaftenaufruf auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode/Funktion/Eigenschaft zurückgeliefert wird, zugreifen zu können, kann folgende Umsetzung verwendet werden. Ein strukturierter Datentyp ist beispielsweise eine Struktur oder ein Funktionsbaustein.

1. Der Rückgabetyp der Methode/Funktion/Eigenschaft wird als „REFERENCE TO <structured type>“ definiert (anstelle von lediglich „<structured type>“).
2. Bei einem solchen Rückgabetyp ist zu beachten, dass – falls beispielsweise eine FB-lokale Instanz des strukturierten Datentyps zurückgeliefert werden soll – der Referenzoperator REF= anstatt des „normalen“ Zuweisungsoperators := verwendet werden muss.

Die Erklärungen und das Beispiel dieses Abschnitts beziehen sich auf den Aufruf einer Eigenschaft. Sie sind aber genauso auf andere Aufrufe übertragbar, die Rückgabewerte liefern (z. B. Methoden oder Funktionen).

Beispiel

Deklaration der Struktur ST_Sample (strukturierter Datentyp):

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

Deklaration des Funktionsbausteins FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

Deklaration der Eigenschaft FB_Sample.MyProp mit dem Rückgabetyp „REFERENCE TO ST_Sample“:

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

Implementierung der Get-Methode von der Eigenschaft FB_Sample.MyProp:

```
MyProp REF= stLocal;
```

Implementierung der Set-Methode von der Eigenschaft FB_Sample.MyProp:

```
stLocal := MyProp;
```

Aufruf der Get- und Set-Methoden im Hauptprogramm MAIN:

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

Durch die Deklaration des Rückgabetyps der Eigenschaft MyProp als „REFERENCE TO ST_Sample“ und durch die Verwendung des Referenzoperators REF= in der Get-Methode dieser Eigenschaft, kann direkt beim Aufruf der Eigenschaft auf ein einzelnes Element des zurückgelieferten strukturierten Datentyps zugegriffen werden.

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

Wenn der Rückgabetyp nur als „ST_Sample“ deklariert wäre, müsste die von der Eigenschaft zurückgelieferte Struktur zunächst einer lokalen Strukturinstanz zugewiesen werden. Die einzelnen Strukturelemente könnten dann anhand der lokalen Strukturinstanz abgefragt werden.

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

7.4.1.2 Objekt Funktionsbaustein

Ein Funktionsbaustein ist eine POU [▶ 80], die bei der Ausführung einen oder mehrere Werte liefert. Die Werte der Ausgabevariablen und der internen Variablen bleiben nach einer Ausführung bis zur nächsten erhalten. Dies bedeutet, dass der Funktionsbaustein bei mehrmaligem Aufruf mit denselben Eingabevariablen nicht unbedingt dieselben Ausgabewerte liefert.

Im SPS-Projektbaum haben Funktionsbaustein-POUs das Suffix (FB). Der Editor eines Funktionsbausteins besteht aus dem Deklarationsteil und dem Implementierungsteil.

Sie rufen einen Funktionsbaustein immer über eine Instanz auf, die eine Kopie des Funktionsbausteins ist.

Zusätzlich zu der in der IEC 61131-3 beschriebenen Funktionalität können Sie Funktionsbausteine in TwinCAT auch für folgende Funktionalitäten der objektorientierten Programmierung verwenden:

- Erweitern eines Funktionsbausteins (Erweitern eines Funktionsbausteins [▶ 201])
- Implementieren von Schnittstellen (Implementieren einer Schnittstelle [▶ 208])
- Methoden (Objekt Methode [▶ 93])
- Eigenschaften (Objekt Eigenschaft [▶ 100])

Die oberste Zeile des Deklarationsteils enthält folgende Deklaration:

FUNCTION_BLOCK <access specifier> <function block> | EXTENDS <function block> |
IMPLEMENTS <comma-separated list of interfaces>

8-Byte-Alignment

Mit TwinCAT 3 wurde ein 8-Byte-Alignment eingeführt. Achten Sie auf ein passendes Alignment, wenn Daten als gesamter Speicherblock mit anderen Steuerungen oder Softwarekomponenten ausgetauscht werden (siehe Alignment [▶ 826]).

Automatisches Anlegen von Schnittstellenelementen in einem Funktionsbaustein

Es gibt zwei Möglichkeiten, wie Sie die Elemente einer Schnittstelle, die ein Funktionsbaustein implementiert, automatisch in diesem Funktionsbaustein erzeugen lassen können.

1. Wenn Sie beim Anlegen eines neuen Funktionsbausteins in dem Feld **Implementiert** des Dialogs **Hinzufügen** eine Schnittstelle angeben, fügt TwinCAT diesem Funktionsbaustein auch automatisch die Methoden und Eigenschaften der Schnittstelle hinzu.
2. Wenn ein bestehender Funktionsbaustein eine Schnittstelle implementiert, können Sie den Befehl **Schnittstellen implementieren** verwenden, um die Schnittstellenelemente in dem Funktionsbaustein erzeugen zu lassen. Den Befehl **Schnittstellen implementieren** finden Sie im Kontextmenü eines Funktionsbausteins im Projektbaum.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Schnittstellen implementieren [▶ 1112]
- Implementieren einer Schnittstelle: Implementieren einer Schnittstelle [▶ 208]

Funktionsbaustein aufrufen

Der Aufruf erfolgt immer über eine Instanz des Funktionsbausteins. Beim Aufruf eines Funktionsbausteins ändern sich nur die Werte der jeweiligen Instanz.

Deklaration der Instanz:

```
<instance> : <function block>;
```

Auf eine Variable des Funktionsbausteins greifen Sie im Implementierungsteil wie folgt zu:

```
<instance>.<variable>
```

- Sie können von außerhalb der Funktionsbaustein-Instanz nur auf Eingabe- und Ausgabevariablen eines Funktionsbausteins zugreifen, nicht auf die internen Variablen.
- Der Zugriff auf eine Funktionsbaustein-Instanz ist auf die POU begrenzt, in der die Instanz deklariert ist, außer Sie haben die Instanz global deklariert.
- Sie können beim Aufruf der Instanz den Funktionsbausteinvariablen die gewünschten Werte zuweisen.

Beispiel:

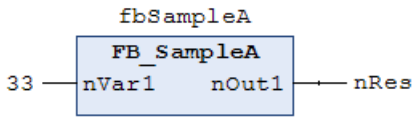
Der Funktionsbaustein FB_SampleA hat die Eingabevariable nVar1 vom Typ INT und die Ausgabevariable nOut1. Im Folgenden wird die Variable nVar1 aus dem Programm MAIN aufgerufen.

ST:

```
PROGRAM MAIN
VAR
    fbSampleA : FB_SampleA;
END_VAR
```

```
fbSampleA.nVar1 := 33; (* FB_SampleA is called and the value 33 is assigned to the variable nVar1 *)
fbSampleA(); (* FB_SampleA is called, that's necessary for the following access to the output variab
le *)
nRes := fbSampleA.nOut1 (* the output variable nOut1 of the FB1 is read *)
```

FUP:



Variablenwerte beim Aufruf zuweisen:

In den textuellen Sprachen AWL und ST können Sie Werte beim Aufruf des Funktionsbausteins direkt an Eingabe- und/oder Ausgabevariablen zuweisen.

Die Zuweisung eines Werts an einen Eingabevariable erfolgt mit :=

Die Zuweisung eines Wert an eine Ausgabevariable erfolgt mit =>

Beispiel:

Die Instanz fbTimer des Timer-Funktionsbausteins wird mit Zuweisungen für die Eingabevariable IN und PT aufgerufen. Anschließend wird die Ausgabevariable Q des Timers der Variablen bVarA zugewiesen

```
PROGRAM MAIN
VAR
    fbTimer : TOF;
    bIn     : BOOL;
    bVarA   : BOOL;
END_VAR
```

```
fbTimer(IN := bIn, PT := t#300ms);
bVarA := fbTimer.Q;
```

Wenn Sie eine Funktionsbaustein-Instanz über die Eingabehilfe einfügen und im Dialog **Eingabehilfe** die Option **Mit Argumenten einfügen** aktiviert ist, fügt TwinCAT den Aufruf mit allen Eingabe- und Ausgabevariablen ein. Sie müssen dann nur die gewünschten Wertzuweisungen einfügen. Im obigen Beispiel fügt TwinCAT den Aufruf wie folgt ein: CMD_TMR (IN:= , PT:= , Q=>).

Mit Hilfe des Attributs **'is_connected'** auf einer lokalen Variablen können Sie zur Zeit des Aufrufs in der Funktionsbaustein-Instanz feststellen, ob ein bestimmter Eingang eine Zuweisung von außen erhält.

7.4.1.3 Objekt Programm

Ein Programm ist eine POU, die bei der Ausführung einen oder mehrere Werte liefert. Alle Werte bleiben nach einer Ausführung des Programms bis zur nächsten Ausführung erhalten. Die Aufrufreihenfolge der Programme innerhalb eines SPS-Projekts definieren Sie in Taskobjekten.

Im SPS-Projektbaum haben die Programm-POUs das Suffix (PRG). Der Editor eines Programms besteht aus dem Deklarationsteil und dem Implementierungsteil.

Die oberste Zeile des Deklarationsteils enthält folgende Deklaration:

```
PROGRAM <program>
```

Programm aufrufen

Programme und Funktionsbausteine können ein Programm aufrufen. In einer Funktion ist ein Programmaufruf nicht erlaubt. Von Programmen gibt es keine Instanzen.

Wenn eine POU ein Programm aufruft und sich dadurch Werte des Programms verändern, bleiben diese Änderungen bis zum nächsten Programmaufruf erhalten. Die Werte des Programms bleiben auch dann erhalten, wenn der erneute Aufruf durch eine andere POU erfolgt. Dies unterscheidet sich vom Aufruf eines Funktionsbausteins. Beim Funktionsbaustein-Aufruf ändern sich nur die Werte der jeweiligen Instanz des Funktionsbausteins. Die Änderungen sind nur zu beachten, wenn eine POU dieselbe Instanz erneut aufruft.

Sie können für ein Programm die Eingabe- und/oder Ausgabeparameter auch direkt beim Aufruf setzen.

Syntax:

<program>(<input variable> := <value>, <output value> => <value>):

Wenn Sie einen Programmaufruf über die Eingabehilfe einfügen und dabei in der **Eingabehilfe** die Option **Mit Argumenten einfügen** aktiviert ist, fügt TwinCAT dem Programmaufruf Eingabe- und/oder Ausgabeparameter gemäß der Syntax zu.

Beispiele:

AWL:

```
1   CAL          SampleProg(
          nIn1:= 2)
    LD           SampleProg.nOut2
    ST           nRes
```

Mit Zuweisung der Parameter:

```
1   CAL          SampleProg(
          nIn1:= 2
         nOut2=> nRes)
```

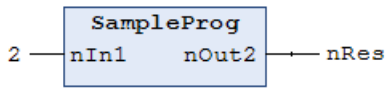
ST:

```
SampleProg();
nRes := SampleProg.nOut2;
```

Mit Zuweisung der Parameter:

```
SampleProg(nIn1 := 2, nOut2 => nRes);
```

FUP:



7.4.2 Objekt Aktion

Symbol: 

In einer Aktion implementieren Sie weiteren Programmcode. Diesen Programmcode können Sie in einer anderen Sprache implementieren als die Basisimplementierung. Die Basisimplementierung ist der Funktionsbaustein oder das Programm, unter der Sie die Aktion eingefügt haben.

Eine Aktion hat keine eigenen Deklarationen und arbeitet mit den Daten der Basisimplementierung. Das bedeutet, dass die Aktion die Eingabe/Ausgabe- und lokalen Variablen ihrer Basisimplementierung verwendet.

Objekt Aktion anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein oder ein Programm.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Aktion…**
   ⇨ Der Dialog **Aktion hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und eine Implementierungssprache aus.
4. Klicken Sie auf **Öffnen**.

⇨ Das Objekt wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog Aktion hinzufügen

Name	Name der Aktion
Implementierungssprache	Auswahlkästchen für die Implementierungssprache

Aktion aufrufen

Syntax:

<program>.<action> oder <FB-instance>.<action>

Wenn Sie eine Aktion nur innerhalb der Basisimplementierung aufrufen wollen, genügt es, nur den Aktionsnamen anzugeben.

Beispiele:

Aufrufe einer Aktion Reset von einer anderen POU aus. Der Aufruf erfolgt also nicht in der Basisimplementierung.

Deklaration:

```
PROGRAM MAIN
VAR
    fbCounterA : FB_Counter;
END_VAR
```

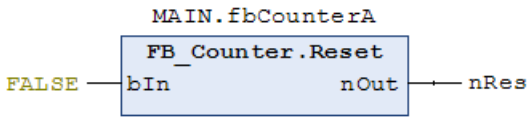
Aufruf der Aktion Reset in einer AWL-POU:

```
1   CAL     fbCounterA.Reset(
            bIn:= FALSE)
    LD      fbCounterA.nOut
    ST      nRes
```

Aufruf der Aktion Reset in einer ST-POU:

```
fbCounterA.Reset(In := FALSE);
nRes := fbCounterA.nOut;
```

Aufruf der Aktion Reset in einer FUP-POU:



Häufige Verwendung finden Aktionen in der Implementierungssprache AS. (AS-Element Aktion [▶ 678])

Siehe auch:

- Objektorientiert programmieren [▶ 178]
- Erweitern eines Funktionsbausteins [▶ 201]

7.4.3 Objekt Transition

Symbol:

In einer Transition legen Sie eine Bedingung fest, unter der ein nachfolgender Schritt aktiv werden soll. Eine Transitionsbedingung kann den Wert TRUE oder FALSE haben. Wenn sie TRUE ist, wird der nachfolgende Schritt ausgeführt. Eine Transitionsbedingung kann auf folgende zwei Arten definiert werden:

- (1) Direkt („Inline-Bedingung“): Sie ersetzen den Standard-Transitionsnamen durch den Namen einer booleschen Variablen, einer booleschen Adresse, einer booleschen Konstante oder einer Anweisung mit booleschem Ergebnis (z. B. (i<100) AND b). Sie dürfen hier keine Programme, Funktionsbausteine oder Zuweisungen angeben.
- (2) Verwendung eines separaten Transitions- oder Eigenschaftenobjekts („multi-use condition“): Sie ersetzen den Standard-Transitionsnamen durch den Namen eines Transitions- oder Eigenschaften-Objekts. Sie legen diese Objekte über den Befehl **Hinzufügen > Transition..** im Kontextmenü an (siehe Abschnitt „Objekt Transition anlegen [▶ 91]“). Dies erlaubt eine Mehrfachverwendung

(„multiple-use“) von Transitionen. Das Objekt kann wie eine „Inline-Bedingung“ eine boolesche Variable, Adresse, Konstante oder Anweisung enthalten, aber auch Mehrfachanweisungen mit beliebigem Code.

Beachten Sie die Hinweise im Abschnitt „Zugriff auf VAR_IN_OUT_Variablen des Funktionsbausteins [▶ 92]“.

Objekt Transition anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein oder ein Programm.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Transition**…
   ⇨ Der Dialog **Transition hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie eine Implementierungssprache aus.
4. Klicken Sie auf **Öffnen**.

⇨ Das Objekt wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog Transition hinzufügen

Name	Name der Transition
Implementierungssprache	Auswahlkästchen für die Implementierungssprache

Transition aufrufen

Syntax:

Im Gegensatz zu TwinCAT 2 PLC Control wird eine Transitionsbedingung wie ein Methodenaufruf behandelt. Die Eingabe erfolgt nach folgender Syntax:

<Transitionsname> := <Transitionsbedingung>

oder

<Transitionsbedingung>

Enthält eine Transition mehrfache Anweisungen, müssen Sie den gewünschten Ausdruck der Transitionsvariable zuweisen (erste Variante der Syntax).

Beispiele:

Aufruf der Transition Trans1 in einer ST-POU:

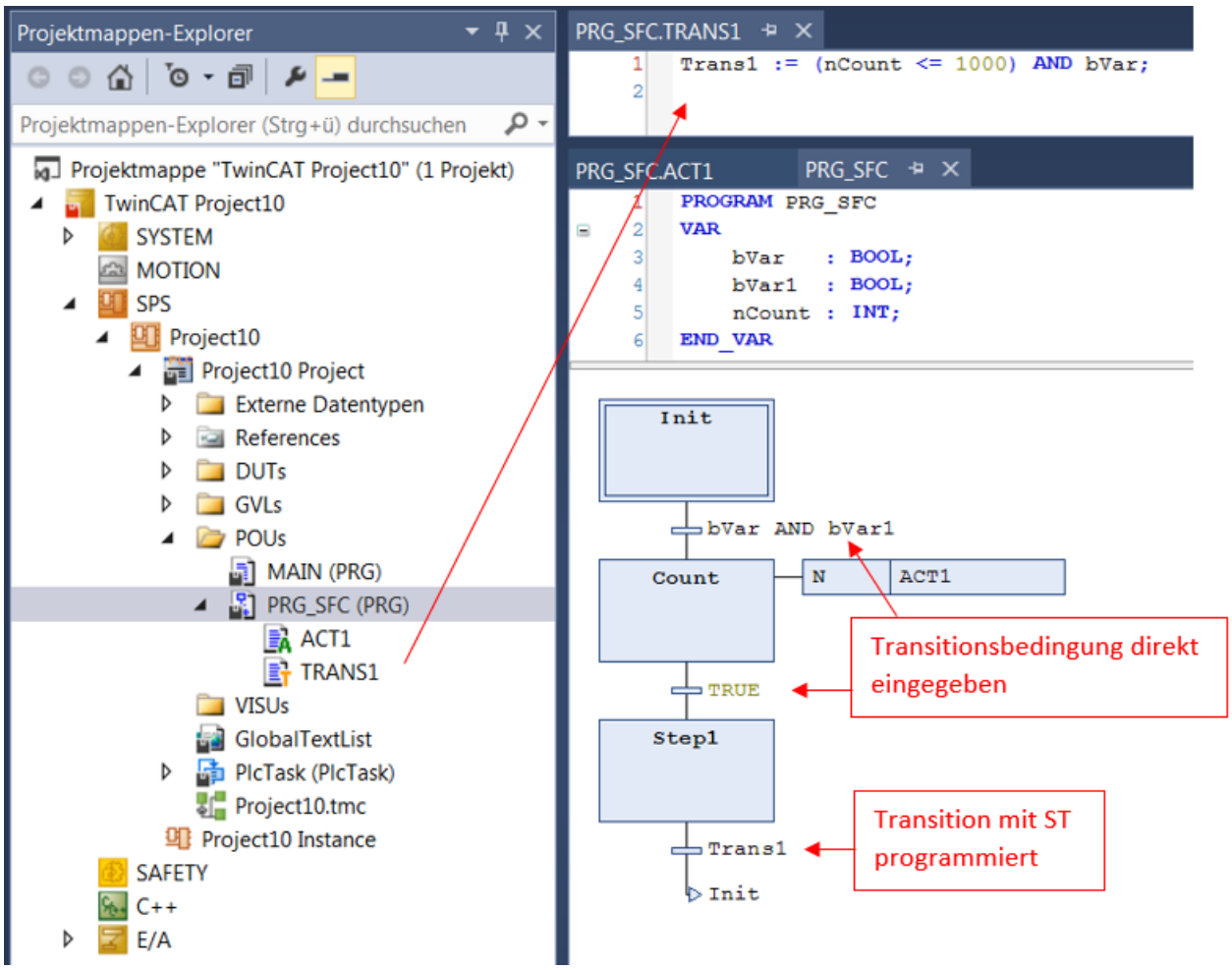
PRG_SFC.Trans1:

```
Trans1 := (nCount<=100);
```

PRG_SFC:

```
nCount := nCount+1;
IF Trans1 = TRUE THEN // IF nCount<=100 THEN …
    nVar:=1;
    ELSE
        nVar:=2;
END_IF
```

Aufruf der Transition Trans1 in einer SFC-POU:



Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition/Eigenschaft

Auf die VAR_IN_OUT-Variablen eines Funktionsbausteins kann in einer Methode, einer Transition oder einer Eigenschaft des Funktionsbausteins prinzipiell zugegriffen werden. Bei einem solchen Zugriff ist folgendes zu beachten:

- Wird der Rumpf oder eine Aktion des Funktionsbausteins von außerhalb des FBs aufgerufen, stellt der Compiler sicher, dass bei diesem Aufruf die VAR_IN_OUT-Variablen des Funktionsbausteins zugewiesen werden.
- Bei dem Aufruf einer Methode, Transition oder Eigenschaft des Funktionsbausteins ist dies nicht der Fall, da die VAR_IN_OUT-Variablen des FBs nicht innerhalb eines Methoden-, Transitions- oder Eigenschaftenaufrufs zugewiesen werden können. Es könnte daher gegebenenfalls ein Zugriff auf die VAR_IN_OUT-Variablen stattfinden, indem die Methode/Transition/Eigenschaft aufgerufen wird, bevor die VAR_IN_OUT-Variablen einer gültigen Referenz zugewiesen wurden. Da dies einem ungültigen Zugriff während der Laufzeit entsprechen würde, ist es potentiell riskant, auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft zuzugreifen.

Aus diesem Grund wird bei einem Zugriff auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft folgende Warnung mit der ID C0371 ausgegeben:

„Warning: Access to VAR_IN_OUT <Var> declared in <POU> from external context <Method/Transition/Property>"

Eine adäquate Reaktion auf diese Warnung ist beispielsweise die Überprüfung der VAR_IN_OUT-Variablen innerhalb der Methode/Transition/Eigenschaft, bevor auf sie zugegriffen wird. Diese Überprüfung ist mithilfe des Operators __ISVALIDREF möglich, mit dem geprüft werden kann, ob eine Referenz auf einen gültigen Wert verweist. Ist diese Prüfung vorhanden, kann zum einen davon ausgegangen werden, dass sich der Anwender des Risikos bewusst ist, das potentiell vorhanden ist, wenn auf die VAR_IN_OUT-Variablen des FBs in einer Methode/Transition/Eigenschaft zugegriffen wird. Zum anderen liegt durch die Überprüfung der Referenz ein angemessener Umgang mit diesem Risiko vor. Somit kann die Warnung für diesen überprüften Bereich mittels Attribut 'warning disable' unterdrückt werden.

Die dazugehörige Beispielimplementierung einer Methode ist im Folgenden dargestellt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

Methode FB_Sample.MyMethod:

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

Siehe auch:

- Objektorientiert programmieren [▶ 178]
- Erweitern eines Funktionsbausteins [▶ 201]
- Referenz Programmierung: AS-Elemente Schritt und Transition [▶ 677]

7.4.4 Objekt Methode

Symbol: 

Das Objekt dient der objektorientierten Programmierung.

Eine Methode enthält eine Abfolge von Anweisungen, ist jedoch im Gegensatz zu einer Funktion keine unabhängige POU, sondern muss einem Funktionsbaustein oder einem Programm zugeordnet sein.

Zur Organisation von Methoden können Sie Schnittstellen verwenden.

Hinweise zu Methoden

- Alle Daten einer Methode sind temporäre Daten und sind nur während der Ausführung einer Methode gültig (Stack-Variablen). Das bedeutet, dass TwinCAT alle Variablen und Funktionsbausteine, die Sie in einer Methode deklariert haben, bei jedem Aufruf der Methode neu initialisiert.
- Methoden können ebenso wie Funktionen einen Rückgabewert zurückliefern.
- Gemäß der Norm IEC 61131-3 können Methoden ebenso wie normale Funktionen zusätzliche Ein- und Ausgänge besitzen. Die Ein- und Ausgänge weisen Sie beim Methodenaufruf zu.
  - **Ab TwinCAT 3.1.4026:** Eingänge ohne explizit vorgegebenen Initialwert müssen beim Aufruf der Methode zugewiesen werden. Eingänge mit explizit vorgegebenen Initialwert können optional zugewiesen werden oder beim Aufruf der Methode unbeachtet bleiben.
- Im Implementierungsteil einer Methode ist der Zugriff auf die Funktionsbausteininstanz- oder Programmvariablen erlaubt.
- Um auf die eigene Instanz zu verweisen, verwenden Sie den THIS-Zeiger.
- Sie können auf VAR_TEMP-Variablen des Funktionsbausteins in einer Methode nicht zugreifen.
- Sie können VAR_INST-Variablen deklarieren, für welche bei erneuten Methodenaufrufen keine Neuinitialisierung stattfindet (siehe auch Kapitel Instanzvariablen).

- Durch Verwendung des Rückgabetyps „REFERENCE TO <structured type>“ können Sie direkt beim Aufruf der Methode auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode zurückgeliefert wird, zugreifen. Weitere Informationen dazu finden Sie im Abschnitt „Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf [▶ 98]“.
- Auf VAR_IN_OUT-Variablen des Funktionsbausteins kann in einer Methode prinzipiell zugegriffen werden. Da dieser Zugriff potenziell riskant ist, sollte er mit Bedacht ausgeführt werden. Weitere Informationen dazu finden Sie im Abschnitt „Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition [▶ 99]“.
- Methoden, die in einer Schnittstelle definiert sind, dürfen nur Eingabe-, Ausgabe- und VAR_IN_OUT-Variablen definieren, aber keine Implementierung enthalten.

Beispiel:

Der Code im folgenden Beispiel bewirkt, dass TwinCAT den Rückgabewert und die Ausgänge der Methode auf lokal deklarierte Variablen schreibt.

Deklarationsteil der Methode „Method1“ des Funktionsbausteins FB_Sample:

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

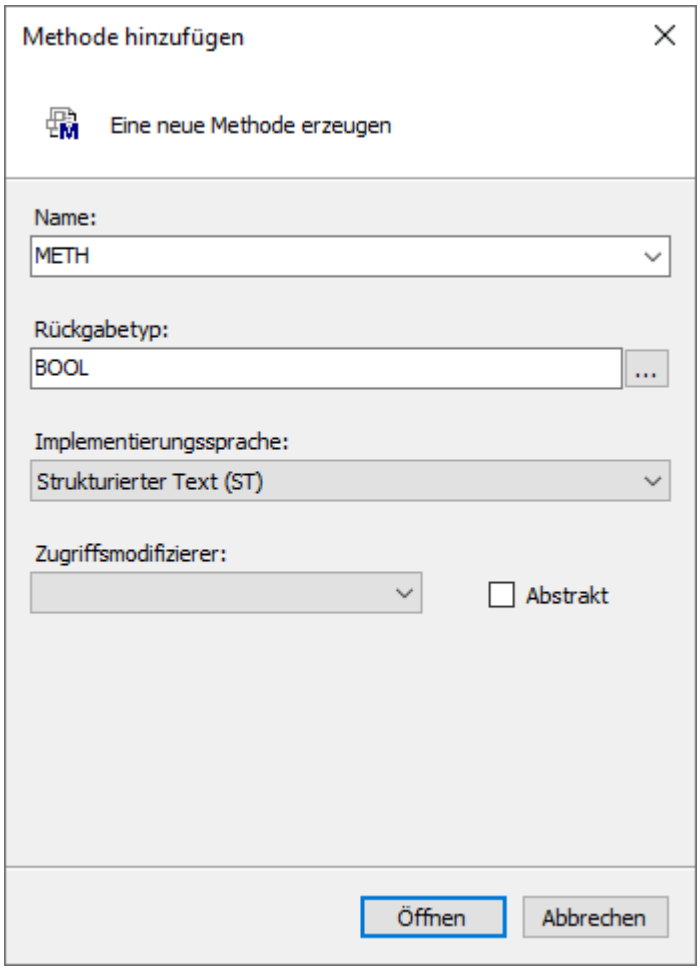
```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

Objekt Methode anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein oder ein Programm.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Methode…**
   ⇨ Der Dialog **Methode hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie einen Rückgabetyp sowie die Implementierungssprache und optional einen Zugriffsmodifizier aus
4. Klicken Sie auf **Öffnen**.

⇨ Das Objekt wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet. Der Editor besteht aus dem Deklarationseditor im oberen Teil und dem Implementierungsteil im unteren Teil.

Dialog Methode hinzufügen



Name	Name der Methode Die Standardmethoden FB_Init und FB_Exit werden in einer Auswahlliste angeboten, wenn sie nicht bereits unterhalb des Bausteins eingefügt sind. Wenn es sich um einen abgeleiteten Funktionsbaustein handelt, bietet die Auswahlliste außerdem alle Methoden des Basisbausteins an.
Rückgabetyp	Typ des Wertes, der zurückgegeben wird
Implementierungssprache	Auswahlliste für die Implementierungssprache

Zugriffsmodifizierer

Zugriffsmodifizierer	Regelt den Zugriff auf die Daten • **PUBLIC**: Der Zugriff ist nicht eingeschränkt (entspricht der Angabe keines Zugriffsmodifizierers). • **PRIVATE**: Der Zugriff auf die Methode ist auf den Funktionsbaustein bzw. das Programm beschränkt. • **PROTECTED**: Der Zugriff auf die Methode ist auf das Programm bzw. den Funktionsbaustein und seine Ableitungen beschränkt. • **INTERNAL**: Der Zugriff auf die Methode ist auf den Namensraum (die Bibliothek) beschränkt. Zusätzlich zu diesen Zugriffsmodifizierern können Sie manuell den Modifizierer FINAL zu einer Methode hinzufügen: • **FINAL**: Überschreibung der Methode in einer Ableitung des Funktionsbausteins ist nicht erlaubt. Das bedeutet, dass die Methode in einer möglicherweise vorhandenen Unterklasse nicht überschrieben/ erweitert werden darf.
Abstrakt	☑ : Kennzeichnet, dass die Methode keine Implementierung hat und die Implementierung durch den abgeleiteten FB bereitgestellt wird. Hintergrundinformationen zu dem Schlüsselwort ABSTRACT finden Sie unter ABSTRACT-Konzept [▶ 212].

Methoden mit einem anderen Zugriffsmodifizierer als PUBLIC werden im Projektmappen-Explorer im SPS-Projektbaum mit einem Signalsymbol gekennzeichnet:

Zugriffsmodifizierer	Objektsymbol	Signalsymbol
PRIVATE		(Schloss)
PROTECTED		(Stern)
INTERNAL		(Herz)

Wenn Sie eine Methode von einer POU zu einer Schnittstelle kopieren oder verschieben, löscht TwinCAT die enthaltenen Implementierungen automatisch.

Spezielle Methoden für einen Funktionsbaustein

FB_init	Deklarationen automatisch implizit. Auch explizite Deklaration möglich. Enthält Initialisierungscode für den Funktionsbaustein, wie im Deklarationsteil des Funktionsbausteins definiert ist. (Methoden FB_init, FB_reinit und FB_exit [▶ 887])
FB_reinit	Explizite Deklaration notwendig. Aufruf, nachdem die Instanz des Funktionsbausteins kopiert wurde (wie während eines Online-Change), und reinitialisiert das neue Instanzmodul. (Methoden FB_init, FB_reinit und FB_exit [▶ 887])
FB_exit	Explizite Deklaration notwendig. Aufruf für jede Instanz des Funktionsbausteins vor einem erneuten Download oder einem Reset oder während eines Online-Change für alle verschobenen oder gelöschten Instanzen. (Methoden FB_init, FB_reinit und FB_exit [▶ 887])
Eigenschaften und Schnittstelleneigenschaften	Bestehen jeweils aus einer Set- und/oder Get-Accessor-Methode. (Objekt Schnittstelleneigenschaft [▶ 112], Objekt Eigenschaft [▶ 100])

Methode aufrufen

Syntax:

```
<return value variable> := <POU name>.<method name>(<method input name> :=
<variable name> (, <further method input name> := <variable name> )* );
```

Beim Methodenaufruf weisen Sie an die Eingabevariablen der Methode Übergabeparameter zu. Beachten Sie dabei die Deklaration. Es genügt, wenn Sie die Namen der Eingangsvariablen angeben, ohne deren Reihenfolge in der Deklaration zu beachten.

Beispiel:

Der Code im folgenden Beispiel bewirkt, dass TwinCAT den Rückgabewert und die Ausgänge der Methode auf lokal deklarierte Variablen schreibt.

Deklarationsteil der Methode „Method1“ des Funktionsbausteins FB_Sample:

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf

Um direkt bei einem Methoden-, Funktions- oder Eigenschaftenaufruf auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode/Funktion/Eigenschaft zurückgeliefert wird, zugreifen zu können, kann folgende Umsetzung verwendet werden. Ein strukturierter Datentyp ist beispielsweise eine Struktur oder ein Funktionsbaustein.

1. Der Rückgabetyp der Methode/Funktion/Eigenschaft wird als „REFERENCE TO <structured type>“ definiert (anstelle von lediglich „<structured type>“).
2. Bei einem solchen Rückgabetyp ist zu beachten, dass – falls beispielsweise eine FB-lokale Instanz des strukturierten Datentyps zurückgeliefert werden soll – der Referenzoperator REF= anstatt des „normalen“ Zuweisungsoperators := verwendet werden muss.

Die Erklärungen und das Beispiel dieses Abschnitts beziehen sich auf den Aufruf einer Eigenschaft. Sie sind aber genauso auf andere Aufrufe übertragbar, die Rückgabewerte liefern (z. B. Methoden oder Funktionen).

Beispiel

Deklaration der Struktur ST_Sample (strukturierter Datentyp):

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

Deklaration des Funktionsbausteins FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

Deklaration der Eigenschaft FB_Sample.MyProp mit dem Rückgabetyp „REFERENCE TO ST_Sample“:

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

Implementierung der Get-Methode von der Eigenschaft FB_Sample.MyProp:

```
MyProp REF= stLocal;
```

Implementierung der Set-Methode von der Eigenschaft FB_Sample.MyProp:

```
stLocal := MyProp;
```

Aufruf der Get- und Set-Methoden im Hauptprogramm MAIN:

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

Durch die Deklaration des Rückgabetyps der Eigenschaft MyProp als „REFERENCE TO ST_Sample“ und durch die Verwendung des Referenzoperators REF= in der Get-Methode dieser Eigenschaft, kann direkt beim Aufruf der Eigenschaft auf ein einzelnes Element des zurückgelieferten strukturierten Datentyps zugegriffen werden.

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

Wenn der Rückgabetyp nur als „ST_Sample“ deklariert wäre, müsste die von der Eigenschaft zurückgelieferte Struktur zunächst einer lokalen Strukturinstanz zugewiesen werden. Die einzelnen Strukturelemente könnten dann anhand der lokalen Strukturinstanz abgefragt werden.

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition/Eigenschaft

Auf die VAR_IN_OUT-Variablen eines Funktionsbausteins kann in einer Methode, einer Transition oder einer Eigenschaft des Funktionsbausteins prinzipiell zugegriffen werden. Bei einem solchen Zugriff ist folgendes zu beachten:

- Wird der Rumpf oder eine Aktion des Funktionsbausteins von außerhalb des FBs aufgerufen, stellt der Compiler sicher, dass bei diesem Aufruf die VAR_IN_OUT-Variablen des Funktionsbausteins zugewiesen werden.
- Bei dem Aufruf einer Methode, Transition oder Eigenschaft des Funktionsbausteins ist dies nicht der Fall, da die VAR_IN_OUT-Variablen des FBs nicht innerhalb eines Methoden-, Transitions- oder Eigenschaftenaufrufs zugewiesen werden können. Es könnte daher gegebenenfalls ein Zugriff auf die VAR_IN_OUT-Variablen stattfinden, indem die Methode/Transition/Eigenschaft aufgerufen wird, bevor die VAR_IN_OUT-Variablen einer gültigen Referenz zugewiesen wurden. Da dies einem ungültigen Zugriff während der Laufzeit entsprechen würde, ist es potentiell riskant, auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft zuzugreifen.

Aus diesem Grund wird bei einem Zugriff auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft folgende Warnung mit der ID C0371 ausgegeben:

„Warning: Access to VAR_IN_OUT <Var> declared in <POU> from external context <Method/Transition/Property>”

Eine adäquate Reaktion auf diese Warnung ist beispielsweise die Überprüfung der VAR_IN_OUT-Variablen innerhalb der Methode/Transition/Eigenschaft, bevor auf sie zugegriffen wird. Diese Überprüfung ist mithilfe des Operators __ISVALIDREF möglich, mit dem geprüft werden kann, ob eine Referenz auf einen gültigen Wert verweist. Ist diese Prüfung vorhanden, kann zum einen davon ausgegangen werden, dass sich der Anwender des Risikos bewusst ist, das potentiell vorhanden ist, wenn auf die VAR_IN_OUT-Variablen des FBs in einer Methode/Transition/Eigenschaft zugegriffen wird. Zum anderen liegt durch die Überprüfung der Referenz ein angemessener Umgang mit diesem Risiko vor. Somit kann die Warnung für diesen überprüften Bereich mittels Attribut 'warning disable' unterdrückt werden.

Die dazugehörige Beispielimplementierung einer Methode ist im Folgenden dargestellt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

Methode FB_Sample.MyMethod:

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
```

```
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

Siehe auch:

- Objektorientiert programmieren [▶ 178]
- Objekt Schnittstelle [▶ 107]
- Implementieren einer Schnittstelle [▶ 208]
- Erweitern eines Funktionsbausteins [▶ 201]
- Methodenaufruf [▶ 210]
- ABSTRACT-Konzept [▶ 212]
- Referenz Programmierung > Instanzvariablen - VAR_INST [▶ 724]

7.4.5 Objekt Eigenschaft

Symbol: 

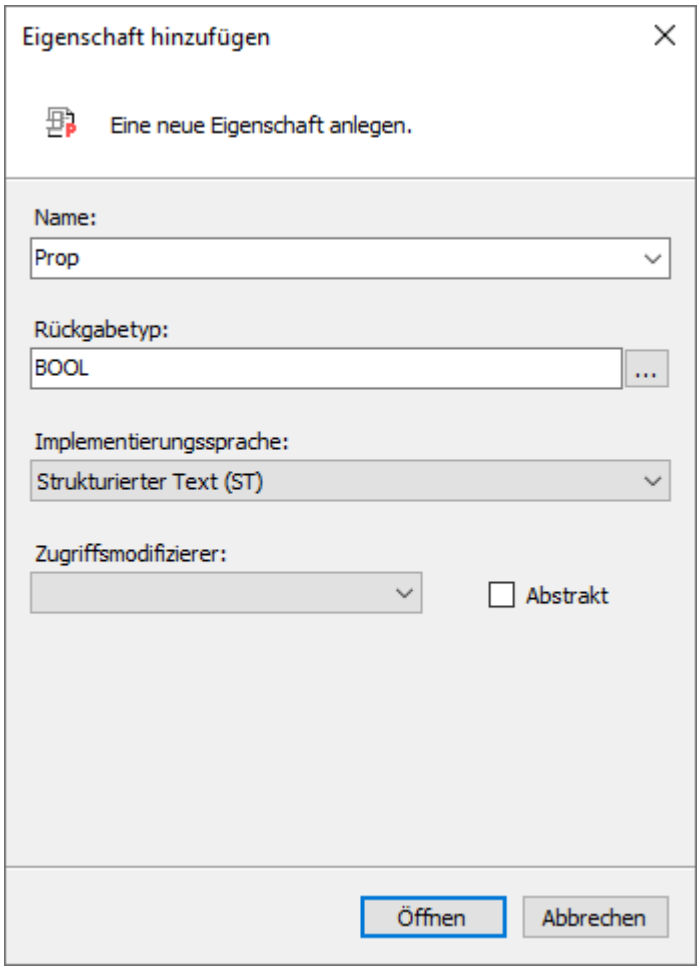
Eine Eigenschaft ist eine Erweiterung der Norm IEC 61131-3 und ist ein Mittel der objektorientierten Programmierung. Sie besteht aus den Accessor-Methoden Get und Set. TwinCAT ruft diese Methoden automatisch auf, sobald ein Lese- oder Schreibzugriff auf den Funktionsbaustein erfolgt, der die Eigenschaft implementiert.

Objekt Eigenschaft anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein oder ein Programm.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Eigenschaft…**
   ⇨ Der Dialog **Eigenschaft hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie einen Rückgabetyp sowie die Implementierungssprache und optional einen Zugriffsmodifizier aus.
4. Klicken Sie auf **Öffnen**.

⇨ Das Objekt wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog Eigenschaft hinzufügen



Name	Name der Eigenschaft
Rückgabetyp	Typ des Wertes, der zurückgegeben wird (Standardtyp oder strukturierter Typ)
Implementierungssprache	Auswahlliste für die Implementierungssprache

Zugriffsmodifizierer

Zugriffsbezeichner	Regelt den Zugriff auf die Daten • **PUBLIC**: Der Zugriff ist nicht eingeschränkt (entspricht der Angabe keines Zugriffsmodifizierers). • **PRIVATE**: Der Zugriff auf die Eigenschaft ist auf den Funktionsbaustein bzw. das Programm beschränkt. • **PROTECTED**: Der Zugriff auf die Eigenschaft ist auf das Programm bzw. den Funktionsbaustein und seine Ableitungen beschränkt. • **INTERNAL**: Der Zugriff auf die Eigenschaft ist auf den Namensraum (die Bibliothek) beschränkt. Zusätzlich zu diesen Zugriffsmodifizierern können Sie manuell den Modifizierer FINAL zu einer Eigenschaft hinzufügen: **FINAL**: Überschreibung der Eigenschaft in einer Ableitung des Funktionsbausteins ist nicht erlaubt. Das bedeutet, dass die Eigenschaft in einer möglicherweise vorhandenen Unterklasse nicht überschrieben/ erweitert werden darf.
Abstrakt	☑ : Kennzeichnet, dass die Eigenschaft keine Implementierung hat und die Implementierung durch den abgeleiteten FB bereitgestellt wird. Hintergrundinformationen zu dem Schlüsselwort ABSTRACT finden Sie unter ABSTRACT-Konzept [▶ 212].

Eigenschaften mit einem anderen Zugriffsmodifizierer als PUBLIC werden im Projektmappen-Explorer im SPS-Projektbaum mit einem Signalsymbol gekennzeichnet:

Zugriffsmodifizierer	Objektsymbol	Signalsymbol
PRIVAT		 (Schloss)
PROTECTED		 (Stern)
INTERNAL		 (Herz)

Eine Eigenschaft kann zusätzliche lokale Variablen enthalten. Eine Eigenschaft kann aber keine zusätzlichen Eingänge enthalten und, im Gegensatz zu einer Funktion oder Methode, keine zusätzlichen Ausgänge.

Wenn Sie eine Eigenschaft von einer POU zu einer Schnittstelle kopieren oder verschieben, löscht TwinCAT die enthaltenen Implementierungen automatisch.

Get- und Set-Accessoren

Die Accessor-Methoden Get und Set fügt TwinCAT automatisch im SPS-Projektbaum unterhalb des Eigenschaftenobjekts ein. Mithilfe des Befehls **Hinzufügen** können Sie sie auch explizit hinzufügen.

TwinCAT ruft den Set-Accessor auf, wenn auf die Eigenschaft schreibend zugegriffen wird, das bedeutet, Sie verwenden den Namen der Eigenschaft als Eingabeparameter.

TwinCAT ruft den Get-Accessor auf, wenn auf die Eigenschaft lesend zugegriffen wird, das bedeutet, Sie verwenden den Namen der Eigenschaft als Ausgabeparameter.

Bitte beachten Sie, dass Sie die Accessor-Methoden implementieren müssen, um auf die Eigenschaft zuzugreifen.

Beispiel für die Implementierung

Deklaration des Funktionsbausteins FB_Sample

```
FUNCTION_BLOCK FB_Sample
VAR
    nVar : INT;
END_VAR
```

```
nVar := nVar + 1;
```

Deklaration der Eigenschaft nValue

```
PROPERTY PUBLIC nValue : INT
```

Implementierung der Accessormethode FB_Sample.nValue.Set

```
nVar := nValue;
```

Implementierung der Accessormethode FB_Sample.nValue.Get

```
nValue := nVar;
```

Aufruf der Eigenschaft nValue

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
END_VAR
```

```
fbSample();
If fbSample.nValue > 500 THEN
    fbSample.nValue := 0;
END_IF;
```

Wenn Sie die Eigenschaft nur für den Lesezugriff oder nur für den Schreibzugriff verwenden wollen, können Sie den Accessor löschen, den Sie nicht verwenden.

Sie können den Accessor-Methoden an folgenden Stellen Zugriffsbezeichner hinzufügen:

- Im Deklarationsteil des Accessors durch manuelles Eintragen.
- Im Dialog **'get'-Accessor hinzufügen** oder **'set'-Accessor hinzufügen**, wenn Sie den Accessor explizit mit dem Befehl **Hinzufügen** einfügen.

Kompatibilitätswarnung bei Eigenschaften vom Typ REFERENCE

Zu TC3.1 Build 4022 ändert sich das Aufrufverhalten von Eigenschaften, die mit dem Rückgabetyp 'REFERENCE TO <…>' definiert sind.

Bei Schreibzugriffen mittels ':=' wird mit Versionen < 3.1.4022.0 der Set-Accessor aufgerufen, sodass die Referenz geschrieben wird. Mit Versionen >= 3.1.4022.0 wird hingegen der Get-Accessor aufgerufen, sodass der Wert geschrieben wird. Um mit Versionen >= 3.1.4022.0 die Referenz zuzuweisen, muss der Referenz-Zuweisungsoperator 'REF= [▶ 661]' verwendet werden.

Monitoring für Eigenschaften im Onlinebetrieb

Für das Monitoring für Eigenschaften im Onlinebetrieb stehen folgende Pragmas zur Verfügung, die Sie an oberster Stelle in der Definition der Eigenschaft einfügen (Attribut 'monitoring' [▶ 846]):

- {attribute 'monitoring':= 'variable'}: Bei jedem Zugriff auf die Eigenschaft speichert TwinCAT den Istwert in einer Variablen und stellt den Wert dieser Variablen dar. Dieser Wert kann unter Umständen veralten, wenn im Code kein Zugriff mehr auf die Eigenschaft erfolgt.
- {attribute 'monitoring' := 'call'}: Bei jeder Darstellung des Werts ruft TwinCAT den Code des Get-Accessors auf. Wenn dieser Code einen Seiteneffekt enthält, dann wird der Seiteneffekt durch das Monitoring ausgeführt.

Sie können eine Eigenschaft mithilfe folgender Funktionalitäten monitoren:

- Inline-Monitoring
  Voraussetzung: In den TwinCAT-Optionen in der Kategorie **TwinCAT > SPS Programmierumgebung > Texteditor** ist in der Registerkarte **Monitoring** die Option **Inline-Monitoring aktivieren** aktiviert.
- Überwachungsliste (Überwachungslisten verwenden [▶ 239])

Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf

Um direkt bei einem Methoden-, Funktions- oder Eigenschaftenaufruf auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode/Funktion/Eigenschaft zurückgeliefert wird, zugreifen zu können, kann folgende Umsetzung verwendet werden. Ein strukturierter Datentyp ist beispielsweise eine Struktur oder ein Funktionsbaustein.

1. Der Rückgabetyp der Methode/Funktion/Eigenschaft wird als „REFERENCE TO <structured type>“ definiert (anstelle von lediglich „<structured type>“).
2. Bei einem solchen Rückgabetyp ist zu beachten, dass – falls beispielsweise eine FB-lokale Instanz des strukturierten Datentyps zurückgeliefert werden soll – der Referenzoperator REF= anstatt des „normalen“ Zuweisungsoperators := verwendet werden muss.

Die Erklärungen und das Beispiel dieses Abschnitts beziehen sich auf den Aufruf einer Eigenschaft. Sie sind aber genauso auf andere Aufrufe übertragbar, die Rückgabewerte liefern (z. B. Methoden oder Funktionen).

Beispiel

Deklaration der Struktur ST_Sample (strukturierter Datentyp):

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

Deklaration des Funktionsbausteins FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

Deklaration der Eigenschaft FB_Sample.MyProp mit dem Rückgabetyp „REFERENCE TO ST_Sample“:

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

Implementierung der Get-Methode von der Eigenschaft FB_Sample.MyProp:

```
MyProp REF= stLocal;
```

Implementierung der Set-Methode von der Eigenschaft FB_Sample.MyProp:

```
stLocal := MyProp;
```

Aufruf der Get- und Set-Methoden im Hauptprogramm MAIN:

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

Durch die Deklaration des Rückgabetyps der Eigenschaft MyProp als „REFERENCE TO ST_Sample“ und durch die Verwendung des Referenzoperators REF= in der Get-Methode dieser Eigenschaft, kann direkt beim Aufruf der Eigenschaft auf ein einzelnes Element des zurückgelieferten strukturierten Datentyps zugegriffen werden.

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

Wenn der Rückgabetyp nur als „ST_Sample“ deklariert wäre, müsste die von der Eigenschaft zurückgelieferte Struktur zunächst einer lokalen Strukturinstanz zugewiesen werden. Die einzelnen Strukturelemente könnten dann anhand der lokalen Strukturinstanz abgefragt werden.

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition/Eigenschaft

Auf die VAR_IN_OUT-Variablen eines Funktionsbausteins kann in einer Methode, einer Transition oder einer Eigenschaft des Funktionsbausteins prinzipiell zugegriffen werden. Bei einem solchen Zugriff ist folgendes zu beachten:

- Wird der Rumpf oder eine Aktion des Funktionsbausteins von außerhalb des FBs aufgerufen, stellt der Compiler sicher, dass bei diesem Aufruf die VAR_IN_OUT-Variablen des Funktionsbausteins zugewiesen werden.
- Bei dem Aufruf einer Methode, Transition oder Eigenschaft des Funktionsbausteins ist dies nicht der Fall, da die VAR_IN_OUT-Variablen des FBs nicht innerhalb eines Methoden-, Transitions- oder Eigenschaftenaufrufs zugewiesen werden können. Es könnte daher gegebenenfalls ein Zugriff auf die VAR_IN_OUT-Variablen stattfinden, indem die Methode/Transition/Eigenschaft aufgerufen wird, bevor die VAR_IN_OUT-Variablen einer gültigen Referenz zugewiesen wurden. Da dies einem ungültigen Zugriff während der Laufzeit entsprechen würde, ist es potentiell riskant, auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft zuzugreifen.

Aus diesem Grund wird bei einem Zugriff auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft folgende Warnung mit der ID C0371 ausgegeben:

„Warning: Access to VAR_IN_OUT <Var> declared in <POU> from external context <Method/Transition/Property>”

Eine adäquate Reaktion auf diese Warnung ist beispielsweise die Überprüfung der VAR_IN_OUT-Variablen innerhalb der Methode/Transition/Eigenschaft, bevor auf sie zugegriffen wird. Diese Überprüfung ist mithilfe des Operators __ISVALIDREF möglich, mit dem geprüft werden kann, ob eine Referenz auf einen gültigen Wert verweist. Ist diese Prüfung vorhanden, kann zum einen davon ausgegangen werden, dass sich der Anwender des Risikos bewusst ist, das potentiell vorhanden ist, wenn auf die VAR_IN_OUT-Variablen des FBs in einer Methode/Transition/Eigenschaft zugegriffen wird. Zum anderen liegt durch die Überprüfung der Referenz ein angemessener Umgang mit diesem Risiko vor. Somit kann die Warnung für diesen überprüften Bereich mittels Attribut 'warning disable' unterdrückt werden.

Die dazugehörige Beispielimplementierung einer Methode ist im Folgenden dargestellt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

Methode FB_Sample.MyMethod:

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
```

```
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

Initialisierungsbeispiel:

Im folgenden Beispiel werden eine Eingangsvariable und eine Eigenschaft von einem Funktionsbaustein initialisiert, welcher über eine FB_init-Methode [▶ 888] mit einem zusätzlichen Parameter verfügt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

Methode FB_Sample.FB_init:

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

Eigenschaft FB_Sample.nMyProperty und die zugehörige Set-Funktion:

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

Initialisierungsergebnis:

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8
  - nLocalInitParam = 7
  - nLocalProp = 9

Siehe auch:

- Objekt Schnittstelleneigenschaft [▶ 112]
- Objektorientiert programmieren [▶ 178]

- Erweitern eines Funktionsbausteins [▶ 201]
- Referenz Programmierung > Methoden FB_init, FB_reinit und FB_exit [▶ 887]

7.4.6 Objekt Schnittstelle

Symbol: 

Schlüsselwort: INTERFACE

Eine Schnittstelle ist ein Mittel der objektorientierten Programmierung. Das Objekt **Schnittstelle** beschreibt ein Set von Methoden- und Eigenschaft-Prototypen. Prototyp bedeutet in diesem Zusammenhang, dass die Methoden und Eigenschaften nur Deklarationen und keine Implementierung enthalten.

Auf diese Weise können Sie verschiedene Funktionsbausteine, die gemeinsame Eigenschaften haben, gleichartig nutzen.

Dem Objekt **Schnittstelle** können Sie die Objekte **Schnittstelleneigenschaft** und **Schnittstellenmethode** hinzufügen.

Objekt Schnittstelle anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Schnittstelle…**
   ⇨ Der Dialog **Schnittstelle hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie optional eine Schnittstelle aus, welche erweitert werden soll.
4. Klicken Sie auf **Öffnen**.

⇨ Die Schnittstelle wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog Schnittstelle hinzufügen

Name	Schnittstellenname

Vererbung

Erweitert	 : Erweitert die Schnittstelle, die Sie im Eingabefeld eingeben oder über die Eingabehilfe  auswählen. Dies bedeutet, dass alle Methoden der Schnittstelle, die die neue Schnittstelle erweitert, auch in der neuen Schnittstelle verfügbar sind.

Anwendungsfälle einer Schnittstelle

1. Prüfung der Schnittstellenvereinbarung mittels Compiler

- In einer Schnittstelle (z. B. I_Sample) deklarieren Sie die gewünschten Methoden und Eigenschaften (einschließlich Rückgabetyp, Eingänge etc.), die zu dieser Schnittstelle gehören sollen.
- In den Funktionsbausteinen, die dieser Schnittstelle entsprechen sollen und somit die zugehörigen Methoden und Eigenschaften bereitstellen sollen, implementieren Sie die Schnittstelle I_Sample.
  - Der Funktionsbaustein enthält die Schnittstelle dabei in seiner IMPLEMENTS-Liste innerhalb seines Deklarationsteils (z. B. FB_Sample IMPLEMENTS I_Sample).
  - Ein Funktionsbaustein kann eine oder mehrere Schnittstellen implementieren (z. B. FB_Sample IMPLEMENTS I_Sample1, I_Sample2).
- Ein Funktionsbaustein, der eine Schnittstelle implementiert, muss alle Methoden und Eigenschaften enthalten, die in dieser Schnittstelle definiert sind (Schnittstellenmethoden und -eigenschaften). Dabei muss die Deklaration der Methoden und Eigenschaften exakt der Deklaration in der Schnittstelle entsprechen (Name, Rückgabetype, Eingänge, Ausgänge).

- Die Funktionsbausteine fügen den Schnittstellenmethoden und Schnittstelleneigenschaften Funktionsbaustein-spezifischen Code hinzu. Wenn eine Schnittstelle von mehreren Funktionsbausteinen implementiert wird, können Sie die gleiche Methode mit identischen Parametern, aber unterschiedlichem Implementierungscode in verschiedenen Funktionsbausteinen verwenden.
- Für die Funktionsbausteine, die eine oder mehrere Schnittstellen implementieren, überprüft der Compiler, ob sich die Funktionsbausteine an die jeweiligen Schnittstellenvereinbarungen halten. Wenn sich die Deklaration der Elemente in der Schnittstelle und in dem Funktionsbaustein unterscheiden oder wenn die Schnittstelle über weitere Elemente verfügt, die der Funktionsbaustein nicht enthält, meldet der Compiler einen Fehler.

2. Aufruf von Methoden und Eigenschaften einer Funktionsbausteininstanz über eine Schnittstellenvariable

Neben der automatischen Prüfung der Schnittstellenvereinbarung mittels Compiler können Sie Schnittstellen verwenden, um über eine Schnittstellenvariable eine Schnittstellenmethode oder Schnittstelleneigenschaft einer Funktionsbausteininstanz aufzurufen.

- Sie instanziieren zunächst sowohl die Schnittstelle (z. B. `iSample : I_Sample;`) als auch den bzw. die Funktionsbausteine, die die Schnittstelle ordnungsgemäß implementieren (siehe Anwendungsfall 1: Prüfung der Schnittstellenvereinbarung mittels Compiler).
- Anschließend können Sie der Schnittstellenvariablen jede Instanz eines Funktionsbausteins zuweisen, der die Schnittstelle ordnungsgemäß implementiert. Wenn für eine Schnittstellenvariable noch keine Zuweisung erfolgt ist, enthält die Variable im Onlinebetrieb den Wert 0.
- Im letzten Schritt können Sie über die Schnittstellenvariable eine Schnittstellenmethode oder -eigenschaft aufrufen, wobei die Methode bzw. Eigenschaft für die Funktionsbausteininstanz aufgerufen wird, auf die die Schnittstelle verweist.
- Über eine solche Umsetzung können Sie verschiedene, aber gleichartige Funktionsbausteine über die Schnittstellenvariable einheitlich verwenden. Abhängig vom Projektzustand können Sie der Schnittstellenvariablen beispielsweise eine bestimmte Funktionsbausteininstanz zuweisen, sodass der Aufruf der Schnittstellenmethoden und -eigenschaften identisch ist, aber je nach Projektzustand eine andere Funktionsbausteininstanz verwendet wird.

Einer Variablen vom Typ einer Schnittstelle müssen Sie die Instanz eines Funktionsbausteins zuweisen, bevor eine Methode oder Eigenschaft über die Schnittstellenvariable aufgerufen werden kann.

Hinweise

- Sie dürfen keine Variablen innerhalb einer Schnittstelle deklarieren. Eine Schnittstelle hat keinen Implementierungsteil und keine Aktionen. Es wird nur eine Sammlung von Methoden und Eigenschaften definiert, welche Deklarationscode, aber keinen Implementierungscode enthalten.
- Eine Variable vom Typ einer Schnittstelle ist eine Referenz auf Instanzen von Funktionsbausteinen. TwinCAT behandelt Variablen, die mit dem Typ einer Schnittstelle deklariert sind, immer als Referenzen.

Schnittstellenreferenzen und Online-Change

- Ab TwinCAT 3.0 Build 3100 werden Schnittstellenreferenzen automatisch umadressiert, sodass in jedem Fall, also auch bei einem Online-Change, die korrekte Schnittstelle referenziert wird. Zusätzlicher Code und mehr Zeit werden dafür benötigt und somit muss abhängig von der Anzahl der betroffenen Objekte mit zeitlichen Problemen gerechnet werden. Daher wird dem Programmierer vor Ausführung des Online-Change die Anzahl der betroffenen Variablen und Schnittstellenreferenzen angezeigt, und er muss dann entschieden, ob der Online-Change durchgeführt oder abgebrochen werden soll.

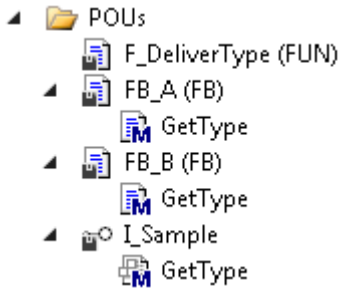
Prüfen von Schnittstellenvariablen

- Den Operator __ISVALIDREF können Sie nur für Operanden vom Typ REFERENCE TO verwenden. Die Prüfung von Schnittstellenvariablen ist mit diesem Operator nicht möglich. Um zu überprüfen, ob einer Schnittstellenvariable bereits eine Funktionsbausteininstanz zugewiesen wurde, können Sie die Schnittstellenvariable auf ungleich 0 prüfen (IF iSample <> 0 THEN …).

Erweitertes Monitoring einer Schnittstellenvariablen

- Ab TwinCAT 3.1 Build 4024 stehen erweiterte Möglichkeiten zum Monitoring/Debugging einer Schnittstellenvariablen zur Verfügung. Dabei kann eine Schnittstellenvariable im Monitoring-Bereich (Deklarationseditor, Überwachungsliste) aufgeklappt werden. Unterhalb der Schnittstellenvariablen werden dann der Symbolpfad und die Onlinedaten der aktuell zugewiesenen FB-Instanz angezeigt.

Beispiel 1



Deklaration der Schnittstelle:

- Sie haben die Schnittstelle I_Sample zu Ihrem Projekt hinzugefügt. Der Schnittstelle fügen Sie die Methode GetType mit dem Rückgabetyp STRING hinzu.
- I_Sample und GetType enthalten keinen Implementierungscode. Die Methode GetType enthält lediglich die benötigten (Variablen-) Deklarationen (z. B. Rückgabetyp). Die Methode GetType können Sie später im Funktionsbaustein ausprogrammieren, der die Schnittstelle I_Sample implementiert.

```
INTERFACE I_Sample
```

Methode I_Sample.GetType:

```
METHOD GetType : STRING
```

Implementierung der Schnittstelle:

- Wenn Sie anschließend einen Funktionsbaustein zum Projekt hinzufügen und in dem Dialog **Hinzufügen** in dem Feld **Implementiert** die Schnittstelle I_Sample angeben, fügt TwinCAT diesem Funktionsbaustein auch automatisch die Methode GetType hinzu. Hier können Sie in den Methoden funktionsbaustein-spezifischen Code implementieren.
- Die Funktionsbausteine FB_A und FB_B implementieren jeweils die Schnittstelle I_Sample:

```
FUNCTION_BLOCK FB_A IMPLEMENTS I_Sample
```

```
FUNCTION_BLOCK FB_B IMPLEMENTS I_Sample
```

- Beide Funktionsbausteine müssen daher eine Methode mit dem Namen GetType und dem Rückgabetyp STRING enthalten. Ansonsten meldet der Compiler einen Fehler (siehe Anwendungsfall 1 im Abschnitt Anwendungsfälle einer Schnittstelle [▶ 107]).

Methode FB_A.GetType:

```
METHOD GetType : STRING
```

```
GetType := 'FB_A';
```

Methode FB_B.GetType:

```
METHOD GetType : STRING
```

```
GetType := 'FB_B';
```

Verwendung der Schnittstelle:

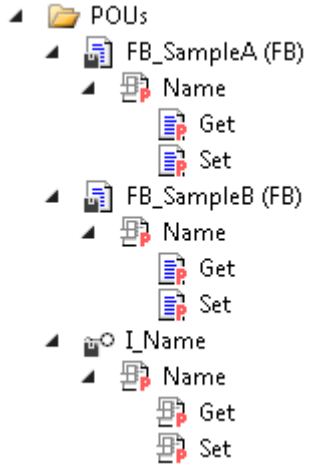
- Eine Funktion F_DeliverType enthält die Deklaration einer Eingangsvariablen vom Typ der Schnittstelle I_Sample. Innerhalb der Funktion wird die Schnittstellenmethode GetType über die Schnittstellenvariable iSample aufgerufen. In diesem Fall hängt es vom übergebenen Funktionsbausteintyp ab, ob FB_A.GetType oder FB_B.GetType aufgerufen wird (siehe Anwendungsfall 2 im Abschnitt Anwendungsfälle einer Schnittstelle [▶ 107]).

```
FUNCTION F_DeliverType : STRING
VAR_INPUT
    iSample : I_Sample;
END_VAR

F_DeliverType := iSample.GetType();
```

- Instanzen von Funktionsbausteinen, die die Schnittstelle I_Sample implementieren (z. B. FB_A und FB_B), können der Eingangsvariablen der Funktion F_DeliverType zugewiesen werden.
- Beispiele für Funktionsaufrufe:
  - Wenn der Funktion F_DeliverType die Funktionsbausteininstanz fbA übergeben wird, wird innerhalb der Funktion die Methode fbA.GetType aufgerufen, da die Schnittstellenvariable iSample auf die Funktionsbausteininstanz fbA zeigt. Dieser Methodenaufruf liefert den Rückgabewert 'FB_A', welcher wiederum von der Funktion F_DeliverType zurückgegeben wird und im Hauptprogramm der Variablen sResultA zugewiesen wird.
  - Entsprechend erhält sResultB den Wert 'FB_B', da innerhalb der Funktion F_DeliverType die Methode fbB.GetType aufgerufen wird.

```
PROGRAM MAIN
VAR
    fbA       : FB_A;
    fbB       : FB_B;
    sResultA  : STRING;
    sResultB  : STRING;
END_VAR

sResultA := F_DeliverType(iSample := fbA); // call with instance of type FB_A
sResultB := F_DeliverType(iSample := fbB); // call with instance of type FB_B
```

Beispiel 2



Deklaration der Schnittstelle:

- Sie haben die Schnittstelle I_Name zu Ihrem Projekt hinzugefügt. Der Schnittstelle fügen Sie die Eigenschaft Name mit dem Rückgabetyp STRING hinzu. Die Eigenschaft besitzt die Accessor-Methoden Get und Set. Der Get-Accessor soll dazu dienen, den Namen irgendeines Objekts aus einem Funktionsbaustein auszulesen, der die Schnittstelle implementiert. Den Set-Accessor verwenden Sie dazu, den Namen in diesen Funktionsbaustein zu schreiben.
- I_Name und Name enthalten keinen Implementierungscode. Die Eigenschaft Name enthält lediglich die benötigte Deklaration (Rückgabetyp). Die Get- und Set-Methoden können Sie innerhalb der Schnittstellendefinition somit nicht bearbeiten, jedoch später im Funktionsbaustein, der die Schnittstelle I_Name implementiert.

```
INTERFACE I_Name
```

Eigenschaft I_Name.Name:

```
PROPERTY Name : STRING
```

Implementierung der Schnittstelle:

- Die Funktionsbausteine FB_SampleA und FB_SampleB implementieren die Schnittstelle I_Name.

- Wenn die Schnittstelle beispielsweise beim Anlegen der Funktionsbausteine im Dialog **Hinzufügen** angegeben wird, fügt TwinCAT die Eigenschaft Name mit den Get- und Set-Methoden automatisch unterhalb der Funktionsbausteine FB_SampleA und FB_SampleB ein.
- Unterhalb der Funktionsbausteine können Sie die Accessor-Methoden editieren, beispielsweise so, dass die Variable sVar1 gelesen wird und Sie somit den Namen eines Objekts erhalten. In FB_SampleB, der dieselbe Schnittstelle I_Name implementiert, können Sie in der Get-Methode Code implementieren, der dann den Namen eines anderen Objekts liefert. Die Set-Methode verwenden Sie, um den Namen, den das Programm MAIN liefert ('abc'), in den Funktionsbaustein FB_SampleB zu schreiben.
- Die Funktionsbausteine FB_SampleA und FB_SampleB implementieren jeweils die Schnittstelle I_Name:

```
FUNCTION_BLOCK FB_SampleA IMPLEMENTS I_Name
VAR
    sVar1 : STRING := 'My name is A.';
END_VAR

FUNCTION_BLOCK FB_SampleB IMPLEMENTS I_Name
VAR
    sVar2 : STRING := 'My name is B.';
END_VAR
```

- Beide Funktionsbausteine müssen daher eine Eigenschaft mit dem Namen Name und dem Rückgabetyp STRING enthalten. Die Eigenschaft muss über eine Get- und eine Set-Methode verfügen. Ansonsten meldet der Compiler einen Fehler (siehe Anwendungsfall 1 im Abschnitt Anwendungsfälle einer Schnittstelle [▶ 107]).

FB_SampleA.Name.Get:

```
Name := sVar1;
```

FB_SampleA.Name.Set:

```
sVar1 := Name;
```

FB_SampleB.Name.Get:

```
Name := sVar2;
```

FB_SampleB.Name.Set:

```
sVar2 := Name;
```

Verwendung der Schnittstelle:

- Auf die Eigenschaften der Funktionsbausteine kann sowohl über die entsprechenden Funktionsbausteininstanzen als auch über eine Schnittstellenvariable vom Typ I_Name zugegriffen werden. Voraussetzung für den Zugriff über eine Schnittstellenvariable ist, dass dieser Variablen zuvor eine konkrete Funktionsbausteininstanz zugewiesen wurde, die die Schnittstelle I_Name implementiert.

```
PROGRAM MAIN
VAR
    iName     : I_Name;

    fbSampleA : FB_SampleA;
    sNameA    : STRING;       // will be 'My name is A.'

    fbSampleB : FB_SampleB;
    sNameB    : STRING;       // will be 'My name is B.' after first cycle
                              // and will be 'New name' afterwards
END_VAR

// assign FB instance fbSample1 to interface variable
iName := fbSampleA;

// access to name property of fbSample1 via interface variable (Get)
sNameA := iName.Name;

// access to name property of fbSample2 via FB instance (Get and Set)
sNameB := fbSampleB.Name;
fbSampleB.Name := 'New name';
```

Siehe auch:

- Implementieren einer Schnittstelle [▶ 208]
- Erweitern einer Schnittstelle [▶ 208]

7.4.6.1 Objekt Schnittstellenmethode

Symbol: 

Eine Schnittstellenmethode ist ein Mittel der objektorientierten Programmierung. Das Objekt **Schnittstellenmethoden** fügen Sie einer Schnittstelle über den Befehl **Hinzufügen > Method…** hinzu.

Wenn eine Methode unterhalb einer Schnittstelle eingefügt ist, können Sie in dieser Methode nur Variablendeklarationen (Eingabe-, Ausgabe- und Ein/Ausgabe-Variablen) hinzufügen und instanziieren.

Der Methode können Sie erst dann Programmcode hinzufügen, wenn ein Funktionsbaustein die Schnittstelle, zu der die Methode gehört, „implementiert“. TwinCAT fügt die Methode dann unterhalb des Funktionsbausteins ein.

Siehe auch:

- Objekt Schnittstelle [▶ 107]
- Objekt Methode [▶ 93]
- Implementieren einer Schnittstelle [▶ 208]

7.4.6.2 Objekt Schnittstelleneigenschaft

Symbol: 

Eine Schnittstelleneigenschaft ist ein Mittel der objektorientierten Programmierung. Das Objekt **Schnittstelleneigenschaft** fügen Sie einer Schnittstelle über den Befehl **Hinzufügen > Property…** hinzu, um die Beschreibung der Schnittstelle mit den Accessor-Methoden Get und/oder Set zu erweitern. Für die Accessor-Methoden ist in der Schnittstelleneigenschaft kein Implementierungscode enthalten. Wenn Sie den Set-Accessor löschen, dann besteht nur ein lesender Zugriff auf die Eigenschaft, und kein schreibender Zugriff.

Der Get-Accessor dient dem Lesezugriff auf die Eigenschaft.
Der Set-Accessor dient dem Schreibzugriff auf die Eigenschaft.

Wenn die Eigenschaft kein Get und/oder Set hat, fügen Sie diesen Accessor der Schnittstelleneigenschaft mit dem Befehl **Hinzufügen** hinzu.

Wenn Sie einen Funktionsbaustein oder ein Programm mit einer Schnittstelle erweitern, die Eigenschaften enthält, fügt TwinCAT diese und ihre zugehörigen Get- und/oder Set-Accessoren automatisch im SPS-Projektbaum unterhalb der POU im SPS-Projektbaum ein. Anschließend können Sie in den Get- und/oder Set-Accessoren den Code implementieren.

Siehe auch:

- Objekt Schnittstelle [▶ 107]
- Objekt Eigenschaft [▶ 100]
- Implementieren einer Schnittstelle [▶ 208]

7.5 Quellcode in IEC erstellen

Quellcode

Mit „Quellcode“ bezeichnen wir den Implementierungscode, den Sie mithilfe der entsprechenden Programmiersprachen-Editoren in den Programmierbausteinen einfügen.

Programmiersprache

Beim Anlegen eines Programmierbausteins entscheiden Sie, in welcher Implementierungssprache Sie programmieren wollen. Neben den IEC-Sprachen steht auch CFC zur Verfügung.

Programmiersprachen-Editoren

Ein Programmierbaustein wird zur Bearbeitung im entsprechenden Programmiersprachen-Editor geöffnet, wenn Sie einen Doppelklick auf das Objekt im SPS-Projektbaum ausführen. Der Baustein erscheint also entweder im textuellen ST-Editor oder in einem der grafischen Editoren für FUP/KOP/AWL, AS oder CFC. Jeder Editor besteht aus zwei Fenstern: Im oberen Fenster nehmen Sie die Deklarationen vor, je nach Einstellung in textueller oder tabellarischer Form. Im unteren Fenster fügen Sie den Implementierungscode ein. Die Darstellung und das Verhalten jedes Editors können Sie projektweit in der zugehörigen Registerkarte der TwinCAT-Optionen konfigurieren.

Beachten Sie die Möglichkeit, einen Programmierbaustein auch während des Onlinebetriebs der Anwendung im Offlinezustand des Editors zu öffnen. (Befehl Objekt (offline) bearbeiten [▶ 926])

Siehe auch:

- Referenz Programmierung > Programmiersprachen und ihre Editoren [▶ 654]

7.5.1 FUP/KOP/AWL

Ein kombinierter Editor ermöglicht das Programmieren in den Sprachen FUP (Funktionsplan), KOP (Kontaktplan) und AWL (Anweisungsliste).

Die Basiseinheit der FUP- und KOP-Programmierung ist ein Netzwerk. Jedes Netzwerk enthält eine Struktur, die Folgendes darstellen kann: Einen logischen oder arithmetischen Ausdruck, den Aufruf eines Programmierbausteins (Funktion, Funktionsbaustein, Programm usw.), einen Sprung, oder eine Return-Anweisung. AWL benötigt eigentlich keine Netzwerke. In TwinCAT besteht jedoch auch ein AWL-Programm aus mindestens einem Netzwerk, um das Konvertieren in FUP oder KOP zu unterstützen. Im Hinblick darauf sollten Sie auch ein AWL-Programm sinnvoll in Netzwerke unterteilen.

Siehe auch:

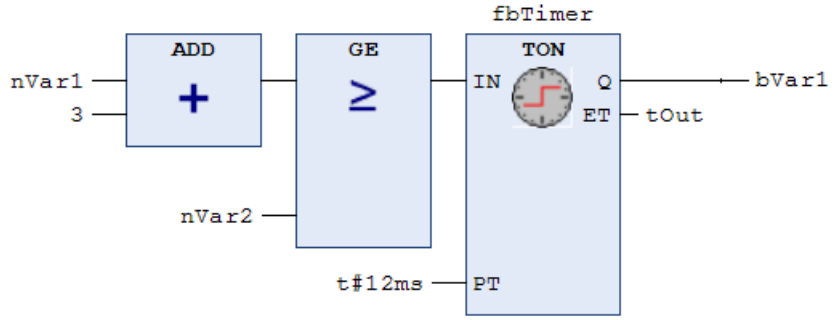
- Referenz Programmierung: Funktionsplan/Kontaktplan/Anweisungsliste (FUP/KOP/AWL) [▶ 685]
- Dokumentation TC3 User Interface: FUP/KOP/AWL [▶ 1072]

Funktionsplan (FUP)

Der Funktionsplan ist eine graphisch orientierte IEC 61131 Programmiersprache. Er arbeitet mit einer Liste von Netzwerken. Dabei enthält jedes Netzwerk eine Struktur, die logische und arithmetische Ausdrücke, Aufrufe von Funktionsblöcken, einen Sprung oder eine Return-Anweisung enthalten kann.

Hierbei werden Bausteine benutzt, die aus der booleschen Algebra bekannt sind. Bausteine und Variablen werden mit Verbindungslinien verknüpft. Der Signalfluss verläuft im Netzwerk von links nach rechts. Der Signalfluss im Editor verläuft von oben nach unten, beginnend mit Netzwerk 1.

Beispiel:



Kontaktplan (KOP)

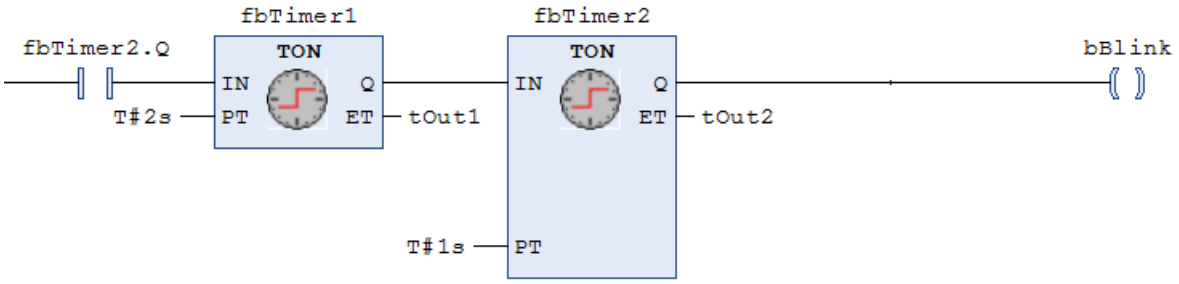
Der Kontaktplan (KOP) ist eine grafisch orientierte Programmiersprache, die dem Prinzip eines elektrischen Schaltplans angenähert ist.

Einerseits eignet sich der Kontaktplan dazu, logische Schaltwerke zu konstruieren, andererseits kann man aber auch Netzwerke wie im FUP erstellen. Daher können Sie den KOP sehr gut zum Steuern von Aufrufen anderer Programmbausteine verwenden.

Der Kontaktplan besteht aus einer Folge von Netzwerken. Ein Netzwerk wird auf der linken Seite von einer vertikalen Linie (Stromschiene) begrenzt. Ein Netzwerk enthält einen Schaltplan aus Kontakten, Spulen, optional Bausteinen (POUs) und Verbindungslinien. Auf der linken Seite eines Netzwerks gibt es einen Kontakt oder eine Folge von Kontakten, die von links nach rechts den Zustand ON oder OFF weitergeben, was den booleschen Werten TRUE und FALSE entspricht. Zu jedem Kontakt gehört eine boolesche Variable. Wenn diese Variable TRUE ist, dann wird der Zustand über die Verbindungslinie von links nach rechts weitergegeben. Ansonsten wird OFF weitergegeben. Somit erhält/erhalten die Spule(n) im rechten Teil des Netzwerks den von links kommenden Wert ON oder OFF und dementsprechend wird der Wert TRUE oder FALSE in die ihnen zugewiesene boolesche Variable geschrieben.

Wenn die Elemente in Reihe geschaltet werden, dann bedeutet dies eine UND-Verknüpfung. Wenn sie parallel geschaltet werden, dann bedeutet dies eine ODER-Verknüpfung. Ein Strich durch ein Element bedeutet eine Negierung des Elements. Die Negation eines Eingangs oder Ausgangs wird durch ein Kreissymbol angezeigt.

Beispiel:



Die IEC 61131-3 definiert einen vollständigen KOP-Befehlssatz, bestehend aus verschiedenen Typen von Kontakten und Spulen. Kontakte leiten den Strom (entsprechend ihres Typs) von links nach rechts. Spulen speichern den eingehenden Wert. Kontakte und Spulen sind booleschen Variablen zugeordnet. Sie können ein KOP-Netzwerk durch Sprünge, Rücksprünge, Marken und Kommentare ergänzen.

Anweisungsliste (AWL)

Die Anweisungsliste ist eine Assembler-ähnliche IEC 61131-konforme Programmiersprache. Sie unterstützt Akkumulator-basiertes Programmieren.

Eine Anweisungsliste (AWL) besteht aus einer Folge von Anweisungen. Jede Anweisung beginnt in einer neuen Zeile und beinhaltet einen Operator und, je nach Art der Operation, einen oder mehrere durch Kommata separierte Operanden. Vor einer Anweisung kann eine Marke stehen, gefolgt von einem Doppelpunkt. Sie dient der Kennzeichnung der Anweisung und Sie können die Marke als Sprungziel verwenden. Ein Kommentar muss das letzte Element in einer Zeile sein. Leere Zeilen können zwischen Anweisungen eingefügt werden.

Alle IEC 61131-3 Operatoren werden unterstützt, ebenso mehrfache Eingänge, mehrfache Ausgänge, Negationen, Kommentare, Set/Reset von Ausgängen und bedingte/unbedingte Sprünge.

Jede Anweisung basiert primär auf dem Laden von Werten in den Akkumulator (LD-Anweisung). Danach wird die entsprechende Operation mit dem Parameter aus dem Akkumulator ausgeführt. Das Ergebnis der Operation wird wieder in den Akkumulator geschrieben, von wo Sie es mithilfe einer ST-Anweisung gezielt speichern sollten.

Für das Programmieren von bedingten Ausführungen oder Schleifen unterstützt die Anweisungsliste Vergleichsoperatoren (EQ, GT, LT, GE, LE, NE) und Sprünge. Sprünge können unbedingt (JMP) oder bedingt (JMPC / JMPCN) sein. Bei bedingten Sprüngen wird geprüft, ob der Wert im Akkumulator TRUE oder FALSE ist.

Beispiel:

```
PROGRAM IL
VAR
    fbTimer1 : TON;
    fbTimer2 : TON;
    bVar : BOOL;
    nVar : INT;
    tIn1 : TIME;
    tOut1 : TIME;
END_VAR
```

```
    LD          bVar
    ST          fbTimer1.IN     starts timer with rising edge, resets time…
    JMP         mark1
    CAL         fbTimer1(
                PT:=tIn1,
                ET:=>tOut1)
    LD          fbTimer1.Q      gets TRUE, delay time (PT) after a rising…
    ST          fbTimer2.IN     starts timer with rising edge, resets time…

mark1:
    LD          nVar
    ADD         230
```

7.5.1.1 Programmieren in Funktionsplan (FUP)

Anlegen einer POU in der Implementierungssprache Funktionsplan (FUP)

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU…**
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie die Implementierungssprache „Funktionsplan (FUP)“.
4. Klicken Sie auf **Öffnen**.

⇨ Die POU wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet. Er besteht aus dem Deklarationseditor im oberen Teil und dem Implementierungsteil mit einem leeren Netzwerk im unteren Teil. Automatisch wird auch die Ansicht **Werkzeugkasten** geöffnet, in der die passenden Elemente, Operatoren und Funktionsbausteine für die FUP-Programmierung bereitstehen.

Programmieren eines Netzwerks

1. Klicken Sie in das automatisch eingefügte leere Netzwerk im Implementierungsteil.
   ⇨ Das Netzwerk wird gelb hinterlegt, der Bereich mit der Netzwerknummer am linken Rand wird rot hinterlegt.
2. Öffnen Sie das Kontextmenü mit der rechten Maustaste.
   ⇨ Sie erhalten unter anderem die Einfügebefehle für die an dieser Stelle einfügbaren Elemente.
3. Fügen Sie die für Ihre Programmierung benötigten Elemente über die Menübefehle oder durch Ziehen der Elemente aus der Ansicht **Werkzeugkasten** ein.
4. Wählen Sie beispielsweise den Befehl **Zuweisung einfügen**.
   ⇨ Eine Zuweisungslinie wird eingefügt. Jeweils drei Fragezeichen stehen für Zuweisungsquelle und Zuweisungsziel.
5. Selektieren Sie die Fragezeichen und ersetzen Sie diese mit der gewünschten Variable. Die Eingabehilfe steht zur Verfügung.
6. Bewegen Sie den Cursor über die Zuweisungslinie.
   ⇨ Die möglichen Einfügepositionen für weitere Elemente werden als graue Rauten angezeigt. Ein Klick auf eine Raute selektiert die Position und es stehen wiederum die passenden Einfügebefehle zur Verfügung.

7. Alternativ können Sie ein Element aus der Ansicht **Werkzeugkasten** mit der Maus ins Netzwerk ziehen. Klicken Sie beispielsweise in der Ansicht **Werkzeugkasten** auf das Baustein-Element, halten die Maustaste gedrückt und bewegen den Cursor über das Netzwerk.
   ⇨ Jede mögliche Einfügeposition leuchtet grün auf.
8. Lassen Sie die Maustaste los, um den Baustein einzufügen.
   ⇨ Die Bausteinbox wird im Netzwerk dargestellt. Der Bausteintyp im Inneren und der im Falle eines Funktionsbausteins benötigte Instanzname oberhalb der Box sind noch mit drei Fragezeichen freigehalten.
9. Selektieren Sie die die drei Fragezeichen im Inneren der Box und ersetzen Sie sie mit dem Bausteinnamen. Die Eingabehilfe steht zur Verfügung.

⇨ Die Eingänge und Ausgänge des gewählten Bausteins werden dargestellt. Sie sind noch mit Fragezeichen freigehalten, bei Funktionsbausteinen ebenso der Instanzname.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Zuweisung einfügen [▶ 1073]

Leitungsverzweigungen (Subnetzwerke) programmieren

1. Fügen Sie im Implementierungsteil Ihrer POU ein neues Netzwerk über den Befehl **Netzwerk einfügen** im Menü **FUP/KOP/AWL** oder im Kontextmenü oder aus der Ansicht **Werkzeugkasten** ein.
2. Ziehen Sie beispielsweise einen Operator ADD in das leere Netzwerk und ersetzen Sie die drei Fragezeichen mit zwei Variablen vom Datentyp INT.
3. Ziehen Sie das Element **Leitungsverzweigung** aus der Ansicht **Werkzeugkasten** in Ihre Implementierung und lassen Sie die Maustaste an der grünen Einfügeposition direkt am Ausgang des Operators los.

⇨ Die Leitungsverzweigung spaltet die Abarbeitungslinie am Ausgang der Operator-Box in zwei Subnetzwerke. Jedem der beiden Subnetzwerke können Sie nun weitere FUP-Elemente und auch weitere Leitungsverzweigungen hinzufügen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Netzwerk einfügen [▶ 1072]

7.5.1.2 Programmieren in Kontaktplan (KOP)

Anlegen einer POU in der Implementierungssprache Kontaktplan (KOP)

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU**…
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie die Implementierungssprache „Kontaktplan (KOP)“.
4. Klicken Sie auf **Öffnen**.

⇨ TwinCAT fügt die POU zum SPS-Projektbaum hinzu und öffnet sie im Editor. Im Implementierungsteil ist ein leeres Netzwerk eingefügt. Das leere Netzwerk ist links durch eine vertikale Linie, die eine Stromschiene darstellt, begrenzt. Automatisch wird auch die Ansicht **Werkzeugkasten** geöffnet, in der die passenden Elemente, Operatoren und Funktionsbausteine für die KOP-Programmierung bereitgestellt werden.

Hinzufügen eines Kontakts und eines Funktionsbausteins (TON)

✓ Eine POU mit der Implementierungssprache KOP ist im Editor geöffnet und ein leeres Netzwerk ist eingefügt.

1. Klicken Sie in der Ansicht **Werkzeugkasten** auf die Kategorie **Kontaktplan Elemente**.
2. Klicken Sie auf das Element **Kontakt**, ziehen Sie es in Ihr Netzwerk und lassen Sie die Maus auf der Einfügeposition **Hier starten** los.
   ⇨ Der Kontakt wird links im Netzwerk direkt an der vertikalen Linie hinzugefügt.

3. Klicken Sie auf **???** und geben Sie den Bezeichner einer booleschen Variablen ein. Hierfür steht Ihnen auch die Eingabehilfe zur Verfügung.
4. Klicken Sie in der Ansicht **Werkzeugkasten** auf die Kategorie Funktionsbausteine und ziehen Sie den Funktionsbaustein TON auf eine Einfügeposition auf der Verbindungslinie rechts von dem eingefügten Kontakt.
   ⇨ TwinCAT fügt den Baustein TON rechts von dem Kontakt ein. Der Kontakt ist mit dem Eingang IN des TON-Bausteins verbunden.
5. Geben Sie am Eingang PT eine Zeitkonstante, beispielsweise T#3s ein.

⇨ Wenn die Variable Ihres Kontakts TRUE wird, wird auch der Eingang IN des TON-Bausteins TRUE. Mit einer Einschaltverzögerung von beispielsweise T#3s leitet der TON-Baustein den Wert TRUE an den Ausgang Q weiter.

Einfügen einer geschlossenen Leitungsverzweigung

✓ Eine POU mit der Implementierungssprache KOP ist im Editor geöffnet und ein leeres Netzwerk ist eingefügt.

1. Klicken Sie in das leere Netzwerk und wählen Sie im Menü **FUP/KOP/AWL** den Befehl **Kontakt einfügen**.
2. Selektieren Sie die Verbindungslinie links des Kontakts und wählen Sie im Menü **FUP/KOP/AWL** den Befehl **Verzweigung Startpunkt setzen**.
   ⇨ Der Startpunkt auf der Verbindungslinie wird mit einem roten Rechteck gekennzeichnet. Alle möglichen Endpunkte der Verzweigung kennzeichnet TwinCAT mit einem blauen Rechteck.
3. Klicken Sie auf ein blaues Rechteck, um den Endpunkt Ihrer geschlossenen Leitungsverzweigung festzulegen.

⇨ TwinCAT fügt die Leitungsverzweigung zwischen dem Start- und dem Endpunkt ein. Der Programmfluss wird beide Zweige bis zum Endpunkt durchlaufen.
Wenn Sie die Leitungsverzweigung nicht an einem Kontakt, sondern an einem Baustein einfügen, wird der Baustein nur aufgerufen, wenn keiner der anderen Zweige TRUE ist.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Kontakt einfügen [▶ 1077]
- Dokumentation TC3 User Interface: Befehl Verzweigung Startpunkt setzen [▶ 1082]

7.5.1.3 Programmieren in Anweisungsliste (AWL)

AWL kann bei Bedarf über die TwinCAT-Optionen aktiviert werden.
(Extras > Optionen > TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL > AWL)

Anlegen einer POU in der Implementierungssprache Anweisungsliste (AWL)

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU**…
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie die Implementierungssprache „Anweisungsliste (AWL)“.
4. Klicken Sie auf **Öffnen**.

⇨ TwinCAT fügt die POU zum SPS-Projektbaum hinzu und öffnet sie im Editor. Im Implementierungsteil ist bereits ein Netzwerk eingefügt.

Programmieren eines Netzwerks (beispielsweise einer ADD-Operation)

✓ Eine POU (AWL) ist im Editor geöffnet und besitzt ein leeres Netzwerk.

1. Klicken Sie in die 1. Spalte der farblich markierten Zeile und geben Sie den Operator LD ein.
2. Drücken Sie die Taste **[Tab]**.
   ⇨ Der Cursor springt in die 2. Spalte.

3. Geben Sie den ersten Summanden Ihrer ADD-Operation ein, zum Beispiel „6“.
4. Drücken Sie **[Strg] + [Eingabetaste]** oder wählen Sie im Menü **FUP/KOP/AWL** den Befehl **AWL-Zeile danach einfügen**.
   ⇨ TwinCAT fügt eine neue Anweisungszeile unterhalb ein. Der Fokus liegt in der ersten Spalte dieser Zeile.
5. Geben Sie ADD ein und drücken Sie **[Tab]**.
6. Geben den 2. Summanden Ihrer ADD-Operation ein, zum Beispiel „12“.
7. Drücken Sie **[Strg] + [Eingabetaste]**
8. Geben den Operator ST ein und drücken Sie **[Tab]**.
9. Geben Sie eine Variable vom Datentyp INT an, zum Beispiel „nVar“.

⇨ Das Ergebnis, im Beispiel „16“, wird in nVar gespeichert.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl AWL-Zeile danach einfügen [▶ 1079]

Aufrufen eines Funktionsbausteins

✓ Eine POU (AWL) ist im Editor geöffnet und besitzt ein leeres Netzwerk. Im Deklarationsteil ist eine Variable mit Datentyp <Funktionsbaustein> deklariert, beispielsweise `fbSample : CTU;`.

1. Klicken Sie in die erste Spalte der farblich markierten Zeile und wählen Sie im Menü **FUP/KOP/AWL** den Befehl **Bausteinaufruf einfügen**.
   ⇨ Die **Eingabehilfe** öffnet sich.
2. Wählen Sie in der Kategorie **Funktionsbausteine** oder in der Kategorie **Bausteinaufrufe** den gewünschten Funktionsbaustein aus, beispielsweise den Zähler CTU aus der Bibliothek Tc2_Standard, und klicken Sie auf **OK**.
3. TwinCAT fügt den ausgewählten Funktionsbaustein CTU wie folgt ein:

```
CAL         ???(
        CU:= ???,
     RESET:= ???,
        PV:= ???,
         Q=> ???,
        CV=> ???)
```

4. Ersetzen Sie die Zeichenfolgen **???** mit dem Variablennamen und den Werten oder Variablen für die Ein-/Ausgänge des Funktionsbausteins.
5. Alternativ zum Einfügen des Funktionsbausteins über die Eingabehilfe können Sie den Aufruf direkt im Editor eingeben.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Bausteinaufruf einfügen [▶ 1073]

7.5.2 Continuous Function Chart (CFC)

Die Implementierungssprache Continuous Function Chart (CFC) ist eine grafische Programmiersprache und erweitert die Standardsprachen der IEC 61131-3.

In einen Programmierbaustein in CFC können Sie Elemente einfügen und frei positionieren. Mit Verbindungen verschalten Sie die Elemente zu einem Netzwerk. Außerdem können Sie Rückkopplungen einfügen. Es entsteht ein übersichtlicher Funktionsplan, den Sie lesen können wie einen Schaltplan oder ein Blockschaltbild.

Die Ausführungsreihenfolge eines Funktionsplans ist datenflussbasierend. Ein Programmierbaustein kann mehrere Datenflüsse verarbeiten. Die Datenflüsse haben dann keine gemeinsamen Daten und im Editor befinden sich mehrere Netzwerke, die untereinander keine Verbindungen haben. Programmierbausteine in FUP, KOP oder AWL haben eine dagegen eine netzwerkbasierende Ausführungsreihenfolge.

Die Implementierungssprache Continuous Function Chart (CFC) - seitenorientiert ist ebenfalls eine grafische Programmiersprache und erweitert die Standardsprachen der IEC 61131-3.

Sie können in dieser Sprache raumeinnehmende, komplexe Funktionspläne grafisch programmieren. Dafür stehen Ihnen dieselben Elemente und Befehle zur Verfügung wie in Continuous Function Chart (CFC). Zusätzlich können Sie den Code auf beliebig vielen Seiten verteilt anordnen. Dadurch können Sie umfangreiche Funktionspläne erstellen, die trotzdem gut zu drucken sind. In den Randbereichen jeder Seite können Sie links Eingänge und Zielverbindungsmarken, rechts Ausgänge und Quellverbindungsmarken anordnen. Das unterstützt Sie beim Einfügen von Verbindungslinien und verhilft zu mehr Übersicht.

Es nicht möglich, einen Programmierbaustein zwischen den Implementierungssprache Continuous Function Chart (CFC) - seitenorientiert und Continuous Function Chart (CFC) umzuschalten.

Siehe auch:

- Referenz Programmierung: Continuous Function Chart (CFC) und Seitenorientierter CFC [▶ 700]
- Dokumentation TC3 User Interface: CFC [▶ 1057]

7.5.2.1 Programmieren in CFC

Sie können im CFC-Editor Programmierbausteine miteinander verdrahten und anschauliche Blockschaltbilder erstellen.

Der Editor unterstützt Sie folgendermaßen:

- Programmieren mit Elementen und Verbindungslinien
- Ziehen von Instanzen und Variablen in den Editierbereich
- Autorouting der Verbindungslinien
- Automatisches Verknüpfen
- Fixieren von Verbindungslinien durch Kontrollpunkte
- Kollisionserkennung
- Eingabeunterstützung bei Verbindungsmarken
- Forcen und Schreiben von Werten im Onlinebetrieb
- Verschieben der Selektion mittels Pfeiltasten
- Reduzierte Darstellung eines Bausteins ohne unverbundene Anschlüsse

Anlegen einer POU in der Implementierungssprache Continous Function Chart (CFC)

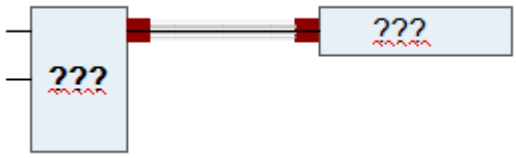
1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU...**.
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie die Implementierungssprache **Continuous Function Chart (CFC)**.
4. Klicken Sie auf **Öffnen**.

⇨ TwinCAT fügt die POU zum SPS-Projektbaum hinzu und öffnet sie im Editor.

Elemente einfügen und mit Verbindungslinien verschalten

1. Platzieren Sie ein Element **Baustein** und ein Element **Ausgang** im Editor. Ziehen Sie dazu die Elemente aus der Ansicht **Werkezugkasten** mit der Maus in den Editor.
2. Klicken Sie auf den Ausgang des Elements **Baustein**.
   ⇨ Der Ausgang wird mit einem roten Quadrat gekennzeichnet.
3. Ziehen Sie mit gedrückter Maustaste eine Verbindungslinie vom Ausgang des Elements **Baustein** zum Eingang des Elements **Ausgang**.
   ⇨ Beim Erreichen des Eingangspins ändert der Cursor sein Symbol.
4. Lassen Sie die Maustaste los.

⇨ Der Ausgangspin des Bausteins ist mit dem Eingangspin des Ausgangs verdrahtet.



Alternativ können Sie auch die beiden Pins mit gedrückter **[Strg]**-Taste selektieren und den Befehl **Selektierte Anschlüsse verbinden** im Menü **CFC** oder im Kontextmenü auswählen.

Instanzen aufrufen

1. Erstellen Sie ein neues Projekt und fügen Sie ein Standard-SPS-Projekt hinzu
2. Erstellen Sie den Funktionsbaustein FB_Sample in der Implementierungssprache ST mit Eingängen und Ausgängen:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nIn1 : INT;
    nIn2 : INT;
    sIn1 : STRING := 'Name';
    bIn1 : BOOL;
END_VAR
VAR_OUTPUT
    nOut : INT;
    sOut : STRING;
    bOut : BOOL;
END_VAR
VAR
END_VAR
```

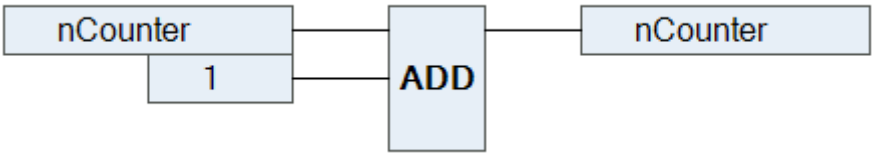
```
nOut := nIn1 + nIn2;

sResult := sIn1;

IF bIn1 THEN
bOut := TRUE;
END_IF
```

3. Erstellen Sie das Programm PRG_First in der Implementierungssprache ST .
4. Instanziieren Sie Funktionsbausteine und deklarieren Sie Variablen:

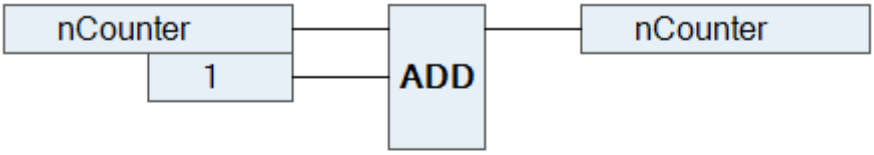
```
PROGRAM PRG_First
VAR
    nCounter  : INT;
    fbSample1 : FB_Sample;
    fbSample2 : FB_Sample;
    nResult   : INT;
    sResult   : STRING;
    bResult   : BOOL;
END_VAR
```

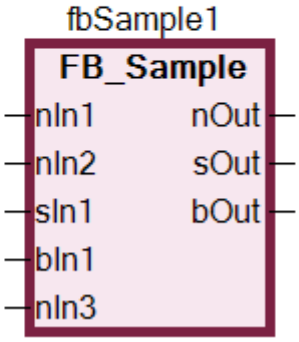
5. Ziehen Sie ein Element **Baustein** aus der Ansicht **Werkzeugkasten** in den Editor.
6. Klicken Sie auf das Feld **???** und geben Sie ADD ein.
   ⇨ Der Bausteintyp ist ADD. Der Baustein fungiert als Addierer.
7. Klicken Sie im Deklarationseditor auf die Zeilennummer 3.
   ⇨ Die Deklarationszeile von `nCounter` ist selektiert.
8. Klicken Sie in die Selektion und ziehen Sie die selektierte Variable in die Implementierung. Fokussieren Sie dort einen Eingang des Bausteins ADD.
   ⇨ Ein Eingang wurde erzeugt, deklariert und mit dem Baustein verbunden.
9. Klicken Sie nochmals in die Selektion und ziehen Sie die Variable zum Ausgang des Bausteins ADD.
   ⇨ Ein Ausgang wurde erzeugt, deklariert und mit dem Baustein verbunden.
10. Ziehen Sie aus der Ansicht **Werkzeugkasten** ein Element **Eingang** in die Implementierung.
11. Klicken Sie auf dessen Feld **???** und geben Sie 1 ein.
12. Verbinden Sie den Eingang 1 mit einem Eingang am Baustein ADD.

⇨ Ein Netzwerk ist programmiert. Zur Laufzeit zählt das Netzwerk die Zyklen und speichert das Ergebnis in `nCounter`.



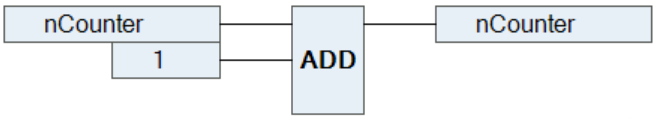
13. Klicken Sie im Deklarationseditor auf die Zeilennummer 4.
    ⇨ Die Zeile mit fbSample1 ist selektiert.
14. Klicken Sie in die Selektion und ziehen Sie die selektierte Instanz in die Implementierung.
    ⇨ Die Instanz erscheint als Baustein im Editor. Typ, Name und die Bausteinpins werden entsprechend angezeigt.

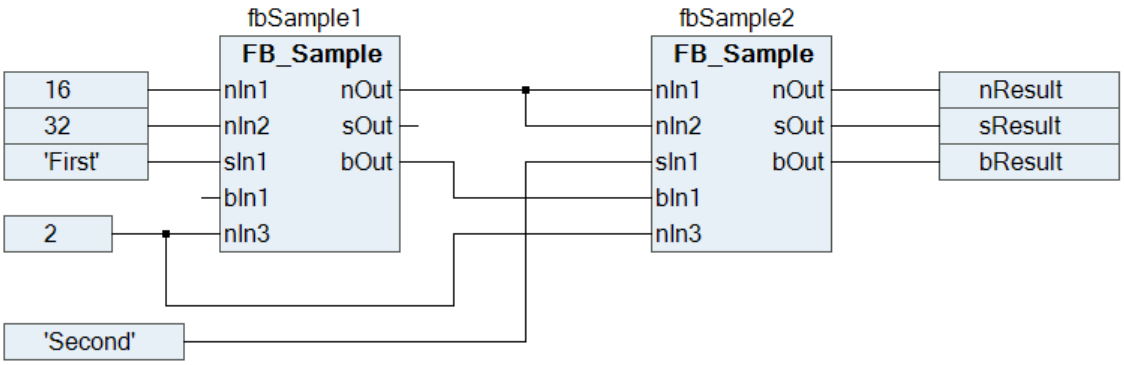




15. Ziehen Sie die Instanz fbSample2 in den Editor. Verschalten Sie die Instanzen untereinander und mit Eingängen und Ausgängen.

⇨ Beispiel:





Ein Programm in ST mit gleicher Funktionalität könnte folgendermaßen aussehen:

```
PROGRAM PRG_First_ST
VAR
    nCounter  : INT;
```

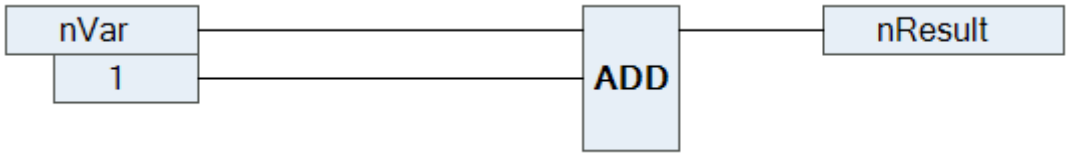
```
    fbSample1 : FB_Sample;
    fbSample2 : FB_Sample;
    nResult   : INT;
    sResult   : STRING;
    bResult   : BOOL;
END_VAR
```

```
nCounter := nCounter + 1;
fbSample1(nIn1 := 16, nIn2 := 32, sIn1 := 'First', xItem := TRUE, nIn3 := 2, nOut => fbSample2.nIn1,
bOut => fbSample2.bIn1);
fbSample2(nIn2 := fbSample1.nOut, sIn1 := 'Second', nIn3 := 2, nOut => nResult, sOut => sResult,
bOut => bResult);
```

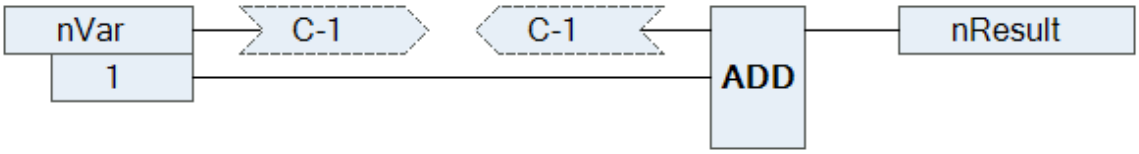
Verbindungsmarken erzeugen

✓ Sie haben einen CFC-Programmierbaustein mit verbundenen Elementen.

1. Selektieren Sie eine Verbindungslinie zwischen zwei Elementen.
   ⇨ Die Verbindungslinie wird markiert, der Eingang oder der Ausgang der Elemente wird mit einem roten Quadrat markiert .

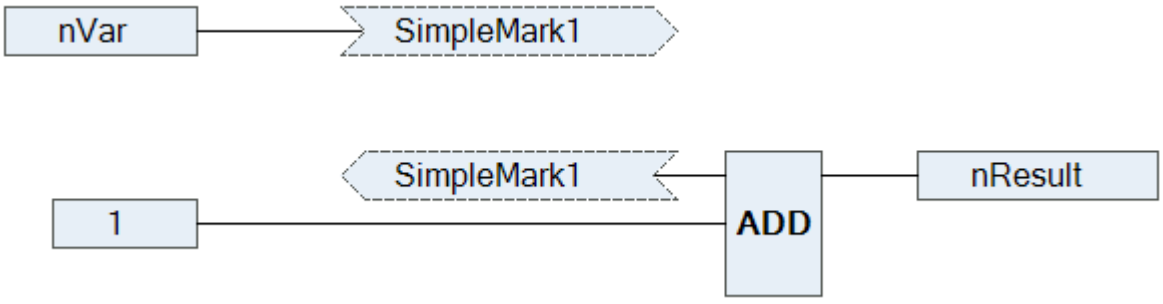


2. Wählen Sie im Kontextmenü oder im Menü **CFC** den Befehl **Verbindungsmarke**.
   ⇨ Die Verbindung wird aufgetrennt und durch eine **Verbindungsmarke - Quelle** und eine **Verbindungsmarke - Ziel** ersetzt. Der Name der Marken wird automatisch erzeugt.



3. Klicken Sie in die Verbindungsmarke-Quelle.
   ⇨ Der Name kann editiert werden.
4. Geben Sie einen Namen für die Quellverbindungsmarke ein.

⇨ Quellverbindungsmarke und Zielverbindungsmarke haben den gleichen Namen.



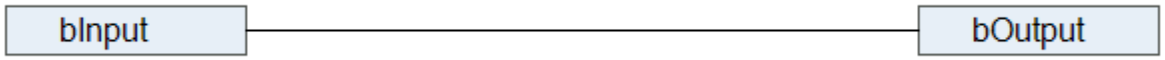
Siehe auch:

- Dokumentation TC3 User Interface: Befehl Verbindungsmarke [▶ 1069]

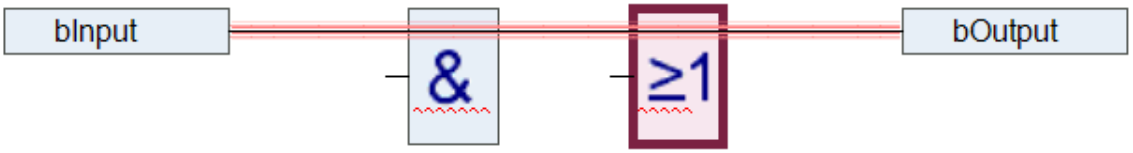
Kollisionen lösen und Verbindungslinien durch Kontrollpunkte fixieren

Das nachfolgende Beispiel zeigt Ihnen die Verwendung des Befehls **Alle Verbindungen routen**, sowie die Verwendung von Kontrollpunkten.

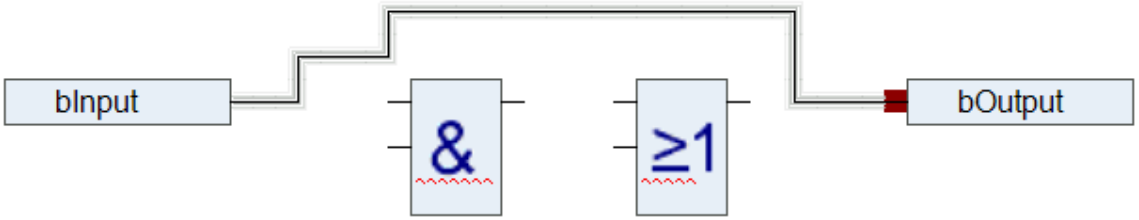
1. Platzieren Sie die Elemente **Eingang** und **Ausgang** und verbinden Sie die Elemente.



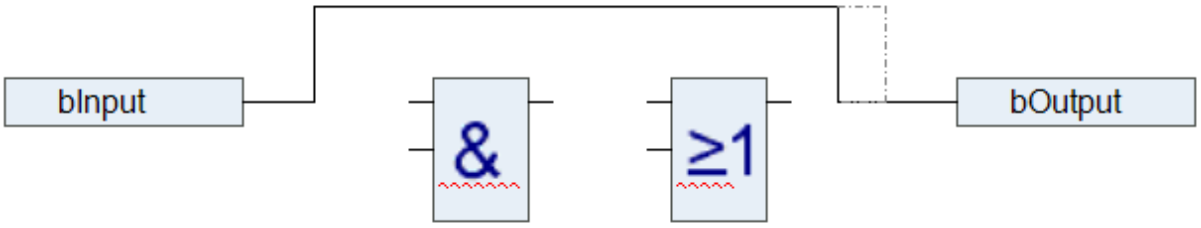
2. Platzieren Sie zwei Elemente **Baustein** auf der Linie.
   ⇨ Die Verbindungslinie und die Bausteine werden wegen der Kollision rot gezeichnet.



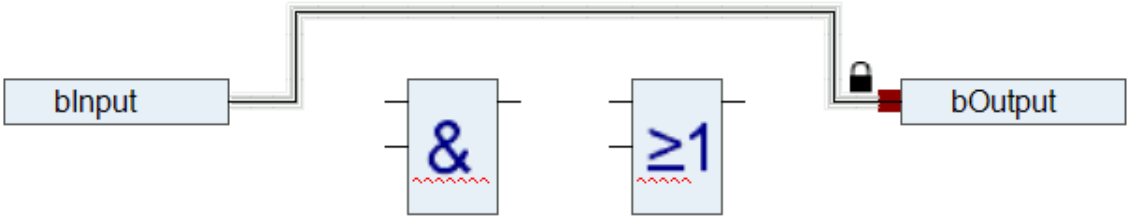
3. Wählen Sie im Menü **CFC > Routing** den Befehl **Alle Verbindungen routen**.
   ⇨ Die Kollision wird gelöst.



4. Verändern Sie schrittweise die Verbindungslinien.

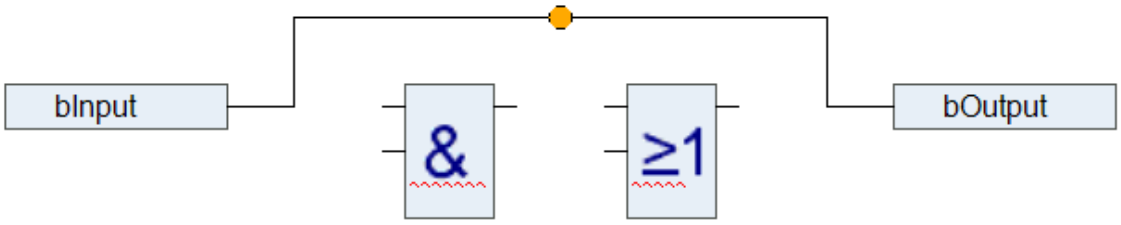


   ⇨ Die Verbindungslinie wurde manuell verändert und ist nun für das Autorouting gesperrt. Dies wird durch ein Schloss am Verbindungsende angezeigt.

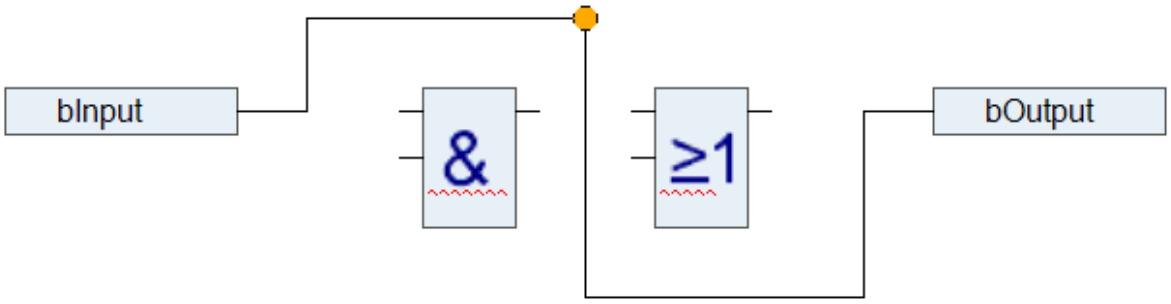


5. Selektieren Sie die Verbindungslinie und wählen Sie den Befehl **Kontrollpunkt erzeugen** im Menü **CFC > Routing**. Alternativ können Sie einen Kontrollpunkt aus der Ansicht **Werkzeugkasten** auf eine Linie ziehen.

⇨ Auf der Verbindungslinie wird ein Kontrollpunkt erzeugt. An dem Kontrollpunkt ist die Verbindungslinie fixiert.



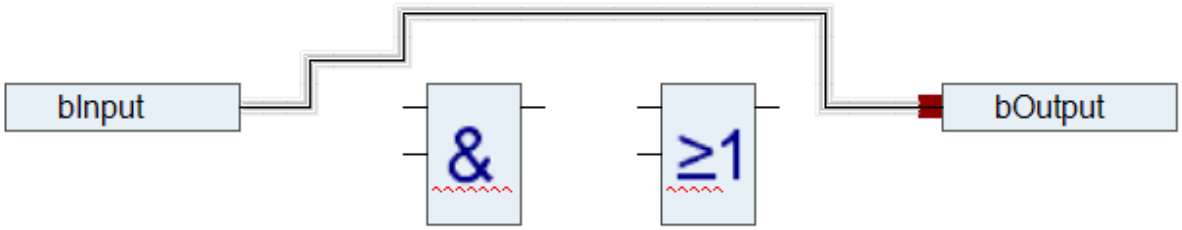
6. Verändern Sie die Verbindung gemäß nachfolgendem Beispiel.



⇨ Durch den Kontrollpunkt können Sie die Verbindungslinie individuell verändern. Sie können beliebig viele Kontrollpunkte setzen.

7. Entfernen Sie den Kontrollpunkt mit dem Befehl **Kontrollpunkt entfernen** im Menü **CFC > Routing**.
8. Lösen Sie die Verbindung mit dem Befehl **Verbindung lösen** oder durch einen Mausklick auf das Schloss-Symbol.
9. Selektieren Sie die Verbindungslinie und wählen Sie den Befehl **Alle Verbindungen routen**.

⇨ Die Verbindungslinie wird automatisch wie unter Schritt 3 dargestellt gezeichnet.



Verbindungen innerhalb einer Gruppe werden nicht automatisch geroutet.

Siehe auch:

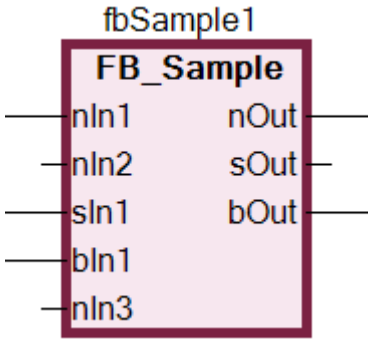
- Dokumentation TC3 User Interface: Befehl Alle Verbindungen routen [▶ 1067]
- Dokumentation TC3 User Interface: Befehl Kontrollpunkt entfernen [▶ 1068]
- Dokumentation TC3 User Interface: Befehl Verbindung lösen [▶ 1065]

Darstellung eines Bausteins reduzieren

Voraussetzung: Ein CFC-Programmierbaustein ist geöffnet. Im Editor werden dessen Bausteine mit allen deklarierten Anschlüssen angezeigt.

1. Selektieren Sie einen Baustein, dessen Anschlüsse teilweise unverbunden sind.

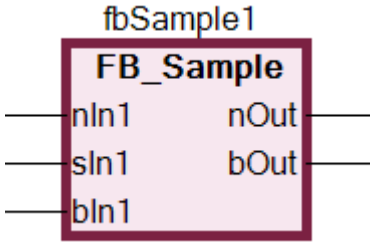
⇨ Beispiel: FB_Sample



⇨ Der Baustein benötigt Platz für alle Anschlüsse.

2. Wählen Sie im Menü **CFC > Routing** den Befehl **Nicht verbundene Anschlüsse entfernen**.

⇨ Der Baustein benötigt weniger Platz und wird nun nur mit den funktional relevanten Anschlüssen dargestellt.



Sehen Sie dazu auch

Befehl Eins vor [▶ 1062]

7.5.2.2 Automatische Ausführungsreihenfolge nach Datenfluss

Die Ausführungsreihenfolge in Programmierbausteinen ist in textbasierenden und netzwerkbasierenden Editoren eindeutig determiniert. Im CFC-Editor können Sie die Elemente frei positionieren, die Ausführungsreihenfolge ist zunächst nicht eindeutig. Sie wird deshalb von TwinCAT nach dem Datenfluss und bei mehreren Netzwerken nach der topologischen Position der Elemente festgelegt. Die oberen werden vor den unteren und die linken vor den rechten Elementen und Netzwerken einsortiert. Der Programmierbaustein wird dadurch zeit- und zyklusoptimiert abgearbeitet.

Sie können sich im Chart über die zeitliche Abfolge der Elemente informieren und kurzzeitig die Ausführungsreihenfolge einblenden lassen. Wenn Sie Netzwerke mit Rückkopplungen programmieren, können Sie ein Element in der Rückkopplungsschleife als Startpunkt festlegen.

Sie können außerdem die Abfolge der Abarbeitung in einem CFC-Objekt bei Bedarf explizit bearbeiten. Dafür ändern Sie den Wert der Eigenschaft **Explicit Execution Order**vom Standardwert False zu True. Im Expliziten Ausführungsreihenfolge-Modus haben Sie die Möglichkeit, die Ausführungsreihenfolge mit Menübefehlen zu bearbeiten.

Datenfluss

Die zeitliche Abfolge, welche Daten wann und wie in welchen Programmierobjekten gelesen oder beschrieben werden, wird allgemein als Datenfluss bezeichnet. Ein Programmierbaustein kann beliebig viele Datenflüsse verarbeiten, die auch unabhängig voneinander ausgeführt werden können.

Ausführungsreihenfolge anzeigen lassen

Im standardmäßig aktivierten automatischen Datenfluss-Modus wird die Ausführungsreihenfolge eines CFC-Objekts automatisch ermittelt. Sie können sich die Ausführungsreihenfolge im CFC-Editor kurzzeitig einblenden lassen.

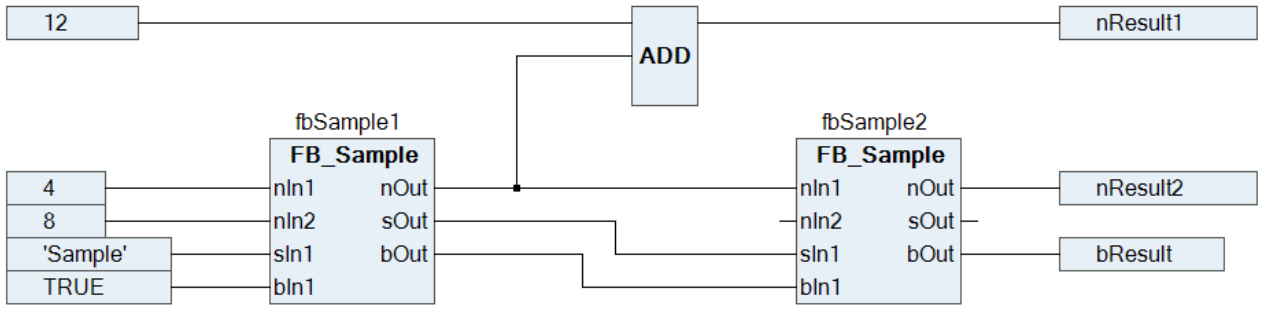
1. Erstellen Sie ein neues Projekt und fügen Sie ein Standard-SPS-Projekt hinzu.
2. Erstellen Sie den Funktionsbaustein `FB_Sample` in der Implementierungssprache ST mit Eingängen und Ausgängen.

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nIn1 : INT;
    nIn2 : INT;
    sIn1 : STRING := 'Name';
    bIn1 : BOOL;
END_VAR
VAR_OUTPUT
    nOut : INT;
    sOut : STRING;
    bOut : BOOL;
END_VAR
VAR
END_VAR
```

```
nOut := nIn1 + nIn2;

sResult := sIn1;

IF bIn1 THEN
bOut := TRUE;
END_IF
```

3. Erstellen Sie das Programm Execute_CFC in der Implementierungssprache CFC.

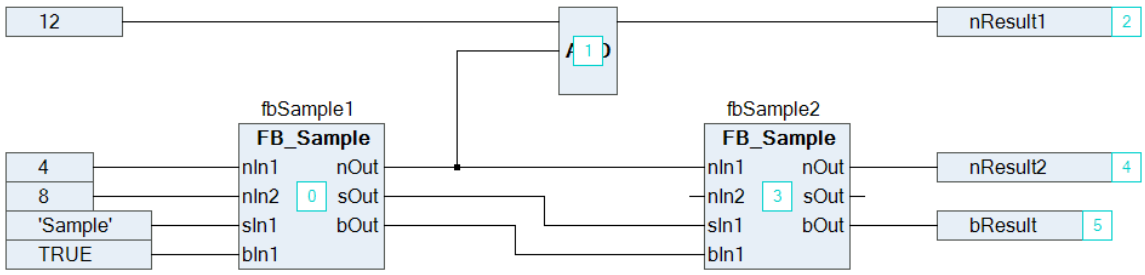
```
PROGRAM Execute_CFC
VAR
    fbSample1 : FB_Sample;
    fbSample2 : FB_Sample;
    nResult1  : INT;
    nResult2  : INT;
    bResult   : BOOL;
END_VAR
```



Neu erstellte Programmierobjekte in CFC haben den automatischen Datenfluss-Modus aktiviert. Die Ausführungsreihenfolge des Programmierobjekts wird intern optimal festgelegt.

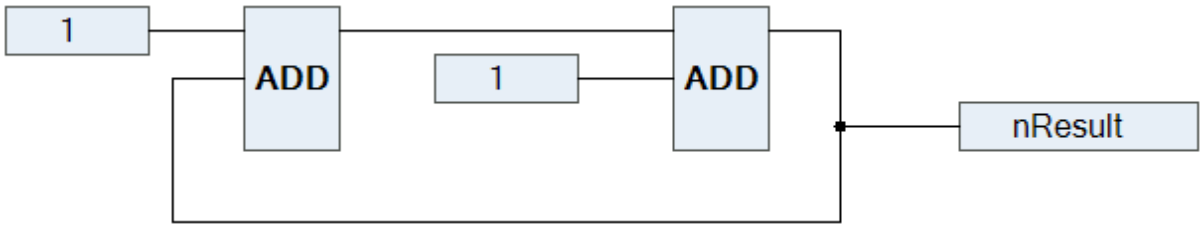
4. Wählen Sie den Befehl **Ausführungsreihenfolge > Ausführungsreihenfolge anzeigen** im Menü **CFC**.

⇨ Die Ausführungsreihenfolge des Objekts ist eingeblendet. Die Bausteine und Ausgänge sind dementsprechend nummeriert und geben die zeitliche Abfolge der Abarbeitung wieder. Sobald Sie erneut in den CFC-Editor klicken, wird die Nummerierung ausgeblendet.

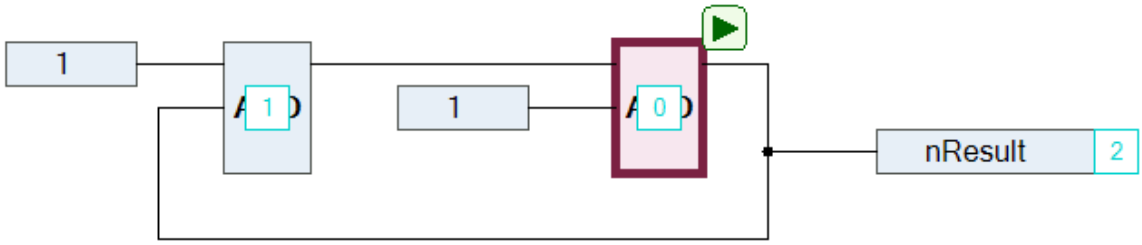


Ausführungsreihenfolge für Netzwerke mit Rückkopplungen festlegen

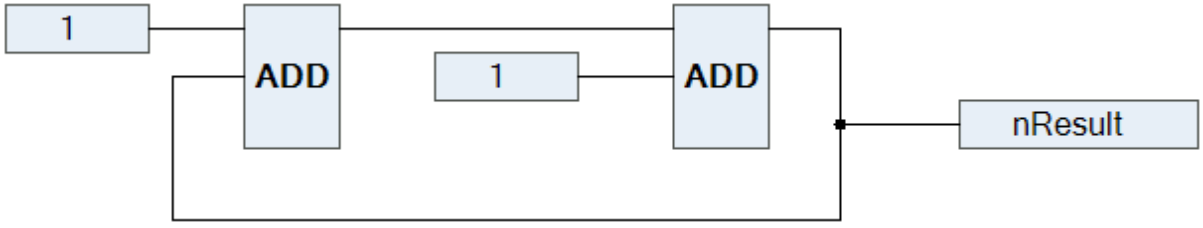
1. Erstellen Sie ein CFC-Programm, das eine Rückkopplung enthält.



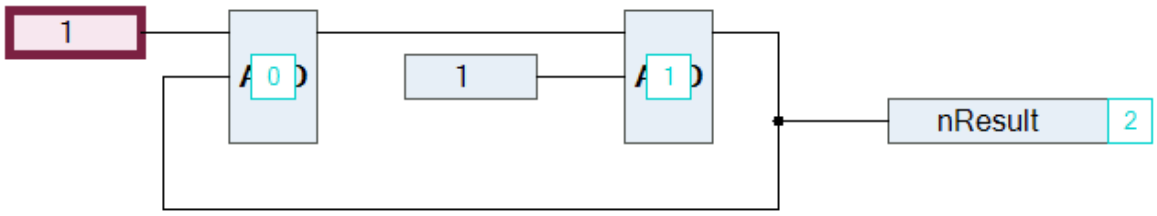
2. Selektieren Sie ein Element innerhalb der Rückkopplung.
   ⇨ Das selektierte Element ist rot hervorgehoben.
3. Wählen Sie den Befehl **Ausführungsreihenfolge > Startpunkt der Rückkopplung setzen** im Menü **CFC**.
   ⇨ Zur Laufzeit wird dieser Baustein als erster abgearbeitet. Der Startbaustein der Rückkopplung ist festgelegt und mit dem Symbol  dekoriert. Die Ausführungsreichenfolge ist neu sortiert und das selektierte Element erhält die 0 (allgemein die niedrigste Nummer in der Rückkopplung).



4. Selektieren Sie den Startbaustein nochmals.
5. Wählen Sie den Befehl **Ausführungsreihenfolge > Startpunkt der Rückkopplung setzen** im Menü **CFC**.
6. Der Baustein ist als Startbaustein abgewählt. Die Ausführungsreihenfolge ist intern festgelegt.



7. Wählen Sie den Befehl **Ausführungsreihenfolge > Ausführungsreihenfolge anzeigen** im Menü **CFC**.
   ⇨ Die Ausführungsreihenfolge nach Datenfluss wird angezeigt.



Ausführungsreihenfolge explizit festlegen

Die automatisch festgelegte Ausführungsreihenfolge nach Datenfluss bewirkt eine zeit- und zyklusoptimierte Ausführung des Programmierbausteins. Sie benötigen somit während des Entwicklungsprozesses keine Information über die intern verwaltete Ausführungsreihenfolge.

Im Expliziten Ausführungsreihenfolge-Modus passen Sie die Ausführungsreihenfolge in Eigenverantwortung an und müssen die Konsequenzen und Effekte selbst beurteilen. Auch deshalb wird die Ausführungsreihenfolge ständig eingeblendet.

Sie können die automatisch festgelegte Ausführungsreihenfolge eines CFC-Objekts explizit ändern, wenn Sie die Eigenschaft Explicit Execution Order aktiviert haben.

1. Selektieren Sie ein CFC-Objekt im Projektbaum.
2. Öffnen Sie das Kontextmenü und wählen Sie den Befehl **Eigenschaften**.
3. Setzen Sie die Option **Excplicit Execution Order** unter CFC auf True.
   ⇨ Der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Im CFC-Editor sind die Netzwerke nummeriert. Unter **Ausführungsreihenfolge** im Menü **CFC** stehen verschiedene Befehle für die Bearbeitung der Ausführungsreihenfolge zur Verfügung.
4. Öffnen Sie das CFC-Objekt.
5. Selektieren Sie ein nummeriertes Element und wählen Sie den Befehl **Ausführungsreihenfolge > An den Anfang**.
   ⇨ Die Ausführungsreihenfolge ist neu sortiert und das selektierte Element hat die Nummer 0.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Eins vor [▶ 1062]

7.5.3 Strukturierter Text (ST), Erweiterter Strukturierter Text (ExST)

Der ST-Editor dient dem Programmieren von POUs in der IEC-61131-3 Programmiersprache „Strukturierter Text (ST)“ und „Erweiterter Strukturierter Text“. Der „Erweitertete Strukturierte Text“ bietet bezüglich der Norm IEC 61131-3 einige zusätzliche Funktionen.

Der Strukturierte Text ist eine Programmiersprache, vergleichbar mit anderen Hochsprachen wie C oder PASCAL, die die Entwicklung komplexer Algorithmen erlaubt. Der Programmcode besteht aus einer Kombination von Ausdrücken und Anweisungen, die auch bedingt (IF...THEN...ELSE) oder in Schleifen (WHILE...DO) ausgeführt werden können.

Ein Ausdruck ist ein Konstrukt, das nach seiner Auswertung einen Wert zurück liefert. Ausdrücke sind auch Operatoren und Operanden zusammen. Auch Zuweisungen können Sie als Ausdruck verwenden. Ein Operand kann eine Konstante, eine Variable, ein Funktionsaufruf oder ein weiterer Ausdruck sein.

Anweisungen steuern, wie die Ausdrücke verarbeitet werden sollen.

Für diesen Texteditor können Sie in den Dialogen **Optionen** und **Anpassen** des Menüs **Extras** verschiedene Einstellungen bezüglich Verhalten, Aussehen und Menüs vornehmen. Für diesen Editor stehen auch die bekannten Windows-Funktionen zur Verfügung (zum Beispiel IntelliMouse).

ExST - Erweiterter Strukturierter Text

„Erweiterter Strukturierter Text (ExST)“ ist eine TwinCAT-spezifische Erweiterung bezüglich des IEC 61131-3 Standards für „Strukturierten Text (ST)“.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Anpassen [▶ 1041]
- Dokumentation TC3 User Interface: Befehl Optionen [▶ 1010]
- Referenz Programmierung: Strukturierter Text und Erweiterter Strukturierter Text (ExST) [▶ 656]

7.5.3.1 Programmieren in Strukturiertem Text (ST)

Prinzip

Sie programmieren die Programmiersprachen Strukturierter Text und Erweiterter Strukturierter Text im ST-Editor. Der Programmcode besteht aus einer Kombination von Ausdrücken und Anweisungen, die auch bedingt oder in Schleifen ausgeführt werden können. Jede Anweisung müssen Sie mit einem ; abschließen.

Die Deklaration der Variablen erfolgt im Deklarationseditor.

Siehe auch:

- Deklarationseditor verwenden [▶ 72]

Anlegen einer POU in der Implementierungssprache Strukturierter Text (ST)

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projekt einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU...**
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie die Implementierungssprache „Strukturierter Text (ST)“.
4. Klicken Sie auf **Öffnen**.

⇨ Die POU wird im SPS-Projektbaum hinzugefügt und im Editor geöffnet. Im oberen Teil der POU fügen Sie nun die Variablendeklarationen ein, im unteren Teil der POU geben Sie den ST-Programmcode ein.

7.5.4 Ablaufsprache (AS)

Mit dem AS-Editor programmieren Sie POUs in der IEC 61131-3 Programmiersprache Ablaufsprache. Die Ablaufsprache (AS) ist eine grafisch orientierte Sprache, die es erlaubt, die chronologische Abfolge einzelner Aktionen in einem Programm zu beschreiben. Zu diesem Zweck werden die Aktionen, die eigenständige Programmierobjekte sind, den Schritt-Elementen zugeordnet. Der Ablauf der Abarbeitung der Schritte wird durch Transitions-Elemente gesteuert.

Siehe auch:

- Referenz Programmierung: Ablaufsprache (AS) [▶ 667]
- Dokumentation TC3 User Interface: AS [▶ 1047]

7.5.4.1 Programmieren in Ablaufsprache (AS)

Programmierbaustein in der Implementierungssprache AS anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projekt einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU**…
   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie die Implementierungssprache „Ablaufsprache (AS)“.
4. Klicken Sie auf **Öffnen**.

⇨ TwinCAT fügt den Programmierbaustein zum SPS-Projektbaum hinzu und öffnet ihn im Editor.

Schritt-Transition hinzufügen

1. Selektieren Sie die Transition nach dem Schritt Init.
   ⇨ Die Transition ist rot markiert.
2. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Schritt-Transition danach einfügen**.
   ⇨ TwinCAT fügt Schritt Step0 und Transition Trans0 ein.
3. Selektieren Sie die Transition Trans0 und wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Schritt-Transition einfügen**.

⇨ TwinCAT fügt Transition Trans1 und Schritt Step1 vor Trans0 ein.

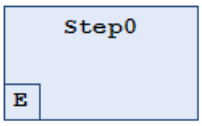
Sie können die Elemente **Schritt** und **Transition** auch mit Drag-and-drop aus der Ansicht **Werkzeugkasten** in das Diagramm ziehen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Schritt-Transition danach einfügen [▶ 1048]
- Dokumentation TC3 User Interface: Befehl Schritt-Transition einfügen [▶ 1047]

Eingangsaktion hinzufügen

1. Selektieren Sie den Schritt Step0.
2. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Eingangsaktion hinzufügen**.
   ⇨ Im Standardfall erhalten Sie eine Eingabeaufforderung zur Festlegung des Duplizierungsmodus für die Schrittaktionen. Sie entscheiden damit, ob beim künftigen Kopieren des Schritts die Referenzinformation zu den bestehenden Schrittaktionsobjekten kopiert wird, oder ob die Objekte „eingebettet“ werden sollen. Das Einbetten hat zur Folge, dass beim Kopieren des Schritts neue Schrittaktionsobjekte angelegt werden. Der Duplizierungsmodus ist in der Schritteigenschaft **Duplizieren oder Kopieren** definiert. Solange diese Eigenschaft deaktiviert ist, rufen die kopierten Schritte dieselben Aktionen auf wie der aktuelle Schritt.
3. Belassen Sie für dieses Beispiel die Standardeinstellungen **Referenz kopieren** und bestätigen sie mit **OK**.
   ⇨ Der Dialog **Eingangsaktion hinzufügen** öffnet sich.
4. Geben Sie „Step0_entry“ als Namen ein und wählen Sie die Implementierungssprache „Strukturierter Text (ST)“. Klicken Sie auf **Hinzufügen**.
   ⇨ TwinCAT fügt die Aktion Step0_entry unterhalb des Bausteins im SPS-Projektbaum ein und öffnet die Aktion im Editor. In der Eingangsaktion Step0_entry programmieren Sie nun Anweisungen, die einmalig bei Aktivierung des Schritts Step0 ausgeführt werden sollen.
5. Schließen Sie den Editor von Step0_entry.
   ⇨ Der Schritt Step0 ist nun mit einem E in der linken unteren Ecke gekennzeichnet. Mit einem Doppelklick auf diese Markierung öffnen Sie den Editor.

   ⇨ Die Eingangsaktion Step0_entry steht nun in den Eigenschaften des Schritts unter **Eingangsaktion**. Dort können Sie bei Bedarf auch eine andere Aktion auswählen.
6. Selektieren Sie den Schritt Step0. Drücken **[Strg] + [C]**, um den Schritt zu kopieren.

⇨ In der eingefügten Kopie des Schritts findet sich dieselbe, oben eingefügte Eingangsaktion wieder. Der neue Schritt ruft also genau dieselbe Aktion auf.

AS-Editor-Optionen

In den TwinCAT-Optionen in der Kategorie **AS-Editor** können Sie einstellen, ob die Eingabeaufforderung zur Festlegung des Duplizierungsmodus beim Einfügen einer Schrittaktion immer erscheinen soll oder standardmäßig einen Duplizierungsmodus festlegen.

Siehe auch:

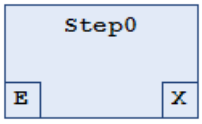
- Referenz Programmierung: AS-Elementeigenschaften [▶ 683]

- Dokumentation TC3 User Interface: Befehl Eingangsaktion hinzufügen [▶ 1054]
- Dokumentation TC3 User Interface: Dialog Optionen - AS-Editor [▶ 1022]

Ausgangsaktion hinzufügen

1. Selektieren Sie den Schritt Step0.
2. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Ausgangsaktion hinzufügen**.
   ⇨ Im Standardfall erhalten Sie eine Eingabeaufforderung zur Festlegung des Duplizierungsmodus für die Schrittaktionen des Schritts. Sehen Sie dazu die Hinweise in den Abschnitten „Hinzufügen einer Eingangsoption“ und „AS-Elementeigenschaften“.
   ⇨ Danach öffnet sich der Dialog **Ausgangsaktion hinzufügen**.
3. Geben Sie „Step0_exit“ als Namen ein und wählen Sie die Implementierungssprache „Strukturierter Text (ST)“. Klicken Sie auf **Hinzufügen**.
   ⇨ TwinCAT fügt die Aktion Step0_exit unterhalb des Bausteins im SPS-Projektbaum ein und öffnet die Aktion im Editor. In der Ausgangsaktion Step0_exit programmieren Sie nun Anweisungen, die einmalig vor dem Deaktivieren des Schritts Step0 ausgeführt werden sollen.
4. Schließen Sie den Editor von Step0_exit.

⇨ Der Schritt Step0 ist nun mit einem X in der rechten unteren Ecke gekennzeichnet. Mit einem Doppelklick auf diese Markierung öffnen Sie den Editor.



⇨ Sie können die Ausgangsaktion in den Eigenschaften des Schritts unter Ausgangsaktion definieren. Dort können Sie bei Bedarf auch eine andere Aktion auswählen.

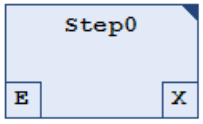
Siehe auch:

- Dokumentation TC3 User Interface: Befehl Ausgangsaktion hinzufügen [▶ 1055]

Aktion hinzufügen

1. Doppelklicken Sie auf den Schritt Step0.
   ⇨ Im Standardfall erhalten Sie eine Eingabeaufforderung zur Festlegung den Duplizierungsmodus für die Schrittaktionen des Schritts. Sehen Sie dazu die Hinweise in den Abschnitten „Hinzufügen einer Eingangsoption“ und „AS-Elementeigenschaften“. Der Dialog **Aktion hinzufügen** öffnet sich.
2. Geben Sie „Step0_active“ als Namen ein und wählen Sie die Implementierungssprache „Strukturierter Text (ST)“. Klicken Sie auf **Hinzufügen**.
   ⇨ TwinCAT fügt die Aktion Step0_active unterhalb des Bausteins im SPS-Projektbaum ein und öffnet die Aktion im Editor. In der Schrittaktion Step0_active programmieren Sie nun Anweisungen, die ausgeführt werden sollen, solange der Schritt aktiv ist.
3. Schließen Sie den Editor von Step0_active.

⇨ Der Schritt Step0 ist nun mit einem schwarzen Dreieck in der rechten oberen Ecke gekennzeichnet.



Sie können die Aktion in den Eigenschaften des Schritts unter Schrittaktion definieren. Dort können Sie auch eine andere Aktion auswählen.

Alternative Verzweigung hinzufügen

1. Selektieren Sie den Schritt Step1.
2. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Verzweigung rechts einfügen**.
   ⇨ TwinCAT fügt den Schritt Step2 rechts von Step1 ein. Die Schritte sind als parallele Verzweigung mit einer Doppellinie verbunden.
3. Selektieren Sie eine der beiden Doppellinien.

⇨ Die Doppellinie wird rot gekennzeichnet.

4. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Alternativ**.

⇨ TwinCAT wandelt die Verzweigung in eine alternative Verzweigung um. Die Doppellinie ändert sich zu einer einfachen Linie.

Sie können eine alternative Verzweigung durch den Befehl **Parallel** in eine parallele Verzweigung umwandeln.

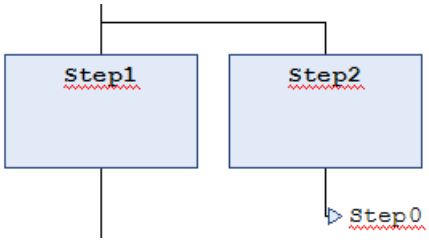
Siehe auch:

- Dokumentation TC3 User Interface: Befehl Verzweigung rechts einfügen [▶ 1049]
- Dokumentation TC3 User Interface: Befehl Alternativ [▶ 1049]

Sprung hinzufügen

1. Selektieren Sie den Schritt Step2.
2. Wählen Sie im Menü **AS** den Befehl **Sprung danach einfügen**.
   ⇨ TwinCAT fügt nach dem Schritt Step2 den Sprung Step ein.
3. Klicken Sie auf das Sprungziel **Step** des Sprungs.

⇨ Sie können nun das Sprungziel manuell eingeben oder über die Eingabehilfe [...] auswählen. Wählen Sie Step0.



Siehe auch:

- Dokumentation TC3 User Interface: Befehl Sprung danach einfügen [▶ 1052]

Makro hinzufügen

1. Selektieren Sie den Schritt Step1.
2. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Makro danach einfügen**.
   ⇨ TwinCAT fügt nach dem Schritt Step1 das Makro Macro0 ein.
3. Doppelklicken Sie auf das Element Macro0.
   ⇨ Das Makro öffnet sich im Implementierungsteil des Editors. In der Kopfzeile steht der Name Macro0.
4. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Schritt-Transition einfügen**.
   ⇨ TwinCAT fügt eine Schritt-Transition-Kombination ein.
5. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Makro verlassen**.

⇨ Der Implementierungsteil zeigt wieder das Hauptdiagramm.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Makro danach einfügen [▶ 1052]
- Dokumentation TC3 User Interface: Befehl Schritt-Transition einfügen [▶ 1047]
- Dokumentation TC3 User Interface: Befehl Makro verlassen [▶ 1053]

Assoziation hinzufügen

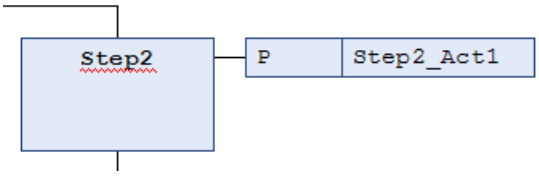
1. Selektieren Sie den Schritt Step2.
2. Wählen Sie im Menü **AS** oder im Kontextmenü den Befehl **Aktionsassoziation einfügen**.
   ⇨ TwinCAT fügt rechts neben dem Schritt Step2 eine Assoziation hinzu.

3. Klicken Sie in das linke Feld der Assoziation zur Auswahl des Qualifizierers.

   ⇨ Sie können den Qualifizierer manuell eingeben oder über die Eingabehilfe [...] auswählen. Wählen Sie „P“.

4. Klicken Sie in das rechte Feld der Assoziation zur Auswahl der Aktion.

   ⇨ Sie können die Aktion manuell eingeben oder über die Eingabehilfe [...] auswählen.



Siehe auch:

- Dokumentation TC3 User Interface: Befehl Aktionsassoziation einfügen [▶ 1051]

Ausdrücke analysieren mit AnalyzeExpression

Der Baustein AnalyzeExpression aus der Bibliothek Tc2_System erlaubt die Analyse von Ausdrücken. Er kann im AS-Diagramm beispielsweise verwendet werden, um das Ergebnis des Flags SFCError zu untersuchen, das der Überwachung von Zeitüberschreitungen im Ablaufdiagramm dient.

Siehe auch:

- Dokumentation Tc2_System: AnalyzeExpression
- Referenz Programmierung: AS-Flags [▶ 672]

7.5.5 Ladder Diagram (LD) (Beta)

Der neue Ladder-Editor ist in einer Beta-Version verfügbar ab TC3.1 Build 4026.

Der Ladder-Editor ist ein netzwerkbasierter Editor für die IEC-Programmiersprache Kontaktplan (KOP). Er ist der Nachfolger des weiterhin verfügbaren FUP/KOP-Editors. Der wesentliche Unterschied zum FUP/KOP-Editor besteht in den vereinfachten Editiermöglichkeiten. Varianten von Elementen, für die bisher separate Elemente aus der Werkzeugbox eingefügt werden mussten, entstehen nun rein durch die gewählte Einfügeposition oder durch das nachträgliche Umschalten eines Modifizierers. Programmierbausteine, die im FUP/KOP-Editor erstellt wurden, können über einen Menübefehl in das Ladder-Format konvertiert werden.

Siehe auch:

- Kapitel Referenz Programmierung: Ladder Diagram (LD) (Beta) [▶ 715]
- Dokumentation Benutzeroberfläche: Ladder-Editor [▶ 1085]

7.5.5.1 Programmieren im Ladder-Editor

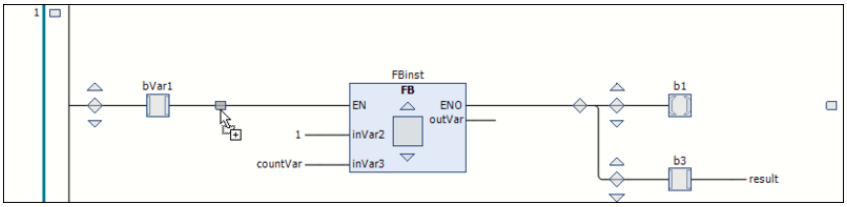
Die Voraussetzung für das Programmieren im Ladder:
Sie haben in Ihrem Projekt einen Programmierbaustein in der Implementierungssprache Ladder angelegt.
Sie haben diesen Baustein zur Bearbeitung im Ladder-Editor geöffnet.

Die folgenden Anleitungen beschreiben nur einzelne Aktionen beim Arbeiten im Editor. Es wird kein konkretes Programm erstellt. Bei Bedarf sehen Sie auch in folgendem Kapitel nach: Navigation im Ladder-Editor [▶ 718]

Elemente in einem Netzwerk einfügen, umpositionieren und modifizieren

Elemente in Netzwerke einfügen:

1. Selektieren Sie in der Werkzeugbox (Fenster Werkzeuge) des Editors beispielsweise das Element **Kontakt**.
2. Ziehen Sie es mit der Maus in das Netzwerk und halten Sie die Maustaste gedrückt.
   ⇨ Wenn Sie über das Netzwerk ziehen, werden mögliche Einfügepositionen durch folgende Symbole angezeigt:
   Grau hinterlegtes Quadrat innerhalb eines bereits vorhandenen Elementsymbols
   Raute auf einer Verbindungslinie
   Dreieck, nach unten oder oben gerichtet, für ein Einfügen oberhalb oder unterhalb
3. Sie können das Kontaktelement dann einfügen, wenn der Mauszeiger ein Plussymbol erhält.



4. Lassen Sie an einer solchen Position die Maustaste los.

⇨ Der Kontakt wird in der Abarbeitungslinie des Netzwerks eingefügt.

Bausteine einfügen:

1. Fügen Sie im Netzwerk ein Element **Baustein** ein.
2. Ersetzen Sie die drei Fragezeichen **???** in der Elementbox durch den Namen des gewünschten Bausteins aus dem Projekt oder einer Bibliothek.
3. Doppelklicken Sie dazu auf die Fragezeichen und verwenden Sie idealerweise die Eingabehilfe über ... .

⇨ Der Baustein wird eingefügt und erhält den passenden Namen.

Bausteine aktualisieren:

Wenn Sie beispielsweise einen Funktionsbaustein aus Ihrem Projekt im Ladder aufrufen, müssen Sie nach einer Änderung am Basis-FB auch im Ladder-Diagramm aktualisieren.

Ein entsprechender "Update"-Befehl ist derzeit noch nicht verfügbar.

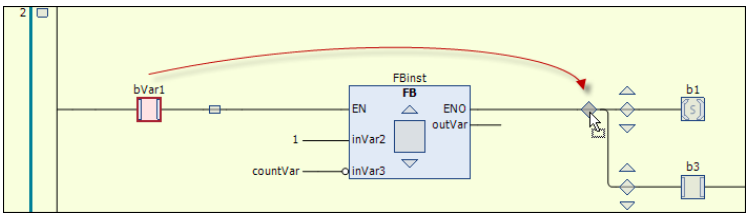
Eingänge einfügen:

1. Fügen Sie einen **Eingang** vor dem Baustein ein.
2. Ersetzen Sie zum Belegen mit Variablen jeweils die drei Fragezeichen **???** am Element durch den Variablennamen.

⇨ Bei Bedarf öffnet sich der Standarddialog **Variable deklarieren, Selektieren, Umpositionieren**.

Elemente im Netzwerk umpositionieren:

1. Selektieren Sie ein Element, so dass es rot hinterlegt erscheint. Im Falle eines Bausteins muss das Quadrat im Inneren des Bausteinsymbols rot hinterlegt erscheinen.
2. Ziehen Sie das Element mit der Maus an eine andere mögliche Einfügeposition, bis das Cursorsymbol ein Rechteck erhält, und lassen Sie die Maustaste los.



⇨ Das Element wird entsprechend positioniert.

Elemente ersetzen:

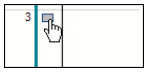
1. Um ein bestehendes Element durch ein anderes zu ersetzen, ziehen Sie das neue Element auf das zu ersetzende.
2. Lassen Sie die Maustaste los, wenn das zu ersetzende Element selbst als Einfügeposition grün aufleuchtet.

⇨ Das Element wird ersetzt.

Neues Netzwerk hinzufügen:

1. Ziehen Sie das Element Netzwerk in den Implementierungsteil.

⇨ Sie erhalten rechteckige Einfügemarken im linken Randbereich, in dem die Netzwerknummerierungen angezeigt werden. Das neue Netzwerk wird oberhalb der gewählten Einfügemarke eingefügt.

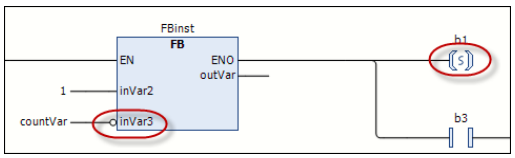


Netzwerke modifizieren:

Versehen Sie Elemente im Netzwerk mit Modifizierern:

1. Selektieren Sie den Eingangspin eines Bausteins.
2. Öffnen Sie das Kontextmenü mit der rechten Maustaste.
3. Wählen Sie den Befehl Negieren, [▶ 1085] um einen Eingang zu negieren, oder einen der Set/Reset [▶ 1087]- oder Flankenerkennungsbefehle (Befehl: Flankenerkennung: Steigende Flanke [▶ 1087], Befehl: Flankenerkennung: Fallende Flanke [▶ 1087]).

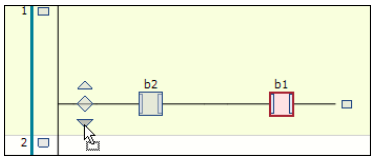
⇨ Der Eingang wird mit dem entsprechenden Symbol versehen.
Beispiel: Negierter Eingang am Baustein, Set-Spule



Leitungsverzweigungen erstellen und bearbeiten

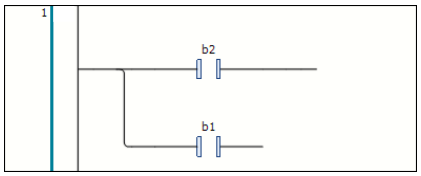
Im Unterschied zum LD-FBD-Editor gibt es im Ladder-Editor keine Elemente für Leitungsverzweigungen. Sie erzeugen und bearbeiten Verzweigungen rein durch das (Um)Positionieren von Elementen. Beispiele:

1. Erstellen Sie folgendes Netzwerk:



2. Ziehen Sie den zweiten Kontakt mit der Maus an die mit einem nach unten zeigenden Pfeil markierte Einfügeposition vor dem ersten Kontakt.

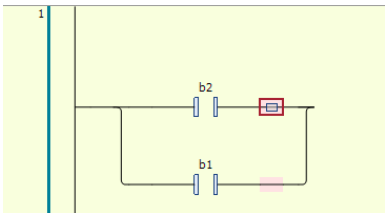
⇨ Eine parallele, nach hinten offene Verzweigung entsteht. Jeder Zweig enthält einen Kontakt.



Um aus der offenen Leitungsverzweigung eine geschlossene zu erzeugen, also ein OR-Konstrukt zu programmieren, gehen Sie folgendermaßen vor:

1. Selektieren Sie beide Zweige hinter dem Kontaktelement (Mehrfachselektion).
2. Die Selektion wird durch das rot hinterlegte kleine Rechteck auf der Leitung angezeigt.
3. Wählen Sie dann den Befehl Parallele Verzweigung schließen [▶ 1086].

⇨ Aus den beiden hinten offenen parallelen Zweigen wird eine geschlossene Verzweigung.



Um die geschlossene Leitungsverzweigung wieder zu öffnen, gibt es zwei Möglichkeiten:

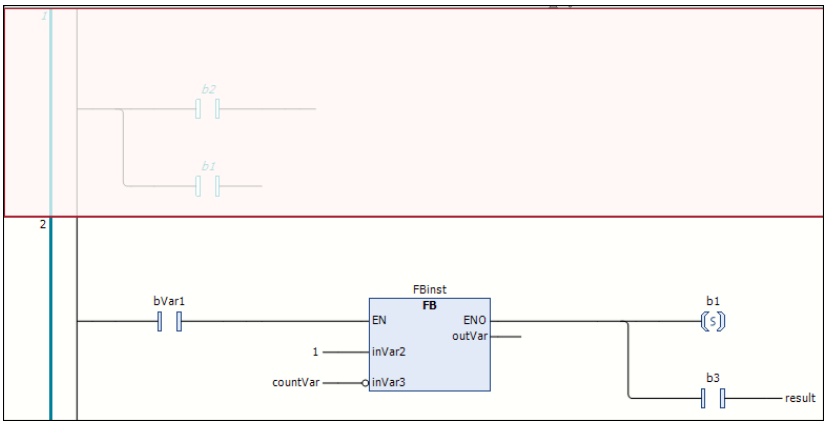
- Sie selektieren einen der beiden Zweige und ziehen das Selektionsrechteck auf dasjenige des anderen Zweigs.
- Sie selektieren beide Zweige und wählen den Befehl Parallele Verzweigung öffnen [▶ 1086].

Wenn Sie geschlossene Leitungsverzweigungen mehrerer paralleler Zweige haben, werden mit dem Befehl **Parallele Verzweigung öffnen** immer alle Zweige geöffnet.

Netzwerk auskommentieren

1. Selektieren Sie ein Netzwerk, so dass es komplett rot hinterlegt erscheint.
2. Wählen Sie den Befehl Auskommentiert [▶ 1085] aus dem Kontextmenü.

⇨ Der Netzwerkinhalt erscheint blass und die Texte in Kursivschrift. Das Netzwerk wird bei der Abarbeitung nicht berücksichtigt.

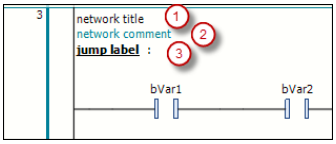


Netzwerktitel, Netzwerkkommentar, Sprungmarke hinzufügen

Sie können einem Netzwerk einen Netzwerktitel (1), einen Netzwerkkommentar (2) und/oder eine Sprungmarke (3) hinzufügen.

1. Klicken Sie dazu in die erste, zweite oder dritte Zeile in der linken oberen Ecke des Netzwerks. Eine Sprungmarke dient als Ziel beim Einfügen eines Sprung [▶ 1089]-Elements.

⇨ Die Anzeige von Netzwerktitel und Netzwerkkommentar wird in den TwinCAT-Optionen, Kategorie Ladder [▶ 1028] festgelegt.



Sehen Sie dazu auch

- Befehl Auskommmentiert [▶ 1085]
- Befehl Sprung einfügen [▶ 1089]
- Dialog Optionen - Ladder-Editor [▶ 1028]

7.6 Taskreferenz erzeugen

In ein SPS-Projekt können Sie ein oder mehrere Tasks einbinden und so die Abarbeitung von Programmbausteinen definieren. Um einen Task verwenden zu können, müssen Sie eine Taskreferenz erzeugen.

7.6.1 Objekt Taskreferenz

Symbol: 

Über die Taskreferenz definieren Sie die in einem Task auszuführenden Programmbausteine.

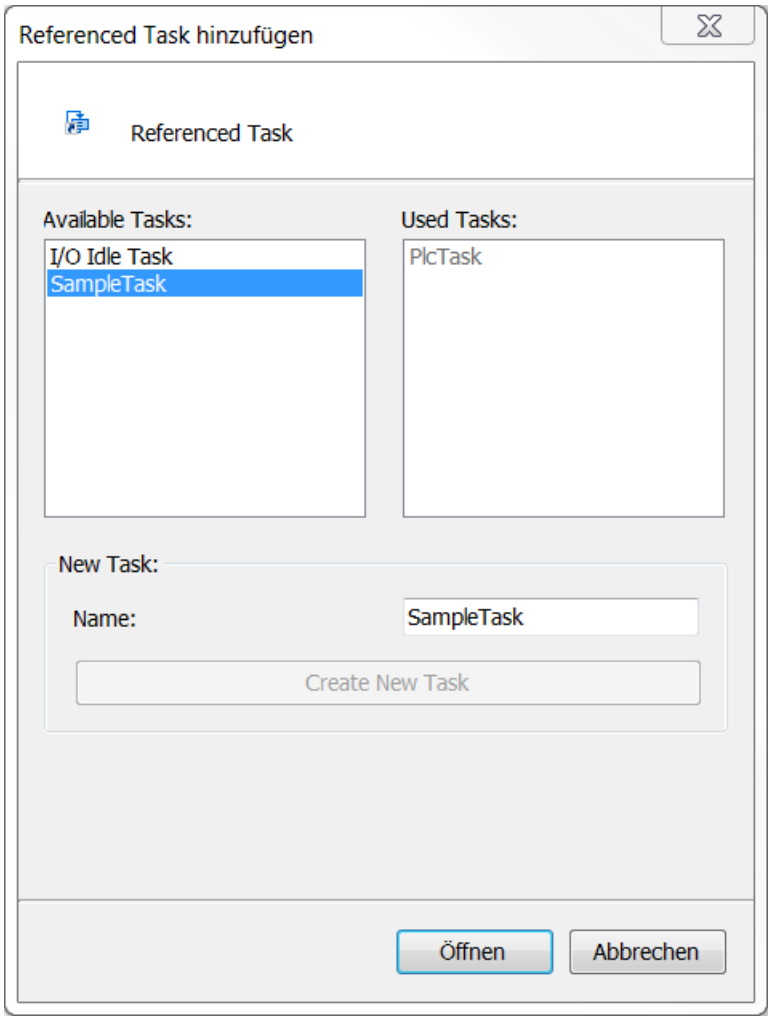
Beim Anlegen eines Standard-SPS-Projekts wird automatisch eine Taskreferenz **PlcTask** angelegt, welche die Abarbeitung des Programmbausteins MAIN definiert.

Objekt Taskreferenz anlegen

1. Selektieren Sie im Projektmappen-Explorer das SPS-Projekt (<Projektname> Project) im SPS-Projektbaum.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Taskreferenz**.
   ⇨ Der Dialog **Referenced Task hinzufügen** öffnet sich. Im Listenfeld **Available Tasks** werden die vorhandenen Tasks angezeigt. Im Listenfeld **Used Tasks** werden die bereits referenzierten Tasks angezeigt.
3. Wenn Sie einen neuen Task erzeugen und einbinden wollen, geben Sie im Textfeld **Name** einen Namen ein und klicken Sie auf **Create New Task**.
   ⇨ Der neue Task wird im Listenfeld **Available Tasks** angezeigt. Gleichzeitig erscheint der neue Task im TwinCAT-Projektbaum im Bereich **SYSTEM** unterhalb des Knotens **Tasks**.
4. Klicken Sie auf **Öffnen**.
5. Wenn der Task, den Sie einbinden wollen, bereits vorhanden ist, wählen Sie diese im Listenfeld **Available Tasks** aus und klicken Sie auf **Öffnen**.

⇨ Die Taskreferenz wird dem SPS-Projektbaum hinzugefügt. Bei einem erneuten Öffnen des Dialogs **Referenced Tasks hinzufügen** wird der Task im Listenfeld **Used Tasks** angezeigt.

Dialog Referenced Task hinzufügen



Available Tasks	Tasks, die nicht im SPS-Projekt referenziert werden.
Used Tasks	Tasks, die im SPS-Projekt referenziert werden.

New Task

Name	Name des Tasks, der neu erstellt werden soll
Create New Task	Erstellt einen neuen Task

Programmbausteine in dem Task festlegen

1. Selektieren Sie die Taskreferenz im SPS-Projektbaum und wählen Sie im Kontextmenü den Befehl **Hiinzufügen > Vorhandenes Element…**

   ⇨ Der Dialog **Eingabehilfe** öffnet sich.

2. Wählen Sie den Programmbaustein aus, der in dem Task ausgeführt werden soll und bestätigen Sie den Dialog mit **OK**.
3. Alternativ können Sie den Programmbaustein im SPS-Projektbaum direkt mit der Maus auf die Taskreferenz ziehen.

   ⇨ Der Programmbaustein erscheint im SPS-Projektbaum unterhalb der Taskreferenz.

4. Wenn Sie sich die Taskkonfiguration ansehen wollen, doppelklicken Sie auf die Taskreferenz im SPS-Projektbaum. Im Projektmappen-Explorer wird der entsprechende Task im TwinCAT-Projektbaum im Bereich **SYSTEM > Tasks** aktiviert. Durch einen weiteren Doppelklick auf den Task wird die Konfiguration des Tasks in einem Editor geöffnet.

⇨ Im SPS-Projektbaum werden unter der Taskreferenz die Programmbausteine angezeigt, die von dem Task aufgerufen werden. Im Editor des Tasks im Bereich **SYSTEM** kann der Task konfiguriert werden.

Wichtige Hinweise für Multitasking-Systeme

Die Möglichkeit von Mehrkern-Systemen, mehrere Befehle unabhängig voneinander durchzuführen, kann eine Prioritätsumkehrung verursachen.

Beispiel:

Ein Programmbaustein, der von einem Task mit niedriger Priorität aufgerufen werden wird, kann einem unabhängigen Kern zugewiesen werden. In diesem Falle besteht die Möglichkeit, dass die Ausführung des Tasks mit der geringeren Priorität früher abgeschlossen ist, als diejenige eines Tasks mit der hohen Priorität.

7.7 Klassendiagramm erzeugen

Weiterführende Informationen finden Sie in der Dokumentation „TF1910 TC3 UML“ im Abschnitt „Neues Klassendiagramm anlegen“.

7.8 Speicherreserve für Online-Change konfigurieren

Sie können für Funktionsbausteine eine Speicherreserve für den Online-Change konfigurieren. Dadurch müssen, vorausgesetzt die Speicherreserve ist ausreichend groß, nach Änderungen an der Deklaration eines Funktionsbausteins beim anschließenden Online-Change die Instanzen des Funktionsbausteins nicht an neue Speicherplätze kopiert werden. Hier sind vor allem Online-Changes gemeint, bei denen einem Funktionsbaustein eine oder mehrere neue Variablen hinzugefügt werden. Wenn die Funktionsbaustein-Instanzen aufgrund der Speicherreserve nicht an neue Speicherplätze kopiert werden müssen, verläuft der Online-Change schneller und es treten weniger Probleme auf. Wenn die Speicherreserve aufgebraucht ist, erscheint vor der Durchführung des Online-Change eine Meldung.

Konfiguration der Speicherreserve vor dem ersten Download

Sie konfigurieren die Speicherreserve für einen Funktionsbaustein am besten vor dem ersten Download des SPS-Projekts auf die Steuerung. Wenn Sie die Speicherreserve erst konfigurieren, wenn sich das SPS-Projekt bereits auf der Steuerung befindet, ist ein aufwendiger Online-Change notwendig.

Alternative Konfigurationsmöglichkeit

Alternativ zu dem Fenster **Online Change Memory Reserve Settings** kann die Speicherreserve pro Funktionsbaustein auch im Eigenschaften-Fenster konfiguriert werden. Dazu muss der Funktionsbaustein im Projektbaum selektiert sein.

Für einen Funktionsbaustein eine Speicherreserve für den Online-Change konfigurieren

✓ An einem Funktionsbaustein des Projekts sollen zukünftig größere Änderungen vorgenommen werden, die beim Online-Change ein Kopieren der Funktionsbaustein-Instanzen an andere Speicherorte erfordern würde.

✓ Das geöffnete Projekt befindet sich idealerweise noch nicht auf der Steuerung.

1. Wählen Sie im Menü **PLC** den Befehl **Fenster > Online Change Memory Reserve Settings**.
   ⇨ Die Ansicht **Online Change Memory Reserve** öffnet sich.
2. Wählen Sie aus der Auswahlliste das SPS-Projekt.
3. Wählen Sie im Menü **Erstellen** den Befehl **Erstellen**.
4. Klicken Sie auf die Schaltfläche **Applikation durchsuchen**.
5. Wählen Sie im Bereich „Funktionsblöcke“ den Eintrag „Alle“.
   ⇨ In der Ansicht werden alle Funktionsbausteine des SPS-Projekts angezeigt.
6. Wählen Sie die Funktionsbausteine aus, für die Sie eine Speicherreserve konfigurieren möchten.
   ⇨ Wenn sich die Applikation noch nicht auf der Steuerung befindet, ist das Eingabefeld „Speicherreserve (in Bytes)“ editierbar.

7. Wenn sich ein SPS-Projekt bereits auf der Steuerung befindet, klicken Sie auf die Schaltfläche **Erlauben** im Bereich „Bearbeitung erlauben“.
   Beachten Sie: Wenn Sie die Speicherreserve eines SPS-Projekts ändern, das sich bereits auf der Steuerung befindet, müssen die Instanzen aller betroffenen Funktionsbausteine im Speicher kopiert werden.
8. Geben Sie die Größe der Speicherreserve in Bytes ein und klicken Sie auf **Für Auswahl anwenden**.
   ⇨ Die eingegebene Anzahl Bytes wird in der Tabelle im Feld **Speicherreserve** angezeigt.
9. Wählen Sie im Menü **Erstellen** den Befehl **Erstellen**.
10. Klicken Sie auf die Schaltfläche **Applikation durchsuchen**.
    ⇨ In der Funktionsbausteinliste für den konfigurierten Funktionsbaustein werden die Informationen „Größe“, „Instanzanzahl“, „Zusätzlicher Speicher für alle Instanzen“ und „Verbleibende Größe der Speicherreserve“ aktualisiert.
11. Laden Sie das SPS-Projekt auf die Steuerung.

⇨ Die Funktionsbausteininstanzen belegen den aktuell benötigten Speicher und zusätzlich der Speicherreserve. Zukünftige größere Änderungen der Funktionsbausteine können somit über den Online-Change auf die Steuerung geladen werden, ohne dass alle Instanzen der Funktionsbausteine im Speicher umkopiert werden müssen.

Siehe auch:

- Dokumentation TC3 User Interface: Referenz Benutzeroberfläche > PLC > Fenster > Befehl Online Change Memory Reserve Settings [▶ 993]
- Ausführen eines Online-Change [▶ 263]

7.9 Funktionsbaustein, Funktion oder Methode mit externer Implementierung aufrufen

Nur in Ausnahmefällen anwendbar

Die Verwendung dieser Funktionalität ist lediglich in speziellen Konstellationen möglich. In aller Regel können Sie die Option der externen Implementierung ignorieren.

Ein Laufzeitsystem kann die Implementierung eines Funktionsbausteins, einer Funktion oder einer Methode enthalten, zum Beispiel aus einer Bibliothek. Wenn Sie dafür in Ihr SPS-Projekt eine gleichnamige POU mit der Eigenschaft **Externe Implementierung ohne Implementierung** erstellen, können Sie die bereits bestehende Implementierung ausführen. Dabei sollten Sie beachten, dass Sie lokale Variablen nur in einem externen Funktionsbaustein deklarieren. Eine externe Funktion oder Methode darf keine lokale Variable enthalten.

Beim Projektdownload sucht TwinCAT für jede externe POU die zugehörige Implementierung im Laufzeitsystem und verlinkt sie.

POU mit externer Implementierung erstellen

1. Wählen Sie im Kontextmenü des SPS-Projektbaums den Befehl **Hinzufügen > POU..**.
2. Aktivieren Sie Funktionsbaustein oder Funktion und geben Sie als Name den Namen der zugehörigen Implementierung des Laufzeitsystems an. Beenden Sie den Dialog mit **Öffnen**.
   ⇨ Die POU mit dem Namen der Laufzeitsystem-POU ist erstellt.
3. Selektieren Sie die POU und aktivieren Sie die Ansicht **Eigenschaften**.
4. Aktivieren Sie die Option **External implementation** (Spätes Verlinken im Laufzeitsystem).

⇨ Die POU ist deklariert und Sie können einen Aufruf der POU implementieren.

Methode mit externer Implementierung erstellen

1. Selektieren Sie einen Funktionsbaustein im SPS-Projektbaum im **Projektmappen-Explorer**.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Methode** und geben Sie als Name den Namen der zugehörigen Implementierung des Laufzeitsystems an. Beenden Sie den Dialog mit **Open**.
   ⇨ Die Methode ist erstellt.

3. Selektieren Sie die Methode und aktivieren Sie die Ansicht **Eigenschaften**.
4. Aktivieren Sie die Option **External implementation** (Spätes Verlinken im Laufzeitsystem).

⇨ Die Methode ist deklariert und Sie können einen Aufruf der Methode implementieren.

7.10 Eingabeunterstützung nutzen

TwinCAT bietet Funktionalitäten und Assistenten, die Ihnen beim Programmieren die Codeeingabe erleichtern.

Dialog Eingabehilfe

Der Dialog bietet Ihnen alle Programmierelemente an, die Sie an der aktuellen Cursorposition einfügen können. Öffnen Sie den Dialog **Eingabehilfe** mit dem Befehl **Eingabehilfe** im Menü **Bearbeiten** oder im Kontextmenü oder durch das Tastaturkürzel **[F2]**.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Eingabehilfe [▶ 913]

Dialog Variable deklarieren

Ein Dialog unterstützt Sie bei der Deklaration von Variablen. Öffnen Sie den Dialog **Variable deklarieren** mit dem Befehl **Variable deklarieren** im Menü **Bearbeiten** oder im Kontextmenü.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]

Komponenten auflisten

Die Funktion **Komponenten auflisten** ist eine Eingabeunterstützung im Texteditor und erleichtert die Eingabe von gültigen Bezeichnern. Sie aktivieren die Funktion folgendermaßen:

1. Wählen Sie den Befehl **Optionen** im Menü **Extras** und anschließend die Kategorie **TwinCAT > SPS Programmierumgebung > Intelligentes Kodieren**.
2. Aktivieren Sie die Option **Komponenten auflisten, nachdem ein Punkt (.) eingegeben wurde**.
   - Wenn Sie anstelle einer globalen Variablen nur einen Punkt eingeben, erscheint eine Auswahlliste mit allen verfügbaren globalen Variablen. Durch Doppelklicken auf eine Variable der Auswahlliste oder Drücken der Taste **[Eingabe]** fügen Sie die ausgewählte Variable hinter dem Punkt ein.
   - Wenn Sie anstelle einer globalen Variablen oder hinter einer Funktionsbaustein-Instanzvariablen oder einer Strukturvariablen einen Punkt eingeben, bietet TwinCAT in einer Auswahlliste entsprechend alle globalen Variablen, alle Ein- und Ausgabevariablen des Funktionsbausteins oder alle Strukturkomponenten an.
     Durch Doppelklicken auf eine Variable der Auswahlliste oder Drücken der Taste **[Eingabe]** fügen Sie die ausgewählte Variable hinter dem Punkt ein.
     Wenn Sie zusätzlich die lokalen Variablen von Funktionsbaustein-Instanzen erhalten wollen, aktivieren Sie in den TwinCAT-Optionen für **Intelligentes Kodieren** auch die Option **Alle Instanzvariablen in der Eingabehilfe auflisten**.
   - Wenn bereits ein Komponentenzugriff (mit Punkt) für eine Auswahlliste erfolgt ist, wird beim nächsten Komponentenzugriff der zuletzt selektierte Eintrag vorausgewählt.
   - Wenn Sie eine beliebige Zeichenfolge eingeben und anschließend **[Strg] + [Leertaste]** drücken, erscheint eine Auswahlliste mit allen verfügbaren POUs und globalen Variablen. Das erste Element dieser Liste, das mit der vorher eingegebenen Zeichenfolge beginnt, wird automatisch ausgewählt und Sie können es über einen Doppelklick oder über die Taste **[Eingabe]** im Editor einfügen. Übereinstimmungen mit der eingegebenen Zeichenfolge werden in der Auswahlliste gelb hervorgehoben. Wenn die eingegebene Zeichenfolge geändert wird, wird die angezeigte Auswahlliste aktualisiert.
   - Im ST-Editor können Sie die angezeigte Auswahlliste nach Gültigkeitsbereichen filtern:
     Abhängig von der jeweils angezeigten Auswahlliste können Sie mit den Tasten **[Pfeil nach rechts]** und **[Pfeil nach links]** zwischen folgenden Auswahllisten wechseln:

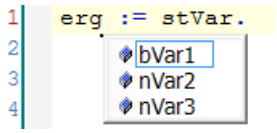
- Alle Einträge
- Schlüsselwörter
- Globale Deklarationen
- Lokale Deklarationen

- Wenn Sie einen Funktionsbaustein, eine Methode oder eine Funktion aufrufen und dazu die öffnende runde Klammer für das Eintragen der POU-Parameter eingeben, zeigt TwinCAT einen Tooltip an. Dieser Tooltip enthält Informationen zu den Parametern, so wie sie innerhalb der POU deklariert sind. Der Tooltip bleibt so lange sichtbar, bis Sie ihn mit einem Mausklick oder durch Setzen des Fokus außerhalb der aktuellen Ansicht schließen. Wenn Sie den Tooltip versehentlich schließen, können Sie ihn mit **[Strg] + [Umschalttaste] + [Leertaste]** wieder öffnen.

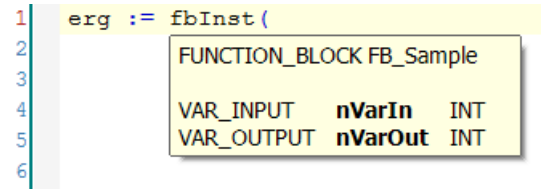
Mit dem Pragmaattribut 'hide' können Sie Variablen aus der Funktion **Komponenten auflisten** ausschließen. (Attribut 'hide' [▶ 841])

Beispiele:

Eingabe einer Strukturvariablen:



Aufruf eines Funktionsbausteins:



Kurzformmodus

Der Kurzformmodus ermöglicht die Eingabe von Kurzformen für die Variablendeklaration im Deklarationseditor und in den Texteditoren, in denen Variablendeklarationen möglich sind. Sie aktivieren diesen Modus, indem Sie eine Deklarationszeile mit der Tastenkombination **[Strg] + [Eingabe]** beenden.

TwinCAT unterstützt folgende Kurzformen:

- Alle Bezeichner bis auf den letzten Bezeichner einer Zeile werden zu Variablenbezeichnern einer Deklaration.
- Der Datentyp der Deklaration wird durch den letzten Bezeichner der Zeile bestimmt. Hierbei gilt:
  - B oder BOOL ergibt BOOL
  - I oder INT ergibt INT
  - R oder REAL ergibt REAL
  - S oder STRING ergibt STRING
- Wenn durch diese Regeln kein Datentyp festgelegt wird, ist der Datentyp automatisch BOOL und der letzte Bezeichner wird nicht als Datentyp benutzt (siehe Beispiel 1).
- Jede eingegebene Konstante wird, je nach Typ der Deklaration, zu einer Initialisierung oder einer Stringlängenangabe (siehe Beispiele 2 und 3).
- Eine Adresse (wie in %MD12) wird automatisch um das AT-Attribut erweitert (siehe Beispiel 4)
- Ein Text nach einem Strichpunkt „;“ wird zu einem Kommentar (siehe Beispiel 3).
- Alle anderen Zeichen in der Zeile werden ignoriert (siehe Ausrufezeichen in Beispiel 5).

Beispiele:

Beispiel	Kurzform	resultierende Deklaration
1	bA	bA: BOOL;
2	nA nB I 2	nA, nB: INT := 2;
3	sC S 2; C string	sC: STRING(2); // C string
4	X %MD12 R 5	X AT %MD12: REAL := 5.0;
5	bE !	bE: BOOL;

Smart-Tag-Funktionen

Verfügbar ab TC3.1 Build 4026

Smart Tags erleichtern das Erstellen von Programmcode, indem sie direkt beim Programmierelement geeignete Befehle zur Auswahl bereitstellen. Wenn Sie den Cursor auf ein Programmierelement setzen, für das eine Smart-Tag-Funktion zur Verfügung steht, erscheint das Symbol  . Bei einem Klick auf  werden die Befehle angezeigt, die Sie auswählen können. Verfügbare Smart Tags:

- Bei nicht deklarierten Variablen im Implementierungsteil im ST-Editor stellt die Smart-Tag-Funktion den Befehl **Variable deklarieren** zur Verfügung.

Siehe auch:

- Variablen deklarieren [▶ 68]
- Dokumentation TC3 User Interface: Dialog Optionen - Intelligentes Codieren [▶ 1026]

7.11 Pragmas verwenden

Pragmas in TwinCAT

Als Pragma gilt ein Text im Quellcode der Applikation, der in geschweiften Klammern steht. Pragmas werden verwendet, um spezielle Anweisungen im Code einzufügen, die der Compiler auswerten kann. Damit kann ein Pragma die Eigenschaften einer oder mehrerer Variablen bezüglich der Vorkompilierung oder der Kompilierung (Codegenerierung) beeinflussen. Pragmas, die der Compiler nicht kennt, überliest er wie einen Kommentar.

Der Anweisungsstring eines Pragmas kann auch mehrzeilig sein. Zur Syntax im Detail sehen Sie bitte die Beschreibungen der einzelnen Pragmas.

Es gibt Pragmas für unterschiedliche Effekte: Initialisierung einer Variablen, Monitoring einer Variablen, Erzwingen von Meldungsausgaben während des Übersetzungsvorgangs, Verhalten einer Variablen unter bestimmten Bedingungen etc..

Die Groß-/Kleinschreibung muss eingehalten werden.

Beispiel:

```
{warning 'This is not allowed'}
{attribute 'obsolete' := 'datatype FB_Sample not valid!'}
```

Mögliche Einfügepositionen

Pragmas in TwinCAT sind keine Eins-zu Eins-Implementierungen der C-Präprozessor-Direktiven. Sie müssen ein Pragma wie eine normale Anweisung positionieren. Sie dürfen ein Pragma nicht innerhalb eines Ausdrucks verwenden.

Ein Pragma, das der Compiler auswerten soll, können Sie an folgenden Positionen einfügen:

- Im Deklarationsteil eines Programmierbausteins:
  - Im textuellen Deklarationseditor geben Sie Pragmas direkt als Zeile(n) ein, entweder am Anfang des Bausteins oder vor einer Variablendeklaration.
  - Im tabellarischen Editor geben Sie Pragmas, die oberhalb der ersten Deklarationszeile stehen sollen, im Dialog **Attribute** ein. Doppelklick auf die Spalte **Attribute**.
- In einer globalen Variablenliste
- Im Implementierungsteil eines Programmierbausteins:
  - Das Pragma muss an einer „Anweisungsposition“ stehen, also am Anfang eines Programmierbausteins in einer separaten Zeile, oder nach einem „;“ oder END_IF, END_WHILE etc..
  - FBD/LD/AWL-Editor: Pragmas in Netzwerken des FUP/KOP/AWL-Editors geben Sie wie eine Sprungmarke ein:
    Wählen Sie dazu den Befehl **FBD/LD/IL > Sprungmarke einfügen** und ersetzen dann den Standardtext **Label:** im Textfeld der Marke durch die entsprechende Pragmaanweisung. Wenn Sie ein Pragma zusätzlich zu einer Sprungmarke verwenden wollen, tragen Sie zunächst das Pragma und dann die Sprungmarke ein.

Falsche und richtige Positionierung eines bedingten Pragmas:

Falsch:	Richtig:

`{IF defined(abc)}`
`IF x = abc THEN`
`{ELSE}`
`IF x = 12 THEN`
`{END_IF}`
`y := {IF defined(cde)} 12;`
`{ELSE} 13;`
`{END_IF}`
`END_IF`
	`{IF defined(abc)}`
	`IF x = abc THEN`
	`{IF defined(cde)}`
	`y := 12;`
	`{ELSE}`
	`y := 13;`
	`{END_IF}`
	`END_IF`
	`{ELSE}`
	`IF x = 12 THEN`
	`{IF defined(cde)}`
	`y := 12;`
	`{ELSE}`
	`y := 13;`
	`{END_IF}`
	`END_IF`
	`{END_IF}`

In den **Eigenschaften** der POU, Kategorie **Advanced**, können Sie **Defines** angeben, die in Pragmas abgefragt werden können.

Wirkungsbereich:

Abhängig vom Typ und Inhalt eines Pragmas wirkt ein Pragma auf Folgendes:

- die nachfolgenden Deklarationen
- genau auf die nachfolgende Anweisung
- auf alle nachfolgenden Anweisungen, bis es mit einem entsprechenden Pragma wieder aufgehoben wird.
- auf alle nachfolgenden Anweisungen, bis dasselbe Pragma mit anderen Parametern ausgeführt oder das Ende des Codes erreicht wird. „Code“ in diesem Kontext heißt: Deklarationsteil, Implementierungsteil, globale Variablenliste, Typdeklaration. Somit wirkt ein Pragma, das allein in der ersten Zeile des Deklarationsteils steht und nicht durch ein weiteres abgelöst oder aufgehoben wird, auf das gesamte Objekt.

Pragma-Kategorien

Die TwinCAT-Pragmas sind in folgende Kategorien aufgeteilt:

- Attributpragmas: Beeinflussung der Kompilierung und der Vorkompilierung (Attribut-Pragmas [▶ 829])
- Meldungspragmas: Ausgabe von benutzerdefinierten Meldungen während des Übersetzungsvorgangs (Meldungspragmas [▶ 828])

- Bedingte Pragmas: Beeinflussung der Codegenerierung (Bedingte Pragmas [▶ 875])
- Benutzerdefinierte Pragmas (Benutzerdefinierte Attribute [▶ 830])

Siehe auch:

- Dialog Variable deklarieren verwenden [▶ 73]

7.12 Text in einer Textliste verwalten

Textlisten dienen dazu, Texte für eine Visualisierung in mehreren Sprachen bereitzustellen. Sie können die Texte im Unicode-Format eingeben, so dass Ihnen sämtliche Sprachen und Schriftzeichen zur Verfügung stehen. Sie können Textlisten exportieren und importieren und damit die Texte außerhalb des aktuellen Projekts übersetzen.

TwinCAT unterscheidet zwischen statischem Text, der im Objekt „GlobalTextList“ verwaltet wird, und dynamischem Text, der in Objekten des Typs „Textliste“ verwaltet wird.

Statische Texte sind Texte innerhalb der Visualisierung, die zur Laufzeit nur ihre Sprache ändern können. Die Text-ID bleibt dabei konstant.

Dynamische Texte können Sie über eine IEC-Variablen steuern, die die Text-ID enthält. Damit können Sie in einem Visualisierungselement zur Laufzeit variierenden Text anzeigen lassen. Zum Beispiel können Sie ein Textfeld so konfigurieren, dass es einen Fehlertext zu einer Fehlernummer ausgibt.

Beide Textlistentypen enthalten eine Tabelle mit Texteinträgen. Ein Eintrag besteht aus einer ID zur Identifizierung, dem Ausgangstext und dessen Übersetzungen. In einer Textliste oder einer globalen Textliste können Sie einen Ausgangstext in beliebig vielen Sprachen übersetzen. Die Übersetzungen sind die Basis für die Sprachauswahl und die Sprachumschaltung in Visualisierungen.

Verzeichnis für Textlistendateien

Die Verzeichnisse, die Textlisten für die Visualisierung bereitstellen, legen Sie in den Projekteigenschaften in der Kategorie **Visualization** fest.

Siehe auch:

- Eingabeunterstützung nutzen [▶ 141]
- Dokumentation TC3 User Interface: Textliste [▶ 1093]

Sprache hinzufügen und Text übersetzen

✓ Ein Projekt mit Textliste oder globaler Textliste ist geöffnet.

1. Doppelklicken Sie im SPS-Projektbaum auf ein Objekt des Typs **Textliste** oder **GlobalTextList**.
   ⇨ Das Menü **Textliste** erscheint in der Menüleiste und die Textliste öffnet im Editor.
2. Wählen Sie im Menü **Textliste** oder im Kontextmenü den Befehl **Sprache einfügen**.
3. Geben Sie einen Namen für die Sprache ein, zum Beispiel „en-US“. Beenden Sie den Dialog mit **OK**.
   ⇨ Eine Spalte mit der Überschrift **en-US** erscheint.
4. Geben Sie in der Spalte die Übersetzung des Ausgangstextes ein.

⇨ Sie können so beliebig viele Sprachen ergänzen.

Im Gegensatz zum Text in der Spalte **Standard** müssen Sie in der Sprachspalte keine Übersetzung eintragen. Wenn für einen Texteintrag keine Übersetzung gefunden wird, wird automatisch der Standardtext angezeigt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Sprache einfügen [▶ 1093]

Textliste exportieren

✓ Ein Projekt mit Textliste oder globaler Textliste ist geöffnet.

1. Doppelklicken Sie im SPS-Projektbaum auf ein Objekt des Typs **Textliste** oder **GlobalTextList**.
   ⇨ Das Menü **Textliste** erscheint in der Menüleiste und die Textliste öffnet im Editor.
2. Wählen Sie im Menü **Textlist** oder im Kontextmenü den Befehl **Import/Export Textlisten**.
   ⇨ Der Dialog **Import/Export** öffnet sich.
3. Klicken Sie bei **Exportdatei auswählen** auf … und wählen Sie dort das Verzeichnis und geben Sie einen Dateinamen ein, zum Beispiel „Text_lists_exported“.
4. Aktivieren Sie die Option **Exportieren**.
5. Beenden Sie den Dialog **Import/Export** mit **OK**.

⇨ TwinCAT exportiert die Textlisteneinträge aller Textlisten des Projekts in eine .csv-Datei. Die Tabelle enthält eine Spalte mit dem Textlistennamen.

Über den Befehl **Alles als Text exportieren** können Sie die Textlisteneinträge als .txt-Datei exportieren. Für jede Textliste wird eine Datei angelegt. Das Verzeichnis, in dem die Export-Dateien automatisch gespeichert werden, definieren Sie in den Projekteigenschaften.

Beispiel:

Inhalt der Datei Text_lists_exported

```
TextList        Id    Default         en-us
TextList_A      A     Information A   Information A_en
TextList_A      B     Information B   Information B_en: Ok
TextList_A      C     Information C   Information C_en
AlarmGroup      1     Warnung 1
AlarmGroup      2     Warnung 2
GlobalTextList        Information G   Information G_en
GlobalTextList        Information H   Information H_en
GlobalTextList        Umschalten      Switch
GlobalTextList        Zähler: %i      Counter: %i
```

Siehe auch:

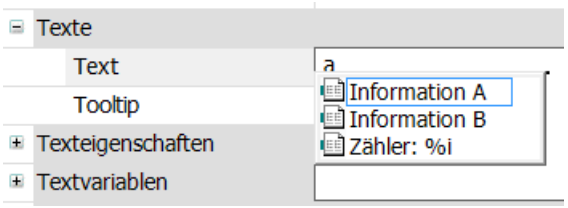
- Dokumentation TC3 User Interface: Befehl Import/Export Textlisten [▶ 1094]

Exportierte Datei für die Eingabehilfe bereitstellen

✓ Sie haben eine Datei, zum Beispiel Text_lists_exported, über den Befehl **Import/Export Textlisten** erstellt. Sie enthält die Texte der Textlisten des Projekts.

1. Wählen Sie den im Menü **Extras** den Befehl **Optionen**, Kategorie **TwinCAT > SPS Programmierumgebung > Visualisierung**, Registerkarte **Dateioptionen**.
2. Klicken Sie in **Textdatei für textuelles Komponenten auflisten** auf … und wählen Sie eine Datei, zum Beispiel Text_lists_exported. Beenden Sie den Dialog mit **OK**.

⇨ Wenn Sie in einer Visualisierung bei einem Element einen statischen Text unter der Eigenschaft **Texte** eingeben, bietet TwinCAT beim Eingeben des ersten Buchstaben die Ausgangstexte der Datei als Eingabehilfe zur Auswahl an.



Siehe auch:

- Eingabeunterstützung nutzen [▶ 141]

Datei mit Textlisteneinträgen importieren

Eine importierbare Datei ist eine Datei mit Format .csv. Die erste Zeile ist eine Kopfzeile, wie zum Beispiel „TextList Id Default en_US“. Die weiteren Zeilen enthalten Textlisteneinträge. So eine Datei erhalten Sie, wenn Sie die Textlisten des Projekts in eine Datei exportierten. Sie können dort, außerhalb von TwinCAT, die Textlisteneinträge bearbeiten und die Datei anschließend importieren. TwinCAT behandelt beim Import die Textlisteneinträge für die GlobalTextList und für dynamische Textlisten aber unterschiedlich.

GlobalTextList:

- Bei einer unbekannten ID legt TwinCAT keine neuen Textlisteneinträge an.
- TwinCAT ignoriert Änderungen, die die ID oder den Ausgangstext betreffen.
- TwinCAT übernimmt Änderungen der Übersetzungen.

Textliste:

- Bei einer neuen ID ergänzt TwinCAT die zugehörige Textliste um einen Textlisteneintrag.
- Bei einer bestehenden ID und unterschiedlichen Ausgangstexten wird der Ausgangstext der Textliste mit dem Ausgangstext der Datei überschrieben.
- TwinCAT übernimmt Änderungen der Übersetzungen.

Datei importieren:

✓ Ein Projekt mit Textliste oder globaler Textliste ist geöffnet.

1. Doppelklicken Sie im SPS-Projektbaum auf ein Objekt des Typs **Textliste** oder **GlobalTextList**.
   ⇨ Das Menü **Textliste** erscheint in der Menüleiste und die Textliste öffnet im Editor.
2. Wählen Sie im Menü **Textlist** oder im Kontextmenü den Befehl **Import/Export Textliste**
   ⇨ Der Dialog **Import/Export** öffnet sich.
3. Klicken Sie in **Datei für Vergleich oder Import auswählen** auf … und wählen Sie dort ein Verzeichnis und einen Dateinamen, zum Beispiel „Text_lists_corrected.csv“.
4. Aktivieren Sie die Option **Importieren**.
5. Beenden Sie den Dialog mit **OK**.

⇨ TwinCAT importiert die Textlisteneinträge der Datei in die zugehörigen Textlisten.

Beispiel:

Inhalt der Datei Text_lists_corrected.csv

```
TextList        Id    Default             en-us
TextList_A      A     Information A       Information A_en
TextList_A      B     Information B: Ok   Information B_en: Ok
TextList_A      C     Information C       Information C_en
TextList_A      D     Information D       Information D_en
AlarmGroup      1     Warnung 1           Warning 1
AlarmGroup      2     Warnung 2
GlobalTextList        Information G       Information G_en: Ok
GlobalTextList        Information HH      Information H_en
GlobalTextList        Information I       Information I_en
GlobalTextList        Umschalten          Switch
GlobalTextList        Zähler: %i          Counter: %i
```

Dieser Inhalt wird so im Projekt in den gleichnamigen Textlisten übernommen.

```
TextList        Id    Default             en-us
TextList_A      A     Information A       Information A_en
TextList_A      B     Information B: Ok   Information B_en:  Ok
TextList_A      C     Information C       Information C_en
TextList_A      D     Information D       Information D_en
AlarmGroup      1     Warnung 1           Warning 1
AlarmGroup      2     Warnung 2
GlobalTextList        Information G       Information G_en: Ok
GlobalTextList        Information H       Information H_en
GlobalTextList        Umschalten          Switch
GlobalTextList        Zähler:  %i         Counter:  %i
```

Textlisten mit Datei vergleichen und Unterschied exportieren

✓ Ein Projekt mit Textliste oder globaler Textliste ist geöffnet.

1. Doppelklicken Sie im SPS-Projektbaum auf ein Objekt des Typs **Textliste** oder **GlobalTextList**.
   ⇨ Das Menü **Textliste** erscheint in der Menüleiste und die Textliste öffnet im Editor.
2. Wählen Sie im Menü **Textlist** oder im Kontextmenü den Befehl **Import/Export Textlisten**.
   ⇨ Der Dialog **Import/Export** öffnet sich.
3. Klicken Sie in **Datei für Vergleich oder Import auswählen** auf … und wählen Sie dort Verzeichnis und Dateiname der Vergleichsdatei aus, zum Beispiel „Text_lists_corrected.csv“.
4. Klicken Sie bei **Exportdatei auswählen** auf … und wählen Sie dort ein Verzeichnis und geben Sie einen Dateinamen für die Datei ein, die das Vergleichsergebnis enthalten wird.
5. Aktivieren Sie die Option **Nur Textunterschiede exportieren**.
6. Beenden Sie den Dialog mit **OK**.

⇨ TwinCAT liest die Importdatei und vergleicht die Textlisteneinträge, die die gleiche ID haben. Wenn sie nicht übereinstimmen, schreibt TwinCAT die Textlisteneinträge der Textliste in die Exportdatei. Bei der globalen Textliste vergleicht TwinCAT bei gleichen Ausgangstexten die Übersetzungen. Wenn sie nicht übereinstimmen, schreibt TwinCAT die Textlisteneinträge in die Exportdatei.

Beispiel:

```
TextList        Id    Default         en-us
TextList_A      B     Information B   Information B_en: Ok
AlarmGroup      1     Warnung 1       Warning 1
GlobalTextList        Information G   Information G_en
GlobalTextList        Information H   Information H_en
```

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Import/Export Textlisten [▶ 1094]

7.12.1 Statischen Text in einer globalen Textliste verwalten

Die globale Textliste ist die zentrale Stelle im Projekt für Texte, die in der Visualisierung ausgegeben werden.

Wenn Sie in der Visualisierung in einem Element einen Text erstmals konfigurieren, erstellt TwinCAT automatisch eine globale Textliste. Sie wird dem SPS-Projektbaum als Objekt hinzugefügt und ist höchstens einmal vorhanden. Sie können die globale Textliste über einen Doppelklick auf das Objekt in einem Editor öffnen.

Die globale Textliste enthält in einer Tabelle alle statischen Texte, die Sie in den Visualisierungen des Projekts verfasst haben. Wenn Sie in einer Visualisierung in einem Element unter der Eigenschaft **Texte** einen weiteren Text verfassen, ergänzt TwinCAT die Tabelle automatisch. Die IDs vergibt TwinCAT als fortlaufende ganzen Zahlen beginnend mit 0.

Sie können die globale Textliste gegen die statischen Texte der Visualisierungen prüfen, aktualisieren und abgleichen. Sie können hier keinen neuen Text verfassen, sondern nur einen bestehenden Text bearbeiten und übersetzen. Sie können den Ausgangstext oder die ID in der Tabelle jedoch nicht direkt bearbeiten. Aber Sie können einen Ausgangstext durch einen anderen ersetzen, indem Sie eine Ersetzungsdatei erstellen und diese importieren. Dafür stehen Ihnen Menübefehle zur Verfügung.

TwinCAT stellt folgende Befehle zur Verfügung, um die GlobalTextList zu konsolidieren:

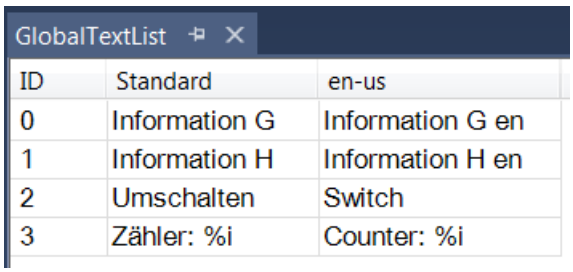
- Visualisierungstext-IDs prüfen
- Visualisierungstext-IDs aktualisieren
- Nicht verwendete Textlisteneinträge entfernen

Siehe auch:

- Dokumentation TC3 User Interface: Textliste [▶ 1093]

Struktur einer globalen Textliste

Symbol: 



GlobalTextList

ID	Standard	en-us
0	Information G	Information G en
1	Information H	Information H en
2	Umschalten	Switch
3	Zähler: %i	Counter: %i

ID	Eindeutiger Identifikator des Texts
Standard	Ausgangstext als Zeichenkette mit maximal einer Formatierungsangabe, zum Beispiel Information A: %i Möglichkeiten . Wenn unter einer Sprachspalte keine Übersetzung verfasst ist, verwendet TwinCAT diesen Text. Doppelklicken Sie in das Feld, um den Text zu editieren.
Die Tabelle enthält beliebig viele von Ihnen hinzugefügte Sprachspalten. Eine Sprachspalte ist mit einem Sprachkürzel betitelt, das Sie beim Erstellen der Spalte mit dem Befehl Sprache einfügen angegeben haben.	
<Sprachkürzel>	Name der Sprache als Sprachkürzel. Zum Beispiel en-US. Diese Spalte enthält die Übersetzung des Texts, der unter Standard verfasst ist. Wenn im Visualisierungsmanager das Sprachkürzel als Sprache ausgewählt ist, gibt eine Visualisierung im laufenden Betrieb die Übersetzung aus. Eine Visualisierung im laufenden Betrieb kann auf Anforderung eines Benutzers auf eine andere Sprache umschalten. Doppelklicken Sie in das Feld, um den Text zu editieren.

Visualisierungselement mit statischem Text konfigurieren

Ein Text in einer GlobalTextList kann eine Formatierungsangabe enthalten.

✓ Ein Projekt mit Visualisierung ist geöffnet. Das Objekt **GlobalTextList** enthält die Texte, die in den Visualisierungen des Projekts definiert sind.

1. Doppelklicken Sie im SPS-Projektbaum auf die Visualisierung.
   ⇨ Der Editor öffnet sich.
2. Selektieren Sie ein Element, das über die Eigenschaft **Text** verfügt, zum Beispiel ein Textfeld.
3. Geben Sie bei der Eigenschaft **Text** einen Text ein, zum Beispiel „Statische Information A“.

⇨ TwinCAT erweitert die globale Textliste in der POUs-Ansicht um den neuen Text.

Globale Textliste prüfen

✓ Ein Projekt mit Visualisierung ist geöffnet. Das Objekt **GlobalTextList** enthält die Texte, die in den Visualisierungen des Projekts definiert sind.

1. Doppelklicken Sie im SPS-Projektbaum auf Objekt **GlobalTextList**.
   ⇨ Die Tabelle mit den statischen Texten als Listeneinträge öffnet sich.
2. Wählen Sie im Menü **Textliste** oder im Kontextmenü des Editors den Befehl **Visualisierungstext-IDs prüfen**.

⇨ TwinCAT meldet, wenn ein Ausgangstext der Textliste nicht übereinstimmt mit dem statischen Text, der über die ID identifiziert ist. Der Ausgangstext in der globalen Textliste und der Text in den Visualisierungen, die die gleiche ID haben, stimmen nicht überein.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Visualisierungstext-IDs prüfen [▶ 1096]

IDs der Globalen Textliste aktualisieren

✓ Ein Projekt mit Visualisierung ist geöffnet. Das Objekt **GlobalTextLis**t enthält die Texte, die in den Visualisierungen des Projekts definiert sind.

1. Doppelklicken Sie im SPS-Projektbaum auf das Objekt **GlobalTextList**.

⇨ Die Tabelle mit den statischen Texten als Listeneinträge öffnet sich.

2. Wählen Sie im Menü **Textliste** oder im Kontextmenü des Editors den Befehl **Visualisierungstext-IDs aktualisieren**.

⇨ TwinCAT ergänzt die globale Textliste, wenn in den Visualisierungen des Projekts ein Text in der Eigenschaft Statischer Text nicht mit einem Ausgangstext übereinstimmt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Visualisierungstext-IDs aktualisieren [▶ 1096]

IDs aus der globalen Textliste entfernen

✓ Ein Projekt mit Visualisierung ist geöffnet. Das Objekt **GlobalTextList** enthält die Texte, die in den Visualisierungen des Projekts definiert waren.

1. Doppelklicken Sie im SPS-Projektbaum auf Objekt **GlobalTextList**.

⇨ Tabelle mit den Texten öffnet sich.

2. Wählen Sie im Menü **Textliste** oder im Kontextmenü des Editors den Befehl **Nicht verwendete Textlisteneinträge entfernen**.

⇨ TwinCAT entfernt die Textlisteneinträge, deren IDs in den Visualisierungen des Projekts nicht referenziert sind.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Nicht verwendete Textlisteneinträge entfernen [▶ 1095]

Globale Textliste mit Ersetzungsdatei bearbeiten

Eine Ersetzungsdatei ist eine Datei im Format .csv. Die erste Zeile ist eine Kopfzeile: defaultalt defaultneu REPLACE. Die weiteren Zeilen enthalten den alten Ausgangstext, den neuen Ausgangstext und dann den Befehl REPLACE. Als Trennzeichen sind Tabulator, Komma oder Strichpunkt erlaubt. Eine Mischung der Trennzeichen innerhalb einer Datei ist nicht erlaubt.

Beispiel (Tabulator als Trennzeichen):

```
defaultalt     defaultneu      REPLACE
Information A  Information A1  REPLACE
```

Wenn Sie eine Ersetzungsdatei importieren, arbeitet TwinCAT die Ersetzungsdatei zeilenweise ab und führt die spezifizierten Ersetzungen in der **GlobalTextList** durch. Außerdem ersetzt TwinCAT in den Visualisierungen den bisherigen Text durch den Ersatztext. Wenn der Ersatztext bereits als statischer Text vorhanden war, erkennt TwinCAT das, harmonisiert die statischen Texte und belässt nur einen Textlisteneintrag.

✓ Ein Projekt mit Textliste oder globaler Textliste ist geöffnet.

1. Doppelklicken Sie im SPS-Projektbaum auf das Objekt **GlobalTextList**.

⇨ Das Objekt öffnet sich.

2. Wählen Sie im Menü **Textliste** oder im Kontextmenü des Editors den Befehl **Import/Export Textlisten**.

⇨ Der Dialog **Import/Export** öffnet sich.

3. Klicken Sie bei **Datei für Vergleich oder Import auswählen** auf … und wählen Sie dort ein Verzeichnis und einen Dateinamen, zum Beispiel „ReplaceGlobalTextList.csv“.
4. Aktivieren Sie die Option **Ersetzungsdatei importieren**.
5. Beenden Sie den Dialog mit **OK**.

⇨ Die Texte in den Textlisten und den Visualisierungen sind ersetzt.

Beispiel:

Die globale Textliste enthält die folgenden Ausgangstexte:

```
GlobalTextList  Counter:%i
GlobalTextList  Information A
GlobalTextList  Information Aa
GlobalTextList  Umschalten
GlobalTextList  Zähler:%i
```

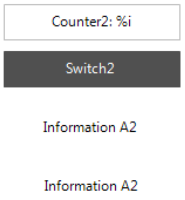
Die Ersetzungsdatei enthält folgende Ersetzungen:

```
defaultalt      defaultneu     REPLACE
Counter:%i      Counter2:%i    REPLACE
Information A   Information A2 REPLACE
Information Aa  Information A2 REPLACE
Umschalten      Switch         REPLACE
Zähler:%i       Counter2:%i    REPLACE
```

TwinCAT erkennt doppelte Textlisteneinträge und entfernt einen. Die globale Textliste enthält danach folgende Einträge:

5	Switch2
4	Counter2: %i
3	Information A2

Die Texte in der Visualisierung sind ausgetauscht.



Siehe auch:

- Dokumentation TC3 User Interface: Befehl Import/Export Textlisten [▶ 1094]

7.12.2 Dynamischen Text in einer Textliste verwalten

Sie können in einer Textliste für dynamischen Text Texte verwalten, erstellen und übersetzen. Einen Text, den Sie hier verfasst haben, können Sie in einer Visualisierung in einem Element in der Eigenschaft **Dynamische Texte** auswählen. Die Visualisierung im laufenden Betrieb gibt diesen Text in der ausgewählten Sprache dynamisch aus.

Sie legen eine Textliste als Objekt im SPS-Projektbaum an. Die Textliste enthält eine Tabelle mit Textlisteneinträgen, die Sie bearbeiten und erweitern können. Ein Textlisteneintrag besteht aus einer ID zur Identifizierung, dem Ausgangstext und dessen Übersetzungen. Sie können einer Textliste neue Textlisteneinträge hinzufügen. Dafür stehen Ihnen Menübefehle zur Verfügung.

Siehe auch:

- Dokumentation TC3 User Interface: Textliste [▶ 1093]

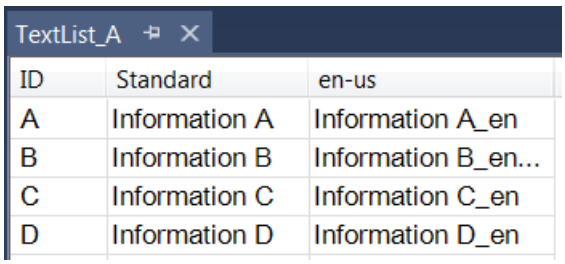
Objekt Textliste anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Textliste…**
   ⇨ Der Dialog **Textliste hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein.
4. Klicken Sie auf **Öffnen**.

⇨ Die Textliste wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Struktur einer Textliste

Symbol: 


TextList_A

ID	Standard	en-us
A	Information A	Information A_en
B	Information B	Information B_en...
C	Information C	Information C_en
D	Information D	Information D_en

ID	Eindeutiger Identifikator des Texts
Standard	Ausgangstext als Zeichenkette, zum Beispiel Information A Doppelklicken Sie in das Feld, um den Text zu editieren.
Die Tabelle enthält beliebig viele von Ihnen hinzugefügte Sprachspalten. Eine Sprachspalte ist mit einem Sprachkürzel betitelt, das Sie beim Erstellen der Spalte mit dem Befehl Sprache einfügen angegeben haben.	
<Sprachkürzel>	Name der Sprache als Sprachkürzel. Zum Beispiel en-US. Unter dieser Spalte ist die Übersetzung des Texts, der unter Standard verfasst ist. Unter der Voraussetzung, dass im Visualisierungsmanager das Sprachkürzel als Sprache ausgewählt ist, gibt eine Visualisierung im laufenden Betrieb den übersetzten Text aus. Wenn keine Übersetzung verfasst ist, verwendet TwinCAT den Text unter Standard. Eine Visualisierung im laufenden Betrieb kann auch eine Sprachumschaltung, die von einem Benutzer angefordert wird, durchführen.
Leere Zeile	Editieren Sie die Zeile, um einen eigenen Text hinzuzufügen.

Textliste für dynamische Textausgabe erstellen

✓ Ein Projekt mit Visualisierung ist geöffnet.

1. Selektieren Sie **Projektmappen-Explorer** im SPS-Projektbaum das SPS-Projektobjekt oder einen darunterliegenden Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Textliste**.
3. Geben Sie einen Namen ein, zum Beispiel „Textliste_A“ und beenden Sie den Dialog mit "Open".
   ⇨ Ein Objekt des Typs **Textliste** ist erstellt.
4. Klicken Sie unter Spalte **Standard** und öffnen Sie das Eingabefeld. Geben Sie einen Text ein, zum Beispiel „Information A“.
   ⇨ Der Ausgangstext ist erstellt. Er dient als Schlüssel in der Tabelle und als Ausgangstext für Übersetzungen.
5. Geben Sie in Spalte **ID** eine beliebige Zeichenkette ein, zum Beispiel „A“.
6. Doppelklicken Sie in die leere Zeile am Ende der Tabelle unter Standard und geben Sie weitere Textlisteneinträge ein.
   ⇨ Die Textlisteneinträge mit Ausgangstext und ID sind definiert. Wenn Sie in einer Visualisierung die Eigenschaft **Dynamische Texte** eines Elements konfigurieren, können Sie nun zum Beispiel die Textliste „Textliste_A“ auswählen und die ID „A“ zuweisen.

Text dynamisch ausgeben

Sie können in einer Visualisierung die dynamische Ausgabe von Texten konfigurieren, die in einer Textliste verfasst wurden, indem Sie die Eigenschaft **Dynamische Texte** eines Elements konfigurieren. Sie können direkt eine Textliste und eine ID zuweisen, aber auch IEC-Variablen, in der Sie programmatisch die Werte setzen.

✓ Ein Projekt mit Visualisierung ist geöffnet und eine Textliste ist im SPS-Projektbaum vorhanden.

1. Öffnen Sie die Textliste im Editor, zum Beispiel „Textliste_A“.
2. Doppelklicken Sie auf das Visualisierungsobjekt.
   ⇨ Der Editor öffnet sich.
3. Ziehen Sie ein Element, zum Beispiel des Typs **Textfeld**, in die Visualisierung.

4. Konfigurieren Sie dessen Eigenschaft **Dynamische Texte**, indem Sie in der Eigenschaft **Textliste** eine auswählen, zum Beispiel „Text_list_A“ und in **Textindex** eine ID aus der Textliste einfügen, zum Beispiel „A“. Achten Sie auf die Hochkommata. Sie können auch jeweils eine IEC-Variable des Typs STRING für den Textlistennamen und der ID zuweisen.
   ⇨ Die IEC-Variablen ermöglichen programmatisch den Zugriff auf die Texte der Textlisten.
5. Übersetzen Sie die Anwendung, laden Sie sie auf die Steuerung und starten Sie sie.

⇨ Die Visualisierung gibt im Textfeld den Text aus der Textliste aus: Information A.

7.13 Bildersammlungen verwenden

Eine Bildersammlung ist eine Tabelle mit Bilddateien. Durch Angabe der ID und zusätzlich des Namens der Bildersammlung kann TwinCAT eine Bilddatei eindeutig referenzieren, wenn Sie sie im Projekt, beispielsweise in einer Visualisierung, verwenden. Ein Projekt kann mehrere Bildersammlungen enthalten. Sie können Bildersammlungen dem SPS-Projektbaum im Projektmappen-Explorer hinzufügen. In einem Bibliotheksprojekt können Sie eine Bildersammlung über ihre Objekteigenschaften zu einer Symbolbibliothek für die Visualisierung machen.

Wir empfehlen, die Größe einer Bilddatei vor ihrer Einbindung so weit wie möglich zu reduzieren. Dies optimiert die Ladezeit der Visualisierung in allen Visualisierungsvarianten (TargetVisu, WebVisu und Programmiersystem).

Wenn Sie ein Bildelement in eine Visualisierung einfügen und eine ID (Statische ID) in den Elementeigenschaften eintragen, legt TwinCAT automatisch eine globale Bildersammlung an. Dabei verwendet TwinCAT den Standardnamen „GlobalImagePool“.

Beachten Sie Folgendes, wenn die ID einer Bilddatei in mehreren Bildersammlungen vorkommt:

- Suchreihenfolge: Wenn Sie ein Bild wählen, das im GlobalImagePool verwaltet wird, müssen Sie den Name der Bildersammlung nicht angeben. Die Suchreihenfolge für Bilddateien entspricht folgender Reihenfolge:
  - 1. Globaler GlobalImagePool
  - 2. Bildersammlungen, die des gerade aktiven SPS-Projekts zugewiesen sind
  - 3. Bildersammlungen, die neben dem GlobalImagePool im **Projektmappen-Explorer** liegen
  - 4. Bildersammlungen in Bibliotheken
- Eindeutiger Zugriff: Sie können das gewünschte Bild direkt und eindeutig ansprechen, indem Sie der ID den Namen der Bildersammlung voranstellen, gemäß Syntax <Name der Sammlung>.<Bild ID>.

7.13.1 Objekt Bildersammlung

Symbol: 

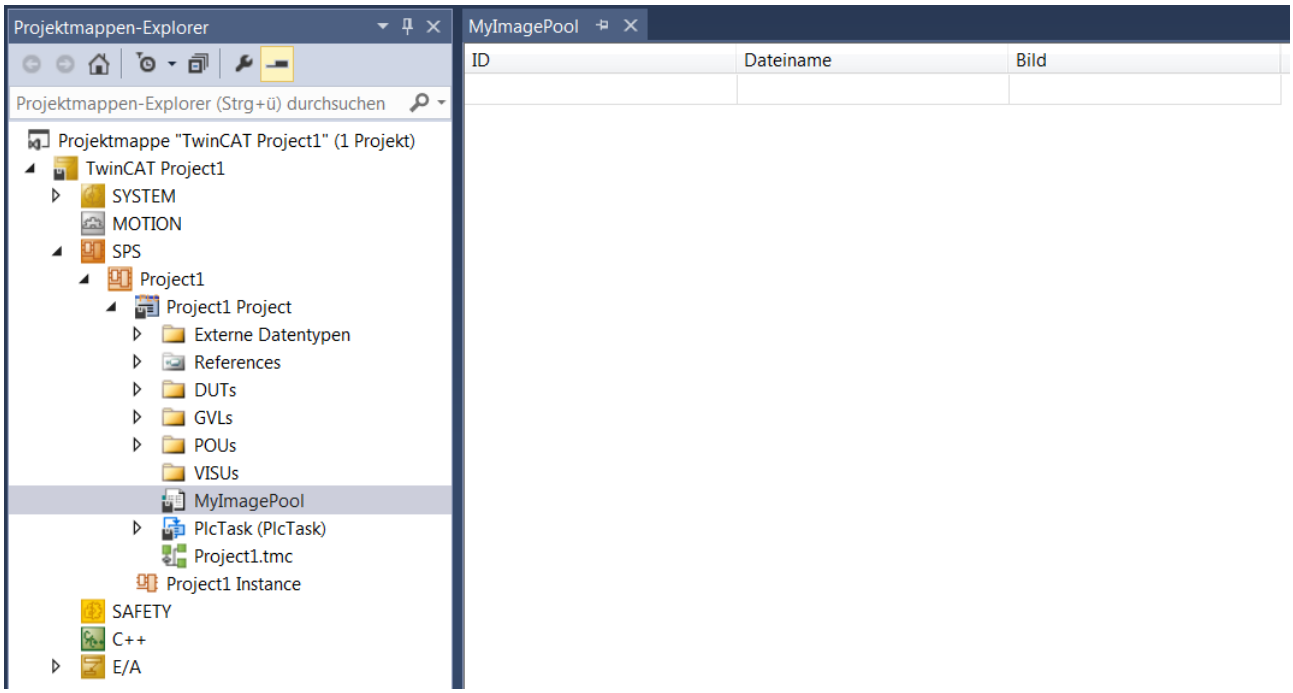
Das Objekt **Bildersammlung** beinhaltet eine Tabelle, in der Bilder IDs zugeordnet sind.

Objekt Bildersammlung anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Bildersammlung**.
   ⇨ Der Dialog **Bildersammlung hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein.
4. Klicken Sie auf **Öffnen**.

⇨ Die Bildersammlung wird zu m SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Struktur einer Bildersammlung



ID	ID des Bilds. Über diese ID referenzieren Sie zum Beispiel in der Visualisierung das Bild.
Dateiname	Dateipfad der Bilddatei. Wenn Sie auf die Schaltfläche … klicken, öffnet sich der Standarddialog zur Bildauswahl.
Bild	Zeigt eine verkleinerte Ansicht des Bilds.

7.13.2 Erstellen einer Bildersammlung

Anlegen einer Bildersammlung

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum das SPS-Projektobjekt oder einen Ordner einen Ordner. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Bildersammlung**.
   ⇨ Es öffnet sich der Dialog **Bildersammlung hinzufügen**.
2. Geben Sie einen Namen für die Bildersammlung ein (zum Beispiel „Images1“) und bestätigen Sie den Dialog mit der Schaltfläche **Öffnen**.
   ⇨ Die Bildersammlung wird im SPS-Projektbaum hinzugefügt und in einem Editor geöffnet.
3. Führen Sie einen Doppelklick auf das Feld **ID** aus und vergeben Sie eine geeignete ID (zum Beispiel „Icon1“).
4. Führen Sie einen Doppelklick auf die Spalte **Dateiname** aus. Klicken Sie auf die Schaltfläche … .
   ⇨ Der Standarddialog **Öffnen** öffnet sich.
5. Wählen Sie die Bilddatei aus.
6. Geben Sie in der Spalte **ID** eine Identifikationsbezeichnung ein. Wenn Sie keine ID eingeben, wird automatisch der Dateiname eingefügt.

⇨ Die Bilddatei wird in verkleinerter Darstellung in der Spalte **Bild** angezeigt. Der Pfad wird im Feld **Dateiname** und der Name der Datei bzw. die von Ihnen gewählte Identifikationsbezeichnung wird im Feld **ID** angezeigt. Die Bilddatei kann nun über den Namen „Images1.Icon1“ referenziert werden.

Verwenden einer Bilddatei im Visualisierungselement Bild

Wenn Sie ein Bildelement in eine Visualisierung einfügen, können Sie den Typ des Bilds festlegen:

- Statisches Bild: Geben Sie in der Konfiguration des Elements (Eigenschaft **Statische ID**) die ID der Bilddatei oder den Namen der Bildersammlung + ID ein. Beachten Sie dabei die Bemerkungen zu Suchreihenfolge und Zugriff.
- Dynamisches Bild: Geben Sie in der Konfiguration des Elements (Eigenschaft **Bitmap ID variable**) die Variable ein, die die ID der Bilddatei definiert, zum Beispiel MAIN.imagevar. Sie können ein dynamisches Element im Onlinebetrieb in Abhängigkeit von einer Variable austauschen

Verwenden einer Bilddatei für den Visualisierungshintergrund

In der Hintergrunddefinition einer Visualisierung können Sie ein Bild angeben. Sie können das Bild, wie oben für ein Visualisierungselement beschrieben, durch den Namen der Bildersammlung plus den Dateinamen definieren.

7.14 Syntax prüfen und Code analysieren

TwinCAT bietet Ihnen nützliche Funktionen, um Sie bei der Programmerstellung zu unterstützen und Fehler festzustellen. Durch die Syntax-Prüfung werden bereits während der Programmeingabe Fehler markiert und entsprechende Meldungen ausgegeben.

Die statische Codeanalyse in TwinCAT hilft Ihnen zusätzlich festgelegte Kodierrichtlinien einzuhalten sowie fehleranfällige Konstrukte zu erkennen.

7.14.1 Syntax prüfen

Während Sie den Code eingeben, führt die Vorkompilierung in TwinCAT bereits einige grundlegende Prüfungen durch. Dabei werden fehlerhafte Stellen im Editor rot unterkringelt.

Nach der Programmierung müssen Sie das SPS-Projekt übersetzen. Dies erfolgt mit dem Befehl **Erstellen** im Menü **Projekt** erstellen. Dabei prüft der Compiler das Programm und gibt die Fehler in der Ansicht **Fehlerliste** aus.

TwinCAT erzeugt den Programmcode aus dem im Entwicklungssystem geschriebenen Quellcode automatisch vor dem Download des SPS-Projekts auf die Steuerung. Dabei wird vor dem Erzeugen des Programmcodes eine Prüfung der Zuweisungen, der Datentypen und der Verfügbarkeit von Bibliotheken durchgeführt. Weiterhin werden beim Erzeugen des Programmcodes die Speicheradressen vergeben.

Sie können eine Prüfung auch explizit über den Befehl **Überprüfe alle Objekte** im Menü **Erstellen** ausführen. Dabei werden auch Fehler in Objekten gefunden, die im aktuellen SPS-Projekt nicht verwendet werden.

TwinCAT gibt alle Fehler und Warnungen in der Ansicht **Fehlerliste** aus. Durch einen Doppelklick auf die Fehlermeldung öffnen Sie die betroffene POU im Editor. Die fehlerhafte Stelle wird dabei markiert. Alternativ können Sie auch über das Kontextmenü der Fehlermeldung zu den fehlerhaften Stellen springen.

Beachten Sie hierzu auch die Einstellungen in den TwinCAT-Optionen in der Kategorie **Intelligentes Kodieren**.

Siehe auch:

- Dokumentation TC3 User Interface: Dialog Optionen - Intelligentes Codieren [▶ 1026]

7.14.2 Codeanalyse (Static Analysis)

Mit der „Statischen Codeanalyse“ prüft TwinCAT 3 PLC vor dem Laden auf das Zielsystem, ob der Quellcode eines Projekts festgelegten Kodierrichtlinien folgt.

Die lizenzfreie Variante der statischen Codeanalyse ist das „Static Analysis Light“. Die dort konfigurierten Prüfungen werden automatisch bei jeder Codeerzeugung durchgeführt. Weiterführende Informationen Sie im Abschnitt „Static Analysis Light [▶ 158]“.

Die lizenzpflichtige Variante der statischen Codeanalyse ist das „Static Analysis“, das verglichen mit der Light-Version über einen stark erweiterten Funktions- und Konfigurationsumfang verfügt. Die statische Codeanalyse kann manuell angestoßen oder automatisch mit der Codeerzeugung durchgeführt werden. Weiterführende Informationen zu dieser Erweiterung finden Sie in der Dokumentation „TE1200 | TC3 PLC Static Analysis“.

Static Analysis Light vs. Static Analysis Full

Nachfolgend finden Sie eine Übersicht über den unterschiedlichen Funktionsumfang der lizenzfreien und der lizenzpflichtigen Variante vom Static Analysis.

Funktionsaspekt	Static Analysis Light (ohne TE1200-Lizenz)	Static Analysis Full (mit TE1200-Lizenz)
Lizenz erforderlich	Nein, kostenfrei nutzbar	Ja, TE1200-Lizenz erforderlich
(Regel-) Konfiguration speichern/ exportieren und laden/importieren	Nicht möglich, gekoppelt an SPS-Projekteigenschaften	Möglich (mit Hilfe der Schaltflächen **Laden/ Speichern** in den Einstellungen)
Kopplung der Ausführung an den Übersetzungsprozess	Ja, nicht konfigurierbar	Konfigurierbar (mit Hilfe der Option **Statische Analyse automatisch durchführen** in den Einstellungen; Manuelle Ausführung mit Hilfe des Befehls Befehl 'Statische Analyse durchführen')
Überprüfung von nicht genutzten Objekten (z.B. innerhalb eines Bibliotheksprojekts)	Nicht möglich	Möglich (mit Hilfe des Befehls Befehl 'Statische Analyse durchführen [Überprüfe alle Objekte]')
Maximale Anzahl an berichteten Fehlern	500 (nicht konfigurierbar) (Weiterführende Informationen zu der Bedeutung von 500 als maximale Fehleranzahl finden Sie in den Einstellungen)	Konfigurierbar (mit Hilfe der Einstellung **Maximale Anzahl an Fehlern** in den Einstellungen)
Maximale Anzahl an berichteten Warnungen	Ausgabe von Warnungen nicht möglich (siehe folgende Zeile)	Konfigurierbar (mit Hilfe der Einstellung **Maximale Anzahl an Warnungen** in den Einstellungen)
Regeln: Aktivierungsmöglichkeiten	• Aktiv und Ausgabe als Fehler • Inaktiv	• Aktiv und Ausgabe als Fehler • Aktiv und Ausgabe als Warnung • Inaktiv
Regeln: Umfang	7 Kodierregeln • SA0033: Nicht verwendete Variablen • SA0028: Überlappende Speicherbereiche • SA0006: Schreibzugriff auf mehrere Tasks • SA0004: Mehrfacher Schreibzugriff auf Ausgang • SA0027: Mehrfachverwendung des Namens • SA0167: Temporäre Funktionsbausteininstanzen • SA0175: Verdächtige Operation auf String	Mehr als 100 Kodierregeln
Regeln: Precompile-Unterschlängelung, QuickFix	Nicht verfügbar	Verfügbar
Namenskonventionen	Nicht verfügbar	Verfügbar
Metriken	Nicht verfügbar	Verfügbar
Unzulässige Symbole	Nicht verfügbar	Verfügbar

Pragmas und Attribute zur temporären Deaktivierung von Regeln	Ja, im Light-Umfang verfügbar: • Pragma {analysis ...} • Attribut {attribute 'no-analysis'} • Attribut {attribute 'analysis' := '...'}	Ja, im Full-Umfang verfügbar: • Pragma {analysis ...} • Attribut {attribute 'no-analysis'} • Attribut {attribute 'analysis' := '...'} • Attribut {attribute 'naming' := '...'} • Attribut {attribute 'nameprefix' := '...'} • Attribut {attribute 'analysis:report-multiple-instance-calls'}

7.14.2.1 Static Analysis Light

Mit der „Statischen Codeanalyse“ prüft TwinCAT 3 PLC vor dem Laden auf das Zielsystem, ob der Quellcode eines Projekts festgelegten Kodierrichtlinien folgt.

Die lizenzfreie Variante der statischen Codeanalyse ist das „Static Analysis Light“. Die dort konfigurierten Prüfungen werden automatisch im Anschluss an jede erfolgreiche Codeerzeugung durchgeführt. Sie definieren den gewünschten Satz an Regeln in den SPS-Projekteigenschaften in der Kategorie **Static Analysis**.

Abweichungen von den Regeln werden als Fehlermeldungen in der **Fehlerliste** ausgegeben. Jede Regel besitzt eine eindeutige Nummer. Wenn während der Statischen Analyse die Verletzung einer Regel festgestellt wird, wird die Nummer zusammen mit einer Fehlerbeschreibung gemäß folgender Syntax in der Fehlerliste ausgegeben. Die Abkürzung „SA“ weist dabei auf „Static Analysis“ hin.

Syntax: "SA<Regelnummer>: <Regelbeschreibung>"

Beispiel für Regelnummer 33 (Nicht verwendete Variablen): "SA0033: Nicht verwendet: Variable 'bSample'"

TwinCAT analysiert nur den Programmcode des aktuellen Projekts, Bibliotheken bleiben unbeachtet!

Beachten Sie, dass das Static Analysis Light automatisch im Anschluss an den erfolgreichen Übersetzungsprozess ausgeführt wird. Falls die Codegenerierung hingegen nicht erfolgreich war, d.h. wenn der Compiler Kompilierfehler festgestellt hat, wird das Static Analysis Light nicht ausgeführt.

Mithilfe von Pragmas können Sie Prüfungen für bestimmte Codeteile ausschalten (siehe unten).

Konfiguration des Regelsatzes

Die Kategorie **Static Analysis** der SPS-Projekteigenschaften definiert die Prüfungen, die die Light-Version der statischen Codeanalyse bei jeder Codeerzeugung durchführt.

Geltungsbereich der Static-Analysis-Konfiguration

Die Einstellungen, die Sie in der Kategorie **Static Analysis** der SPS-Projekteigenschaften vornehmen, sind sogenannte **Solution options** und wirken sich daher nicht nur auf das SPS-Projekt aus, dessen Eigenschaften Sie aktuell bearbeiten. Der konfigurierte Regelsatz wird für alle SPS-Projekte übernommen, die sich in der Entwicklungsumgebung befinden.

Verfügbare Regeln innerhalb des Static Analysis Light:

- SA0033: Nicht verwendete Variablen
- SA0028: Überlappende Speicherbereiche

- SA0006: Schreibzugriff aus mehreren Tasks
- SA0004: Mehrfacher Schreibzugriff auf Ausgang
- SA0027: Mehrfachverwendung des Namens
- SA0167: Temporäre Funktionsbausteininstanzen
- SA0175: Verdächtige Operation auf String

SA0033: Nicht verwendete Variablen

Funktion	Ermittelt Variablen, die deklariert sind, aber innerhalb des kompilierten Programmcodes nicht verwendet werden.
Begründung	Nicht verwendete Variablen machen ein Programm weniger gut lesbar und wartbar. Nicht verwendete Variablen belegen unnötig Speicher und kosten bei der Initialisierung unnötig Laufzeit.
Wichtigkeit	Mittel
PLCopen-Regel	CP22/CP24

SA0028: Überlappende Speicherbereiche

Funktion	Ermittelt die Stellen, durch die zwei oder mehr Variablen denselben Speicherplatz belegen.
Begründung	Wenn zwei Variablen auf dem gleichen Speicherplatz liegen, dann kann sich der Code sehr unerwartet verhalten. Dies ist in jedem Fall zu vermeiden. Wenn es unumgänglich ist, einen Wert in verschiedenen Interpretationen zu verwenden, zum Beispiel einmal als DINT und einmal als REAL, dann sollten Sie eine UNION definieren. Auch über einen Pointer können Sie auf einen Wert anders getypt zugreifen, ohne dass der Wert umgewandelt wird.
Wichtigkeit	Hoch

Beispiel:

In dem folgenden Beispiel verwenden beide Variablen Byte 21, d.h. die Speicherbereiche der Variablen überlappen.

```
PROGRAM MAIN
VAR
    nVar1 AT%QB21  : INT;        // => SA0028
    nVar2 AT%QD5   : DWORD;      // => SA0028
END_VAR
```

SA0006: Schreibzugriff aus mehreren Tasks

Funktion	Ermittelt Variablen, auf die von mehr als einer Task geschrieben wird.
Begründung	Eine Variable, die in mehreren Tasks geschrieben wird, kann unter Umständen ihren Wert unerwartet ändern. Das kann zu verwirrenden Situationen führen. Stringvariablen und auf einigen 32-Bit-Systemen auch 64-Bit-Integer-Variablen können sogar einen inkonsistenten Zustand bekommen, wenn die Variable gleichzeitig in zwei Tasks geschrieben wird.
Ausnahme	In bestimmten Fällen kann es nötig sein, dass mehrere Tasks eine Variable schreiben. Stellen Sie dann sicher, beispielsweise durch die Verwendung von Semaphoren, dass der Zugriff nicht zu einem inkonsistenten Zustand führt.
Wichtigkeit	Hoch
PLCopen-Regel	CP10

Sehen Sie auch die Regel SA0103.

Aufruf entspricht Schreibzugriff

Bitte beachten Sie, dass Aufrufe als Schreibzugriff interpretiert werden. Beispielsweise wird der Aufruf einer Methode für eine Funktionsbausteininstanz als Schreibzugriff auf die Funktionsbausteininstanz angesehen. Eine genauere Analyse der Zugriffe und Aufrufe ist z.B. aufgrund von virtuellen Aufrufen (Zeiger, Interface) nicht möglich.

Wenn Sie die Regel SA0006 für eine Variable (z.B. für eine Funktionsbausteininstanz) deaktivieren möchten, können Sie das folgende Attribut oberhalb der Variablendeklaration einfügen: {attribute 'analysis' := '-6'}

Beispiele:

Die beiden globalen Variablen nVar und bVar werden von zwei Tasks geschrieben.

Globale Variablenliste:

```
VAR_GLOBAL
    nVar  : INT;
    bVar  : BOOL;
END_VAR
```

Programm MAIN_Fast, aufgerufen von der Task PlcTaskFast:

```
nVar := nVar + 1;                // => SA0006
bVar := (nVar > 10);             // => SA0006
```

Programm MAIN_Slow, aufgerufen von der Task PlcTaskSlow:

```
nVar := nVar + 2;                // => SA0006
bVar := (nVar < -50);            // => SA0006
```

SA0004: Mehrfacher Schreibzugriff auf Ausgang

Funktion	Ermittelt Ausgänge, die an mehr als einer Position geschrieben werden.
Begründung	Die Wartbarkeit leidet, wenn ein Ausgang an verschiedenen Stellen im Code geschrieben wird. Es ist dann unklar, welcher Schreibzugriff derjenige ist, der tatsächlich Auswirkungen im Prozess hat. Gute Praxis ist es, die Berechnung der Ausgangsvariablen in Hilfsvariablen durchzuführen und an einer Stelle am Ende des Zyklus den berechneten Wert zuzuweisen.
Ausnahme	Es wird kein Fehler ausgegeben, wenn eine Ausgangsvariable in verschiedenen Zweigen von IF- bzw. CASE-Anweisungen geschrieben wird.
Wichtigkeit	Hoch
PLCopen-Regel	CP12

Diese Regel kann **nicht** über ein Pragma oder Attribut abgeschaltet werden!
Weitere Informationen zu Attributen finden Sie unter Pragmas und Attribute.

Beispiel:

Globale Variablenliste:

```
VAR_GLOBAL
    bVar     AT%QX0.0 : BOOL;
    nSample  AT%QW5   : INT;
END_VAR
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    nCondition        : INT;
END_VAR
```

```
IF nCondition < INT#0 THEN
    bVar    := TRUE;              // => SA0004
    nSample := INT#12;            // => SA0004
END_IF

CASE nCondition OF
    INT#1:
        bVar := FALSE;            // => SA0004

    INT#2:
        nSample := INT#11;        // => SA0004

ELSE
     bVar    := TRUE;             // => SA0004
     nSample := INT#9;            // => SA0004
END_CASE
```

SA0027: Mehrfachverwendung des Namens

Funktion	Ermittelt die Mehrfachverwendung eines Namens/Bezeichners einer Variable oder eines Objekts (POU) innerhalb des Gültigkeitsbereichs eines Projekts. Die folgenden Fälle werden abgedeckt: • Der Name einer Enumerationskonstanten ist identisch mit dem Namen in einer anderen Enumeration innerhalb der Applikation oder in einer eingebundenen Bibliothek. • Der Name einer Variablen ist identisch mit dem Namen eines anderen Objekts in der Applikation oder in einer eingebundenen Bibliothek. • Der Name einer Variablen ist identisch mit dem Namen einer Enumerationskonstanten in einer Enumeration in der Applikation oder in einer eingebundenen Bibliothek. • Der Name eines Objekts ist identisch mit dem Namen eines anderen Objekts in der Applikation oder in einer eingebundenen Bibliothek.
Begründung	Gleiche Namen können beim Lesen des Codes verwirrend sein. Sie können zu Fehlern führen, wenn unbeabsichtigt auf das falsche Objekt zugegriffen wird. Definieren und befolgen Sie deshalb Namenskonventionen zur Vermeidung solcher Situationen.
Ausnahme	Enumerationen, die mit dem Attribut 'qualified_only' deklariert sind, sind von der SA0027-Prüfung ausgenommen, da auf ihre Elemente nur qualifiziert zugegriffen werden kann.
Wichtigkeit	Mittel

Beispiel:

Das folgende Beispiel erzeugt Fehler/Warnung SA0027, da die Bibliothek Tc2_Standard im Projekt eingebunden ist, welche den Funktionsbaustein TON zur Verfügung stellt.

```
PROGRAM MAIN
VAR
    ton  : INT;                   // => SA0027
END_VAR
```

SA0167: Temporäre Funktionsbausteininstanzen

Funktion	Ermittelt Funktionsbausteininstanzen, die als temporäre Variable deklariert sind. Dies betrifft Instanzen, die in einer Methode oder in einer Funktion oder als VAR_TEMP deklariert sind, und die deshalb in jedem Abarbeitungszyklus bzw. bei jedem Bausteinaufruf neu initialisiert werden.
Begründung	Funktionsbausteine haben einen Zustand, der meist über mehrere SPS-Zyklen hinweg erhalten bleibt. Eine Instanz auf dem Stack existiert nur für die Dauer des Funktionsaufrufs. Es ist daher nur selten sinnvoll, eine Instanz als temporäre Variable anzulegen. Zweitens sind Funktionsbausteininstanzen häufig groß und verbrauchen sehr viel Platz auf dem Stack (der auf Steuerungen meist begrenzt ist). Drittens kann die Initialisierung und häufig auch die Terminierung eines Funktionsbausteins ziemlich viel Zeit in Anspruch nehmen.
Wichtigkeit	Mittel

Beispiele:

Methode FB_Sample.SampleMethod:

```
METHOD SampleMethod : INT
VAR_INPUT
END_VAR
VAR
    fbTrigger : R_TRIG;          // => SA0167
END_VAR
```

Funktion F_Sample:

```
FUNCTION F_Sample : INT
VAR_INPUT
END_VAR
VAR
    fbSample  : FB_Sample;       // => SA0167
END_VAR
```

Programm MAIN:

```
PROGRAM MAIN
VAR_TEMP
    fbSample  : FB_Sample;       // => SA0167
    nReturn   : INT;
END_VAR
```

```
nReturn := F_Sample();
```

SA0175: Verdächtige Operation auf String

Funktion	Ermittelt Codestellen, die bei einer UTF-8-Kodierung verdächtig sind.

Erfasste Konstrukte	1. Indexzugriff auf einen Single-Byte-String
	◦ Beispiel: `sVar[2]`
	◦ Meldung: Verdächtige Operation auf String: Indexzugriff '<expression>'
	2. Adresszugriff auf einen Single-Byte-String
	◦ Beispiel: `ADR(sVar)`
	◦ Meldung: Verdächtige Operation auf String: Möglicher Indexzugriff '<expression>'
	3. Aufruf einer String-Funktion der Tc2_Standard-Bibliothek außer CONCAT und LEN
	◦ Beispiel: `FIND(sVar, 'a');`
	◦ Meldung: Verdächtige Operation auf String: Möglicher Indexzugriff '<expression>'
	4. Einzelnes Byte-Literal, das Nicht-ASCII-Zeichen enthält
	◦ Beispiele:
	`sVar := '99€';`
	`sVar := 'Ä';`
	◦ Meldung: Verdächtige Operation auf String: Literal '<literal>' enthält Nicht-ASCII-Zeichen

Wichtigkeit	Mittel

Beispiele:

```
VAR
    sVar  : STRING;
    pVar  : POINTER TO STRING;
    nVar  : INT;
END_VAR
```

```
// 1) SA0175: Suspicious operation on string: Index access
sVar[2];                        // => SA0175

// 2) SA0175: Suspicious operation on string: Possible index access
pVar := ADR(sVar);              // => SA0175

// 3) SA0175: Suspicious operation on string: Possible index access
nVar := FIND(sVar, 'a');        // => SA0175

// 4) SA0175: Suspicious operation on string: Literal '<...>' contains Non-ASCII character
sVar := '99€';                  // => SA0175
sVar := 'Ä';                    // => SA0175
```

Pragmas und Attribute für Static Analysis Light

Mithilfe eines Pragmas bzw. eines Attributs können Sie Codeteile aus der Prüfung ausklammern. Verwenden Sie das Pragma {analysis ...} [▶ 163], um Kodierregeln im Implementierungsteil auszuschalten und das Attribut {attribute 'analysis' := '...'} [▶ 164], um Kodierregeln im Deklarationsteil auszuschalten.

Voraussetzung: Sie haben die Regeln in den Projekteigenschaften aktiviert.

Außerdem können Sie das Attribut {attribute 'no-analysis'} [▶ 164] verwenden, um Programmierobjekte von der Statischen Analyse auszuschließen.

Regeln, die in den Projekteigenschaften deaktiviert sind, können Sie auch nicht über Pragma oder Attribut aktivieren.

Regel SA0004 (Mehrfacher Schreibzugriff auf Ausgang) kann nicht über Pragma deaktiviert werden.

Pragma {analysis ...}

Das Pragma {analysis -/+<Regelnummer>} können Sie im Implementierungsteil eines Programmierbausteins verwenden, um einzelne Kodierregeln für die nachfolgenden Codezeilen auszuschalten. Sie deaktivieren Codierregeln durch die Angabe der Regelnummern und einem vorangestellten Minuszeichen (“-”). Zur Aktivierung wird ein Pluszeichen (“+”) vorangestellt. Mit Hilfe einer Kommaseparierung können Sie im Pragma beliebig viele Regeln angeben.

Einfügeort:

- Deaktivierung von Regeln: Im Implementierungsteil vor der ersten Codezeile, ab der die Codeanalyse deaktiviert wird, mit {analysis - ...}.
- Aktivierung von Regeln: Nach der letzten Zeile der Deaktivierung mit {analysis + ...}.
- Für die Regel SA0164 kann das Pragma auch im Deklarationsteil vor einem Kommentar eingefügt werden.

Syntax:

- Deaktivierung von Regeln:
  - eine Regel: {analysis -<Regelnummer>}
  - mehrere Regeln: {analysis -<Regelnummer>, -<weitere Regelnummer>, -<weitere Regelnummer>}
- Aktivierung von Regeln:
  - eine Regel: {analysis +<Regelnummer>}
  - mehrere Regeln: {analysis +<Regelnummer>, +<weitere Regelnummer>, +<weitere Regelnummer>}

Beispiele:

Sie möchten Regel 24 (nur getypte Literale erlaubt) für eine Zeile deaktivieren (d.h. es ist in diesen Zeilen nicht nötig, "nTest := DINT#99" zu schreiben) und danach wieder aktivieren:

```
{analysis -24}
nTest := 99;
{analysis +24}
nVar := INT#2;
```

Angabe mehrerer Regeln:

```
{analysis -10, -24, -18}
```

Attribut {attribute 'analysis' := '...'}

Das Attribut {attribute 'analysis' := '-<Regelnummer>'} können Sie verwenden, um bestimmte Regeln für einzelne Deklarationen oder für ein ganzes Programmierobjekt abzuschalten. Sie deaktivieren die Kodierregel durch die Angabe der Regelnnummer(n) und einem vorangestellten Minuszeichen. Sie können im Attribut beliebig viele Regeln angeben.

Einfügeort:

oberhalb der Deklaration eines Programmierobjekts oder in der Zeile oberhalb einer Variablendeklaration

Syntax:

- eine Regel: {attribute 'analysis' := '-<Regelnummer>'}
- mehrere Regeln: {attribute 'analysis' := '-<Regelnummer>, -<weitere Regelnummer>, -<weitere Regelnummer>'}

Beispiele:

Sie möchten Regel 33 (Nicht verwendete Variablen) für alle Variablen der Struktur ausschalten.

```
{attribute 'analysis' := '-33'}
TYPE ST_Sample :
STRUCT
    bMember  : BOOL;
    nMember  : INT;
END_STRUCT
END_TYPE
```

Sie möchten die Prüfung von Regel 28 (Überlappende Speicherbereiche) und von Regel 33 (Nicht verwendete Variablen) für die Variable nVar1 ausschalten.

```
PROGRAM MAIN
VAR
    {attribute 'analysis' := '-28, -33'}
    nVar1 AT%QB21  : INT;
    nVar2 AT%QD5   : DWORD;

    nVar3 AT%QB41  : INT;
    nVar4 AT%QD10  : DWORD;
END_VAR
```

Sie möchten Regel 6 (Gleichzeitiger Zugriff) für eine globale Variable ausschalten, sodass keine Fehlermeldung generiert wird, wenn die Variable von mehr als einer Task geschrieben wird.

```
VAR_GLOBAL
    {attribute 'analysis' := '-6'}
    nVar  : INT;
    bVar  : BOOL;
END_VAR
```

Attribut {attribute 'no-analysis'}

Das Attribut {attribute 'no-analysis'} können Sie verwenden, um ein gesamtes Programmierobjekt von der Prüfung durch die Statische Analyse auszuschließen. Für dieses Programmierobjekt wird die Prüfung der Kodierregeln, der Namenskonventionen und der unzulässigen Symbole nicht durchgeführt.

Einfügeort:

oberhalb der Deklaration eines Programmierobjekts

Syntax:

{attribute 'no-analysis'}

Beispiele:

```
{attribute 'qualified_only'}
{attribute 'no-analysis'}
VAR_GLOBAL
    …
END_VAR
```

```
{attribute 'no-analysis'}
PROGRAM MAIN
VAR
    …
END_VAR
```

7.15 Orientieren und Navigieren

7.15.1 Verwendungsstellen mit der Querverweisliste finden

Sie können sich die Verwendungsstellen einer Variablen oder einer POU (Programm, Funktionsbaustein, Funktion) in einer so genannten **Querverweisliste**, ausgeben lassen und von dieser Liste aus an die jeweiligen Stellen im Projekt springen.

✓ Es ist eine POU im Editor geöffnet.

1. Setzen Sie den Cursor auf eine Variable oder den Namen einer POU in der Deklaration oder in der Implementierung.
2. Wählen Sie im Menü **PLC > Fenster** den Befehl **Querverweisliste** oder im Kontextmenü des Editors den Befehl **Alle Verweise suchen**.
   ⇨ Die Ansicht **Querverweisliste** öffnet sich und zeigt die Verwendungsstellen der Variablen oder der POU. Immer wird dabei die Deklarationsstelle und darunter eingerückt die Verwendungsstellen im Projekt dargestellt. Wenn Sie eine strukturierte Variable oder einen Funktionsbausteinnamen suchen, werden die Verwendungsstellen der Members oder die der Funktionsbausteininstanzen eingerückt unterhalb der Deklarationsstelle dargestellt.
3. Doppelklicken Sie auf eine Verwendungsstelle der Querverweisliste.

⇨ Das entsprechende Objekt wird im Editor geöffnet und die Verwendungsstelle wird markiert.

Wenn die Ansicht **Querverweisliste** geöffnet ist, können Sie außerdem auch folgendermaßen eine Variable, eine POU oder eine DUT angeben, um ihre Verwendungsstellen zu suchen:

- Aktivieren Sie im Menü **Extras > Optionen** in der Kategorie **TwinCAT > SPS Programmierumgebung > Intelligentes Kodieren** die Option **Querverweise automatisch bei Selektionsänderung aktualisieren**. Stellen Sie danach den Cursor in den Namen der Variablen/der POU/des DUT im Editorfenster der POU.
- Geben Sie manuell den Variablennamen in das Feld **Name** ein.

Sie können die Platzhalter „*“ (beliebig viele Zeichen) oder „?“ (genau ein beliebiges Zeichen) in Kombination mit einer Teil-Zeichenkette eines Variablenbezeichners verwenden.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Verweise suchen [▶ 921]
- Dokumentation TC3 User Interface: Befehl Querverweisliste [▶ 984]
- Dokumentation TC3 User Interface: Dialog Optionen - Intelligentes Codieren [▶ 1026]

7.15.2 Deklaration finden

TwinCAT bietet die Möglichkeit, das gesamte Projekt nach der Definitionsstelle einer Variablen oder Funktion zu durchsuchen. Dabei wird der Baustein, der die Definition enthält, im Editor geöffnet und die Deklaration markiert.

Finden der Deklaration einer Variablen

✓ Es ist eine POU im Editor geöffnet.

1. Setzen Sie den Cursor auf einen Bezeichner in der Implementierung.
2. Wählen Sie im Kontextmenü des Editors den Befehl **Gehe zur Definition**.

⇨ Die POU mit der Deklaration wird im Editor geöffnet und die Definition der Variablen wird markiert. Wenn die Definition in einer „übersetzten“ Bibliothek liegt, wird der entsprechende Baustein im Bibliotheksverwalter geöffnet.

Sie können den Befehl im Offline- und Onlinebetrieb verwenden.

Beispiele:

Der folgende Baustein enthält eine Funktionsbaustein-Definition (fbInst), einen Programmaufruf (SampleProg()) und einen Funktionsbaustein-Aufruf (fbInst.nOut):

```
VAR
    fbInst : FB_Sample;
    nVar : INT;
    nRes : INT;
END_VAR

SampleProg();
nVar := SampleProg.nVar1;
nRes := fbInst.nOut;
```

Wenn Sie den Cursor auf SampleProg stellen, öffnet der Befehl das Programm SampleProg in seinem Editor.

Wenn Sie den Cursor auf fbInst stellen, setzt der Befehl den Fokus ins Deklarationsfenster in Zeile `fbInst : FBSample;`

Wenn Sie den Cursor auf nOut stellen, öffnet der Befehl den Funktionsbaustein FB_Sample in seinem Editor.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Gehe zur Definition [▶ 920]

7.15.3 Lesezeichen setzen und verwenden

Sie können Lesezeichen verwenden, um das Navigieren in langen Programmen zu erleichtern. Lesezeichen können in allen Editoren der Programmiersprachen außer AS (Ablaufsprache) verwendet werden. Über Befehle können Sie direkt zu den markierten Programmstellen navigieren.

Die Lesezeichen werden standardmäßig in der Benutzeroptionsdatei des Visual-Studio-Projekts (.suo) abgelegt. Diese Datei ist nutzerspezifisch und damit nicht kompatibel mit einem Source-Control-Managementsystem. Demnach sind die Lesezeichen nur für das lokale Projekt gültig.

Um die Lesezeichen nutzerübergreifend zu sichern, gibt es die Möglichkeit, diese zusätzlich in einer separaten Datei (Write Bookmarks to File [▶ 961]) zu sichern, welche dann Bestandteil der TwinCAT-Archivmöglichkeiten ist. Auch diese Datei ist nicht kompatibel mit einem Source-Control-Managementsystem.

Lesezeichen setzen/löschen

✓ Ein Programmierbaustein ist im Editor geöffnet.

1. Setzen Sie den Cursor an eine beliebige Programmzeile.

2. Wählen Sie im Menü **SPS > SPS Lesezeichen** den Befehl **Lesezeichen ein-/ausschalten**.

   ⇨ An dieser Programmstelle wird ein Lesezeichen gesetzt. Dies wird durch das Lesezeichensymbol gekennzeichnet.

3. Setzen Sie mehrere Lesezeichen an verschiedenen Programmstellen.
4. Um ein Lesezeichen zu löschen, setzen Sie den Cursor an eine Programmzeile mit Lesezeichen.
5. Wählen Sie im Menü **SPS > SPS Lesezeichen** den Befehl **Lesezeichen ein-/ausschalten**.

⇨ Das Lesezeichen wird wieder entfernt. Das Lesezeichensymbol wird gelöscht.
Alternativ dazu können Sie im Fenster **SPS Lesezeichen** ein oder mehrere Lesezeichen über die Schaltfläche löschen. Hierfür müssen die entsprechenden Lesezeichen selektiert sein.

Um alle Lesezeichen des aktiven Programmierbausteins zu entfernen, wählen Sie im Menü **SPS > SPS Lesezeichen** den Befehl **Alle Lesezeichen löschen (aktiver Editor)**.

Um alle Lesezeichen eines Projekts zu löschen, wählen Sie im Menü **SPS > SPS Lesezeichen** den Befehl **Alle Lesezeichen löschen.**

Siehe auch:

- Befehl Lesezeichen ein-/ausschalten [▶ 997]
- Befehl Alle Lesezeichen löschen (aktiver Editor) [▶ 999]
- Befehl Alle Lesezeichen löschen [▶ 998]

Innerhalb eines Programmierbausteins zu Lesezeichen springen

✓ Ein Programmierbaustein ist im Editor geöffnet. Mehrere Lesezeichen sind gesetzt.

1. Wählen Sie im Menü **SPS > SPS Lesezeichen** den Befehl **Nächstes Lesezeichen (aktiver Editor)**.

   ⇨ Abhängig von der aktuellen Cursorposition springt der Cursor nach unten zum nächsten Lesezeichen.

2. Wählen Sie im Menü **SPS > SPS Lesezeichen** den Befehl **Vorheriges Lesezeichen (aktiver Editor)**.

⇨ Abhängig von der aktuellen Cursor-Position springt der Cursor nach oben zum vorherigen Lesezeichen.

Siehe auch:

- Befehl Nächstes Lesezeichen (aktiver Editor) [▶ 998]
- Befehl Vorheriges Lesezeichen (aktiver Editor) [▶ 998]

Zu Lesezeichen verschiedener Programmierbausteine eines Projekts springen

✓ Ein Projekt mit mehreren Programmierbausteinen ist geöffnet. Mehrere Lesezeichen sind in verschiedenen Programmierbausteinen gesetzt.

1. Wählen Sie im Menü **SPS > Fenster** den Befehl **SPS Lesezeichen**

   ⇨ Die Ansicht **Lesezeichen** öffnet sich.
   Alle Lesezeichen des Projekts werden in der Ansicht tabellarisch aufgelistet.

2. Klicken Sie auf die Schaltfläche **Nächstes Lesezeichen** .

   ⇨ In der Ansicht **Lesezeichen** wird das Lesezeichen in der Zeile unterhalb des aktuell selektierten Lesezeichens selektiert.
   Der Programmierbaustein mit dem neu selektierten Lesezeichen der Tabelle öffnet sich im Editor und die Zeile mit dem Lesezeichen ist selektiert.

3. Schritt 2 entsprechend können Sie über die Schaltfläche **Vorheriges Lesezeichen** zu dem Lesezeichen des Projekts springen, das in der Ansicht **Lesezeichen** in der Zeile darüber angezeigt wird.

Siehe auch:

- Befehl SPS Lesezeichen [▶ 994]

- Befehl Nächstes Lesezeichen [▶ 997]
- Befehl Vorheriges Lesezeichen [▶ 998]

7.16 Projektweites Suchen und Ersetzen

In TwinCAT können Sie Zeichenketten in einzelnen Objekten oder im ganzen Projekt suchen und wenn gewünscht durch eine andere Zeichenkette ersetzen.

1. Aktivieren Sie im Menü **Bearbeiten > Suchen und Ersetzen** den Befehl **Schnellsuche (In Dateien suchen)**.
   ⇨ Der Dialog **Suchen und Ersetzen** öffnet sich.
2. Geben Sie im Feld **Suchen nach** die Zeichenkette ein, nach der Sie suchen wollen.
3. Legen Sie in der Auswahlliste **Suchen in** fest, in welchen Objekten gesucht werden soll.
4. Wählen Sie die **Suchoptionen** aus.
5. Wählen Sie die **Ergebnisoptionen** aus.
6. Klicken Sie auf die Schaltfläche **Weitersuchen**.
   ⇨ Der erste Treffer wird im Editor angezeigt.
7. Klicken Sie auf **Alle suchen**, wenn Sie eine Übersicht über alle Treffer erhalten wollen.
   ⇨ Die Ansicht **Sucherergebnisse** mit einer Auflistung der Treffer öffnet sich.
8. Klicken Sie auf die Schaltfläche **In Dateien ersetzen**, wenn Sie die gesuchte Zeichenkette durch einen andere ersetzen wollen.
9. Geben Sie im Feld **Ersetzen durch** die Zeichenkette ein, welche die ursprüngliche Zeichenkette ersetzen soll.
10. Klicken Sie auf die Schalfläche **Ersetzen**.
11. Klicken Sie auf **Alle ersetzen**, wenn Sie alle Zeichenkette ersetzen wollen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Schnellersetzung (In Dateien ersetzen) [▶ 924]
- Dokumentation TC3 User Interface: Befehl Schnellsuche (In Dateien suchen) [▶ 923]

7.17 Refactoring

Im Allgemeinen ist Refactoring eine Methode, um bereits geschriebene Software im Design zu verbessern, ohne ihr Verhalten zu ändern.

Refactoring in TwinCAT bietet Funktionalitäten zum Umbenennen von Objekt- und Variablennamen und zum Aktualisieren von Bausteinanschlüssen. Sie können alle Verwendungsstellen umbenannter Objekte und Variablen anzeigen lassen und diese dann gesamt oder einzeln ausgewählt umbenennen. Zusätzlich können Sie in den TwinCAT-Optionen (**Extras > Optionen**) in der Kategorie **TwinCAT > SPS Programmierumgebung > Refactoring** konfigurieren, ob und an welchen Stellen TwinCAT Sie automatisch zum Refactoring auffordert.

Globale Variable umbenennen

Eine globale Variable projektweit umbenennen:

✓ Sie haben ein Projekt geöffnet, das mindestens einen Funktionsbaustein FB_Sample und eine globale Variablenliste GVL beinhaltet. Die globale Variablenliste ist in ihrem Editor geöffnet und enthält die Deklaration einer Variablen, beispielsweise nGlob1. FB_Sample verwendet nGlob1.

1. Selektieren Sie den Namen einer globalen Variablen, zum Beispiel „nGlob1“.
2. Wählen Sie im Kontextmenü den Befehl **Refactoring > 'nGlob1' umbenennen**.
3. Geben Sie im Dialog **Umbenennen** in das Eingabefeld **Neuer Name** einen neuen Namen ein, zum Beispiel „nGlobNeu“ und klicken Sie auf **OK**.

⇨ Der Dialog **Refactoring** öffnet sich. Im linken Fenster, im Projektbaum, sind die Objekte GVL und FB_Sample farblich rot gekennzeichnet und gelb hinterlegt. Im rechten Fenster ist FB_Sample in seinem Editor geöffnet und nGlob1 ist bereits in nGlobNeu umbenannt.

4. Klicken Sie auf **OK**.

⇨ In Ihrem Projekt ist die globale Variable nGlob1 überall in nGlobNeu umbenannt.

Eine globale Variable projektweit mit Ausnahme einer POU umbenennen:

1. Selektieren Sie den Namen einer globalen Variablen, zum Beispiel „nGlob1“.
2. Wählen Sie im Kontextmenü den Befehl **Refactoring > 'nGlob1' umbenennen**.
3. Geben Sie im Dialog **Umbenennen** in das Eingabefeld **Neuer Name** einen neuen Namen ein, zum Beispiel „nGlobNeu“ und klicken Sie auf **OK**.

   ⇨ Der Dialog **Refactoring** öffnet sich. Im linken Fenster, im Projektbaum, sind die Objekte GVL und FB_Sample farblich rot gekennzeichnet und gelb hinterlegt. Im rechten Fenster ist der Funktionsbaustein FB_Sample in seinem Editor geöffnet. Statt nGlob1 ist nGlobNeu aufgelistet.
4. Positionieren Sie den Cursor in das rechte Fenster und öffnen Sie das Kontextmenü.
5. Wählen Sie den Befehl **Dieses Objekt verwerfen** und klicken Sie auf **OK**.

⇨ In Ihrem Projekt ist die globale Variable nGlob1 in FB_Sample vorhanden. In den Objekten, in denen die Variable sonst vorkam, ist die Variable nGlobNeu angegeben. Im Meldungsfenster erscheint eine Fehlermeldung, dass der Bezeichner nGlob1 nicht definiert ist.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Refactoring - <Variable> umbenennen [▶ 926]

Eingangsvariablen hinzufügen und entfernen

Sie können im Deklarationsteil von Bausteinen über Refactoring-Befehle Eingangs- oder Ausgangsvariablen hinzufügen oder entfernen. An den Verwendungsstellen/Aufrufstellen der Bausteine aktualisiert TwinCAT entsprechend, wobei Sie dies pro Verwendungsstelle akzeptieren oder verwerfen können. Sie erhalten dazu den Dialog **Refactoring**.

✓ Sie haben eine Funktion F_Sample im Editor geöffnet. Die Funktion besitzt bereits Eingangsvariablen nInput1 und nInput2 und nInputx. Sie wird in den Programmen MAIN und POU aufgerufen.

1. Setzen Sie den Fokus in den Deklarationsteil der Funktion F_Sample.
2. Wählen Sie im Kontextmenü den Befehl **Refactoring > Variable hinzufügen**.

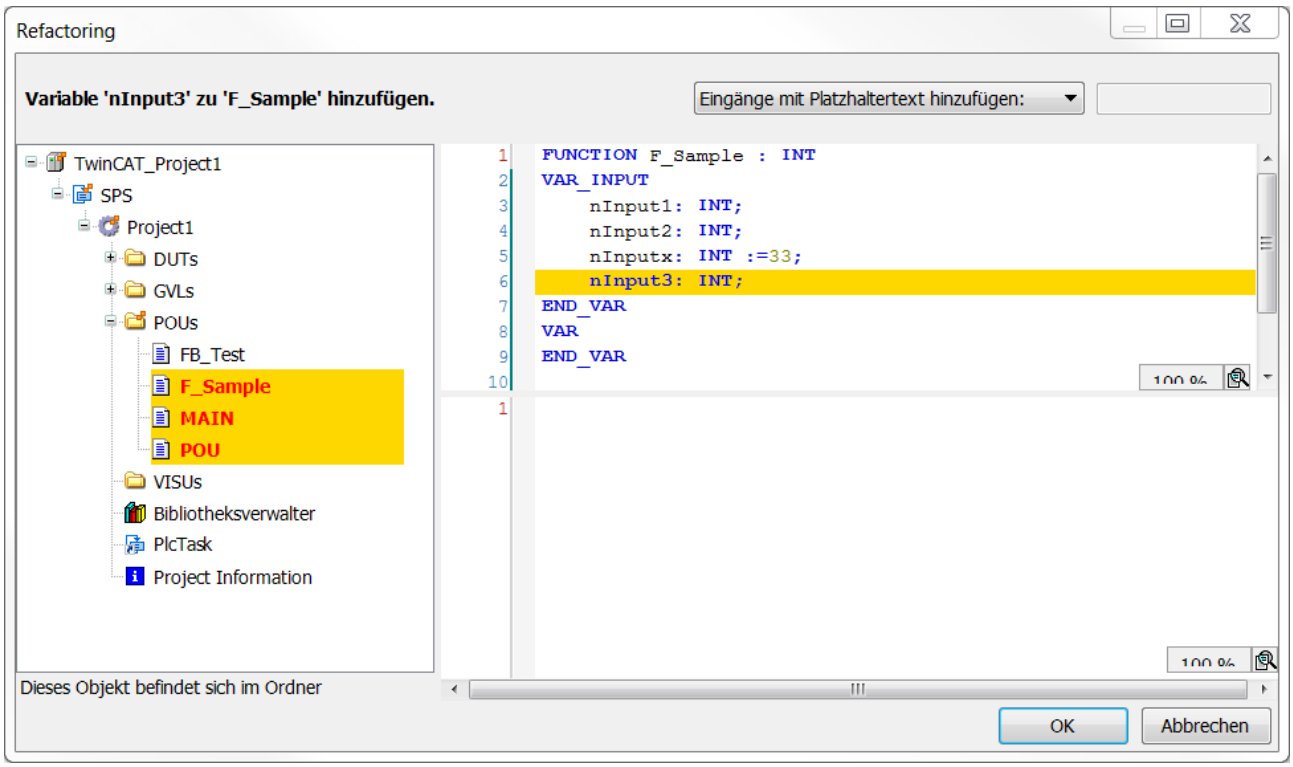
   ⇨ Der Standarddialog zum Deklarieren einer Variablen erscheint.
3. Deklarieren Sie Variable nInput3 mit Gültigkeitsbereich VAR_INPUT und Datentyp INT. Schließen Sie den Dialog mit **OK**.

   ⇨ Der Dialog **Refactoring** erscheint (siehe Abbildung unten). Die betroffenen Stellen sind gelb markiert.
4. Wählen Sie rechts oben die Option **Eingänge mit Platzhaltertext hinzufügen**.
5. Setzen Sie im linken Fenster den Cursor auf einen der gelb hinterlegten Objekte, beispielsweise MAIN. Wählen Sie im Kontextmenü den Befehl **Das gesamte Projekt annehmen**, um die neue Variable an den Verwendungsstellen von F_Sample im gesamten Projekt hinzuzufügen.

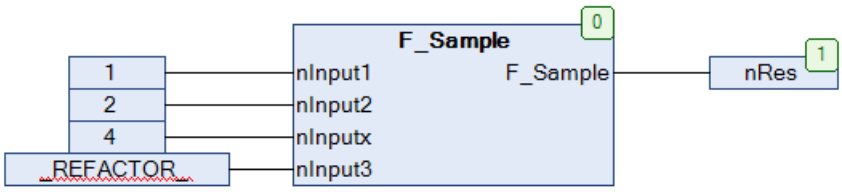
   ⇨ Im rechten Fenster sehen Sie die Änderung im Implementierungsteil von MAIN: Platzhalter _REFACTOR_ erscheint an der Stelle, an der die neue Variable eingefügt wurde.
6. Schließen Sie den Dialog **Refactoring** mit **OK**.

⇨ Wählen Sie Befehl **Bearbeiten > Suchen und Ersetzen**. Suchen Sie im Projekt nach „_REFACTOR_“, um die betroffenen Stellen zu überprüfen und entsprechend zu bearbeiten.

Alternativ können Sie die neue Variable direkt mit einem gewünschten Initialisierungswert einfügen, ohne zuerst mit einem Platzhalter zu arbeiten. In diesem Fall wählen Sie bei Schritt 4 die Option „Eingänge mit folgendem Wert hinzufügen“ und tragen den Wert rechts davon ein.



Beispiel für eine neue Variable mit Platzhaltertext in einem CFC-Baustein:



Beachten Sie auch die Möglichkeit, Variablen über Refactoring zu entfernen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Refactoring - <Variable> entfernen [▶ 929]
- Dokumentation TC3 User Interface: Befehl Refactoring - Variable hinzufügen [▶ 928]

Variablen in der Deklaration neu anordnen

Im Deklarationsteil von Bausteinen können Sie über Refactoring die Reihenfolge von Deklarationen verändern. Dies ist möglich bei Deklarationen der Gültigkeitsbereiche VAR_INPUT, VAR_OUTPUT oder VAR_IN_OUT.

```
VAR_INPUT
    nInVar2 : INT;
    nInVar1 : INT;
    in      : DUT;
    bVar    : BOOL;
    nInVar3 : INT;
END_VAR
```

✓ Sie haben den Deklarationsteil einer POU geöffnet, der beispielsweise die oben abgebildeten Deklarationen enthält.

1. Setzen Sie den Cursor in diesen Deklarationsblock und drücken die rechte Maustaste, um das Kontextmenü zu öffnen.
2. Wählen Sie im Kontextmenü den Befehl **Refactoring > Variablen neu ordnen**.
   ⇨ Der Dialog **Neu ordnen** erscheint mit einer Liste der VAR_INPUT-Variablen.
3. Selektieren Sie beispielsweise den Eintrag `nInVar1 : INT;` und ziehen ihn mit der Maus vor den Eintrag `nInVar2 : INT;`.

⇨ Die Deklaration von nInVar1 steht jetzt an oberster Stelle.

4. Schließen Sie den Dialog mit **OK**.
   ⇨ Der Dialog **Refactoring** erscheint. Die betroffenen Stellen sind gelb markiert (siehe Abbildung oben).
5. Bestätigen Sie mit **OK**.

⇨ Die neue Reihenfolge wird in den Baustein übernommen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Refactoring - Variablen neu ordnen [▶ 930]

Variablendeklaration ändern und Refactoring automatisch anwenden

Sie werden in der Deklaration beim Umbenennen von Variablen (mithilfe der Autodeklaration) von Refactoring unterstützt.

✓ Sie haben einen Funktionsbaustein FB_Sample im Editor geöffnet. Der Funktionsbaustein besitzt eine Eingangsvariable „nVAR“.

✓ In den TwinCAT-Optionen (**Extras > Optionen**) in der Kategorie **TwinCAT > SPS Programmierumgebung > Refactoring** ist die Option **Beim Umbenennen von Variablen** aktiviert.

1. Selektieren Sie in der Deklaration von FB_Sample die Variable nVar. Alternativ können Sie den Cursor vor oder in die Variable setzen.
2. Wählen Sie im Menü **Bearbeiten** oder im Kontextmenü den Befehl **Variable deklarieren**.
   ⇨ Der Dialog **Variable deklarieren** öffnet sich. Der Dialog enthält die Einstellungen von nVar.
3. Ändern Sie den Namen von „nVar“ nach „nCounterVar“.
4. Die Option **Änderung mit Hilfe von Refactoring anwenden** erscheint und ist aktiviert. Die Option erscheint bei Änderung des Variablennamens unabhängig von den Einstellungen in den TwinCAT-Optionen. Sie ist jedoch nur dann automatisch aktiviert, wenn die in den Voraussetzungen genannte TwinCAT-Refactoring-Option aktiviert ist.
5. Klicken Sie auf **OK.**
   ⇨ Der Dialog **Refactoring** öffnet sich. Dort sind alle von der Variablenumbenennung betroffenen Stellen markiert und können geändert werden.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Variable deklarieren [▶ 915]
- Dokumentation TC3 User Interface: Dialog Optionen - Refactoring [▶ 1024]

7.18 Datenpersistenz

Persistente Daten sind Daten, die auch bei einem erneuten Laden des SPS-Projekts ihre Werte behalten. Die Werte persistenter Daten können Sie nach einem unkontrollierten Beenden (Stromausfall), einem Download oder einem Kaltstart wiederherstellen.

Neben den PERSISTENT-Variablen sind die RETAIN-Variablen eine weitere Möglichkeit, Daten remanent zu halten. RETAIN-Variablen behalten bei einem unkontrollierten Beenden (Stromausfall), einem Download oder einem Kaltstart ebenfalls ihre Werte.

Siehe auch:

- Reset des SPS-Projekts durchführen [▶ 230]
- Referenz Programmierung: Remanente Variablen – PERSISTENT, RETAIN [▶ 726]
- Retain Daten

7.19 Bausteine für implizite Prüfung verwenden

TwinCAT stellt spezielle POUs zur Verfügung, die implizite Überwachungsfunktionen implementieren. Diese speziellen POUs können Sie zu einer Anwendung hinzufügen, um sie mit implizit zur Verfügung gestellten Überwachungsfunktionalitäten auszustatten. Diese Funktionen überprüfen während der Laufzeit die Grenzen von Arrays oder Unterbereichstypen, die Gültigkeit von Pointer-Adressen oder eine Division durch 0. Beachten Sie, dass diese Möglichkeit bei Geräten deaktiviert sein kann, wenn solche Prüfbausteine durch eine spezielle implizite Bibliothek bereitgestellt werden.

7.19.1 Objekt POUs für implizite Prüfungen

Objekt POU für implizite Prüfungen anlegen

1. Selektieren Sie einen Ordner im SPS-Projektbaum.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU für implizite Prüfungen...**.
   ⇨ Der Dialog **POU für implizite Prüfungen hinzufügen** öffnet sich.
3. Aktivieren Sie die gewünschten Funktionen.
4. Klicken Sie auf die Schaltfläche **Öffnen**.
   ⇨ Die ausgewählten POUs werden in den SPS-Projektbaum eingefügt.
5. Öffnen Sie die POUs im Editor.
6. Passen Sie den Implementierungsvorschlag Ihren Anforderungen an.

Um ein mehrfaches Einbinden zu verhindern, steht eine bereits eingefügte Überwachungsfunktion im Dialog **POU für implizite Prüfungen hinzufügen** einfügen nicht mehr zur Auswahl.

Deklarationsteil nicht verändern

Um die Funktionalität der Überwachungsfunktionen zu erhalten, dürfen Sie den Deklarationsteil nicht verändern. Als einzige Ausnahme dürfen Sie lokale Variablen hinzufügen.

Kein Online-Change möglich

Nach dem Entfernen impliziter Überwachungsfunktionen, wie beispielsweise CheckBounds, aus dem Projekt, ist kein Online-Change mehr möglich, nur ein Download. Eine entsprechende Meldung wird ausgegeben.

Implizite Prüfungen für Bausteine aus Bibliotheken

TwinCAT führt implizite Prüfungen standardmäßig nicht für Bausteine aus Bibliotheken aus. Sie können jedoch durch die Compiler-Definition „checks_in_libs“ die Prüfung auf Bausteine aus Bibliotheken ausweiten. Um dies umzusetzen, tragen Sie die Compiler-Definition „checks_in_libs“ in den **SPS-Projekteigenschaften** in das Feld **Compilerdefinitionen** ein, welches Sie in der Kategorie **Übersetzen** finden. Dies wirkt sich auf Source-Bibliotheken (*.library) aus.

Ab TC3.1 Build 4026 kann die Prüfung auch auf Bausteine aus geschützten Bibliotheken (*.compiled-libraries) ausgeweitet werden. Dazu müssen Sie vor dem Speichern und Installieren als Bibliothek in den **SPS-Projekteigenschaften** in der Kategorie **Common** die Option „Implizite Prüfungen für Compiled Libraries erlauben“ aktivieren. Zusätzlich müssen Sie auch für geschützte Bibliotheken die Compiler-Definition „checks_in_libs“ eintragen.

Einzelne POUs von der Prüfung ausschließen

Sie können die Prüfung spezieller POUs im Projekt mit dem Attribut 'no_check' [▶ 848] deaktivieren.

Dialog Add POU für implizite Prüfungen

Verfügbare Funktionen

Überwachungsfunktion	Typ
CheckBounds	Feldgrenzenprüfungen (Bound Checks): Angemessene Behandlung von Verletzungen von Feldgrenzen (beispielsweise durch Setzen eines Fehlerflags oder durch Verändern des Feldindex).
CheckDivDInt	Divisionsprüfungen (Division Checks): Überwachung des Divisorwerts, um ein Dividieren durch 0 zu vermeiden.
CheckDivLInt	
CheckDivReal	
CheckDivLReal	
CheckRangeSigned	Bereichsprüfungen (Range Checks): Überwachung der Bereichsgrenzen eines Unterbereichstypen während der Laufzeit. Gilt für die Datentypen DINT/UDINT
CheckRangeUnsigned	
CheckLRangeSigned	L-Bereichsprüfungen: Überwachung der Bereichsgrenzen eines Unterbereichstypen während der Laufzeit. Gilt für die Datentypen LINT/ULINT
CheckLRangeUnsigned	
CheckPointer	Pointerprüfungen (Pointer Checks): Diese Funktion müssen Sie komplett selbst mit Implementierungscode füllen. Sehen Sie dazu die Hilfeseite zu POU CheckPointer. Die Funktion soll überwachen, ob der übergebene Zeiger auf eine gültige Speicheradresse verweist und ob die Ausrichtung des referenzierten Speicherbereichs zum Typ der Variablen passt, auf die der Zeiger verweist. Wenn beide Bedingungen erfüllt sind, wird der Zeiger selbst zurückgegeben. Andernfalls sollte CheckPointer eine angemessene Fehlerbehandlung durchführen. CheckPointer überwacht in der gleichen Weise auch Variablen vom Typ REFERENCE TO

Siehe auch:

- Referenz Programmierung: Attribut 'no_check' [▶ 848]

7.19.1.1 Bound Checks (POU CheckBounds)

Funktionen zur Überprüfung der Feldgrenzen: CheckBounds

Aufgabe dieser Überwachungsfunktion ist eine angemessene Behandlung von Verletzungen von Feldgrenzen. Eine Reaktion auf eine Verletzung kann beispielsweise das Setzen eines Error-Flags oder das Verändern des Arrayindex sein. Die Prüfung erfolgt nur bei einem variablen Arrayindex. Ein fehlerhafter konstanter Arrayindex führt zu einem Compilerfehler. TwinCAT ruft die Funktion implizit auf, sobald einer Variablen vom Typ ARRAY Werte zugewiesen werden.

Nach dem Einfügen der Funktion erhalten Sie automatisch erzeugten Code im Deklarationsteil und Implementierungsteil.

Deklarationsteil nicht verändern

Um die Funktionalität der Überwachungsfunktionen zu erhalten, dürfen Sie den Deklarationsteil nicht verändern. Als einzige Ausnahme dürfen Sie lokale Variablen hinzufügen.

Standardimplementierung

Deklarationsteil:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper: DINT;
END_VAR
```

Implementierungsteil:

```
// Implicitly generated code : Only an Implementation suggestion
{noflow}
IF index < lower THEN
    CheckBounds := lower;
ELSIF index > upper THEN
```

```
    CheckBounds := upper;
ELSE
    CheckBounds := index;
END_IF
{flow}
```

Beim Aufruf erhält die Funktion folgende Eingangsparameter:

- index: Index des Arrayelementes
- lower: Untergrenze des Arraybereichs
- upper: Obergrenze des Arraybereichs

Rückgabewert ist der Index des Arrayelements, sofern sich dieser im gültigen Bereich befindet. Ansonsten gibt TwinCAT je nach Verletzung des Grenzbereichs die Ober- oder Untergrenze zurück.

Beispielimplementierung

Deklarationsteil:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
    index, lower, upper: DINT;
END_VAR
// User defined local variables
VAR
    sMessageLow   : STRING := 'CheckBounds: Index too low (%d)';
    sMessageHigh  : STRING := 'CheckBounds: Index too high (%d)';
END_VAR
```

Implementierungsteil:

```
{noflow}
// Index too low
IF index < lower THEN
    CheckBounds := lower;
    // Increase global counter
    GVL_CheckBounds.nTooLow := GVL_CheckBounds.nTooLow + 1;
    // Log message
    ADSLOGDINT(msgCtrlMask := ADSLOG_MSGTYPE_WARN,
               msgFmtStr   := sMessageLow,
               dintArg     := index);

// Index too high
ELSIF index > upper THEN
    CheckBounds := upper;
    // Increase global counter
    GVL_CheckBounds.nTooHigh := GVL_CheckBounds.nTooHigh + 1;
    // Log message
    ADSLOGDINT(msgCtrlMask := ADSLOG_MSGTYPE_WARN,
               msgFmtStr   := sMessageHigh,
               dintArg     := index);

// Index OK
ELSE
    CheckBounds := index;
END_IF
{flow}
```

Wie in der Standardimplementierung wird in dieser Beispielimplementierung eine Verletzung des Arraybereichs korrigiert, indem die Ober- oder Untergrenze zurückgegeben wird. Zusätzlich dazu wird jeweils ein globaler Zähler hochgezählt, wenn außerhalb des definierten Arraybereichs auf das Array zugegriffen wird. Die globalen Zähler repräsentieren somit projektweit die Anzahl der Verletzungen der Ober- bzw. Untergrenze. Des Weiteren wird die Funktion ADSLOGDINT aus der Tc2_System-Bibliothek verwendet, um eine Verletzung der Arraygrenzen als Warnung im Meldungsfenster auszugeben.

Anwendungsbeispiel

Im unten stehenden Programm überschreitet der Index die definierte Obergrenze des Feldes aSample. Die CheckBounds-Funktion korrigiert diesen Zugriff auf das Array, welcher außerhalb der definierten Arraygrenzen stattfindet.

```
PROGRAM MAIN
VAR
    aSample : ARRAY[0..7] OF BOOL;
    nIndex  : INT := 10;
END_VAR
```

```
aSample[nIndex] := TRUE;
```

Die Funktion CheckBounds bewirkt in diesem Beispiel, dass der Index 10 in die Obergrenze des Arraybereichs von aSample (7) abgeändert wird. Damit wird der Wert TRUE dem Element aSample[7] zugewiesen. Auf diese Weise korrigiert die Funktion Arrayzugriffe außerhalb des gültigen Feldbereichs. Es sollte aber unbedingt auch die Fehlerursache, nämlich der Zugriff auf das Element an der Stelle 10, korrigiert werden. Damit dieser automatisch korrigierte, fehlerhafte Zugriff nicht unentdeckt bleibt, wird in diesem Beispiel, wie oben beschrieben, eine Warnung im Meldungsfenster ausgegeben.

Siehe auch:

- Bausteine für implizite Prüfung verwenden [▶ 172]
- Objekt POUs für implizite Prüfungen [▶ 172]

7.19.1.2 Division Checks (POUs CheckDivDInt, CheckDivLInt, CheckDivReal, CheckDivLReal)

Funktionen zur Vermeidung des Divisor-Werts „0“: CheckDivDInt, CheckDivLInt, CheckDivReal und CheckDivLReal

Um ein Teilen durch 0 zu vermeiden, können Sie die Funktionen CheckDivDInt, CheckDivLInt, CheckDivReal und CheckDivLReal verwenden. Wenn Sie diese Funktionen in das SPS-Projekt einbinden, werden sie vor jeder im Code auftretenden Division aufgerufen.

Deklarationsteil nicht verändern

Um die Funktionalität der Überwachungsfunktionen zu erhalten, dürfen Sie den Deklarationsteil nicht verändern. Als einzige Ausnahme dürfen Sie lokale Variablen hinzufügen.

Standardimplementierung der Funktion CheckDivReal

Deklarationsteil:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
    divisor : REAL;
END_VAR
```

Implementierungsteil:

```
// Implicitly generated code : Only an Implementation suggestion
{noflow}
IF divisor = 0 THEN
    CheckDivReal := 1;
ELSE
    CheckDivReal := divisor;
END_IF
{flow}
```

Der Operator DIV verwendet die Ausgabe der Funktion CheckDivReal als Divisor. Im unten stehenden Beispielprogramm verhindert diese Funktion eine Division durch 0 dadurch, dass der implizit mit „0“ initiierte Wert des Divisors fDivisor vor Ausführung der Teilung von der Funktion CheckDivReal in 1 geändert wird. Somit lautet das Ergebnis der Division 799.

```
PROGRAM MAIN
VAR
    fResult   : REAL;
    fDivident : REAL := 799;
    fDivisor  : REAL := 0;
END_VAR
fResult := fDivident / fDivisor;
```

7.19.1.3 Range/LRange Checks (POUs CheckRangeSigned, CheckRangeUnsigned, CheckLRangeSigned, CheckLRangeUnsigned)

Funktionen zur Überwachung der Bereichsgrenzen eines Unterbereichstypen

- CheckRangeSigned prüft Unterbereichstypen vom Typ SINT, INT, DINT.
- CheckRangeUnsigned prüft Unterbereichstypen vom Typ BYTE, WORD, DWORD, USINT, UINT, UDINT.
- CheckLRangeSigned prüft Unterbereichstypen vom Typ LINT.
- CheckLRangeUnsigned prüft Unterbereichstypen vom Typ LWORD, ULINT.

Aufgabe dieser Überwachungsfunktionen ist eine angemessene Behandlung von Verletzungen der Bereichsgrenzen. Eine Reaktion auf eine Verletzung kann beispielsweise das Setzen eines Error-Flags oder das Verändern eines Wertes sein. Der Aufruf der Funktionen erfolgt implizit bei der Zuweisung eines Wertes an eine Variable des Unterbereichstyps.

Deklarationsteil nicht verändern

Um die Funktionalität der Überwachungsfunktionen zu erhalten, dürfen Sie den Deklarationsteil nicht verändern. Als einzige Ausnahme dürfen Sie lokale Variablen hinzufügen.

Beim Aufruf werden der Funktion folgende Eingangsparameter übergeben:

- value: Wert, der der Variablen des Unterbereichstyps zugewiesen werden soll.
- lower: Bereichsuntergrenze
- upper: Bereichsobergrenze

Rückgabewert ist der Zuweisungswert selbst, sofern sich dieser im gültigen Bereich befindet. Ansonsten wird je nach Verletzung des Unterbereichs seine Ober- oder Untergrenze zurückgegeben.

Die Zuweisung `i := 10*y` wird nun implizit ersetzt durch

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

Hat `y` beispielsweise den Wert "1000", so wird der Variablen `i` nicht wie im ursprünglichen Code vorgesehen der Wert "10*1000=10000" zugewiesen, sondern der Wert der Bereichsobergrenze, also "4095".

Dasselbe gilt für die Funktionen CheckRangeUnsigned, CheckLRangeSigned und CheckLRangeUnsigned.

Steht Ihnen keine Funktion zur Bereichsüberwachung zur Verfügung, so erfolgt während der Laufzeit keine Überprüfung des Unterbereichs bei entsprechenden Variablen. In diesem Fall können Sie einer Variablen eines Unterbereichstyps von DINT/UDINT jeden Wert zwischen -2147483648 und +2147483648 beziehungsweise zwischen 0 und 4294967295 zuweisen. Einer Variablen eines Unterbereichstyps von LINT/ULINT können Sie dann jeden Wert zwischen --9223372036854775808 und +9223372036854775807 beziehungsweise zwischen 0 und18446744073709551615 zuweisen.

Endlosschleifen

Wenn Sie Bereichsüberwachungsfunktionen einbinden, können Endlosschleifen entstehen. Dies ist beispielsweise der Fall, wenn die Zählvariable einer FOR-Schleife vom Unterbereichstyp ist und der Zählbereich der Schleife den definierten Unterbereich verlässt!

Beispiel für eine Endlosschleife:

```
VAR
    nVar : DINT(0..10000);
    ...
END_VAR
```

```
FOR nVar := 0 TO 10000 DO
    ...
END_FOR
```

Das Programm verlässt niemals die FOR-Schleife, da die Überwachungsfunktion CheckRangeSigned verhindert, dass `nVar` auf einen Wert größer als 10000 gesetzt wird.

Standardimplementierung der Funktion CheckRangeSigned

Wird einer DINT-Variablen eines vorzeichenbehafteten Unterbereichstyps ein Wert zugewiesen, so bedingt dies einen automatischen Aufruf der Funktion CheckRangeSigned. Die Funktion, welche den Zuweisungswert auf den bei der Variablendeklaration festgesetzten Unterbereich beschränkt, ist standardmäßig wie folgt in ST implementiert:

Deklarationsteil:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
    value, lower, upper : DINT;
END_VAR
```

Implementierung:

```
// Implicitly generated code : Only an Implementation suggestion
{noflow}
IF (value < lower) THEN
    CheckRangeSigned := lower;
ELSEIF(value > upper) THEN
    CheckRangeSigned := upper;
ELSE
    CheckRangeSigned := value;
END_IF
{flow}
```

7.19.1.4 Pointer Checks (POU CheckPointer)

Überwachungsfunktion CheckPointer für Zeiger

Verwenden Sie die Funktion, um den Speicherzugriff von Zeigern während der Laufzeit zu überwachen. Im Unterschied zu anderen Überwachungsfunktionen existiert für die Implementierung von CheckPointer kein standardmäßiger Vorschlag. Eine Implementierung muss der Benutzer selbst vornehmen!

Die Funktion CheckPointer soll überprüfen, ob der übergebene Zeiger auf eine gültige Speicheradresse verweist und ob die Ausrichtung des referenzierten Speicherbereichs zum Typ der Variablen passt, auf die der Zeiger verweist. Sind beide Bedingungen erfüllt, so wird der Zeiger selbst zurückgegeben. Andernfalls sollte die Funktion eine angemessene Fehlerbehandlung durchführen.

Deklarationsteil nicht verändern

Um die Funktionalität der Überwachungsfunktionen zu erhalten, dürfen Sie den Deklarationsteil nicht verändern. Als einzige Ausnahme dürfen Sie lokale Variablen hinzufügen.

Für den THIS-Zeiger und den SUPER-Zeiger findet kein impliziter Aufruf der Überwachungsfunktion statt.

Die Funktion CheckPointer wirkt auch auf Variablen vom Typ REFERENCE in gleicher Weise wie auf Zeigervariablen.

Vorlage

Deklaration:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
    ptToTest : POINTER TO BYTE;
    iSize    : DINT;
    iGran    : DINT;
    bWrite   : BOOL;
END_VAR
```

Implementierung (unvollständig!):

```
// No standard way of implementation. Fill your own code here
{noflow}
CheckPointer := ptToTest;
{flow}
```

Beim Aufruf übergibt TwinCAT der Funktion folgende Eingabeparameter:

- ptToTest: Zieladresse des Zeigers
- iSize: Größe der referenzierten Variable in Bytes
- iGran: Granularität der referenzierten Größe in Bytes. Also der größte in der referenzierten Variablen enthaltene nicht-strukturierte Datentyp
- bWrite: Art des Zugriffs (TRUE=Schreibzugriff, FALSE=Lesezugriff)

Bei positivem Ergebnis der Überprüfung wird der unveränderte Eingabezeiger zurückgegeben (ptToTest).

Beispiel:

Folgendes Implementierungsbeispiel erzeugt eine Meldung im TwinCAT Ausgabefenster, sobald ein ungültiger Zeiger erkannt werden konnte. Diese Implementierung erkennt verschiedene Arten von ungültigen Zeigern. Es können aber nicht alle ungültigen Zeiger erkannt werden.

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
    ptToTest : POINTER TO BYTE;
    iSize    : DINT;
    iGran    : DINT;
    bWrite   : BOOL;
END_VAR
```

```
IF ptToTest=0 THEN
    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to invalid dest
ination address.','');
ELSIF iSize<=0 THEN
    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to invalid size
.','');
ELSIF iGran<=0 THEN
    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to invalid gran
ularity.','');
// -> Please note that the following memory area check is time consuming:
//ELSIF F_CheckMemoryArea(pData:=ptToTest,nSize:=DINT_TO_UDINT(iSize)) = E_TcMemoryArea.Unknown THEN
//    ADSLOGSTR(ADSLOG_MSGTYPE_ERROR OR ADSLOG_MSGTYPE_STRING,'CheckPointer failed due to unknown me
mory area.','');
END_IF
CheckPointer := ptToTest;
```

⚠ VORSICHT

Folge eines ungültigen Zeigers

Die Folge eines ungültigen Zeigers ist meist ein Stopp der Laufzeit, sobald ein beliebiger Zugriff über diesen Zeiger erfolgt. Die Funktion CheckPointer kann dies meist nicht verhindern. Zweck dieser Überwachungsfunktion ist vielmehr, eine effiziente Ursachendiagnose zu ermöglichen.

7.20 Objektorientiert programmieren

TwinCAT 3 unterstützt die objektorientierte Programmierung (OOP) und stellt dafür die folgenden Funktionalitäten und Objekte zur Verfügung:

- Funktionsbausteine (Objekt Funktionsbaustein [▶ 180])
- Methoden (Objekt Methode [▶ 182])
- Eigenschaften (Objekt Eigenschaft [▶ 189])
- Schnittstellen (Objekt Schnittstelle [▶ 196], Implementieren einer Schnittstelle [▶ 208])
- Vererbung
  - Definition von Funktionsbausteinen als Erweiterungen anderer Funktionsbausteine [▶ 201]
  - Definition von Strukturen als Erweiterungen anderer Strukturen [▶ 207]
  - Definition von Schnittstellen als Erweiterungen anderer Schnittstellen [▶ 208]

- Methodenaufruf [▶ 210]
- ABSTRACT-Konzept [▶ 212]

Grundgedanke der objektorientierten Programmierung

Bei der objektorientierten Programmierung wird die Software in Objekte unterteilt. Dabei werden alle Beschreibungen, die zu diesem Objekt gehören, in einem Element (z. B. in einem Funktionsbaustein) zusammengeführt. Die Beschreibungen umfassen dabei die Daten und die Prozeduren, über die das Objekt verfügen soll. Zudem kann über Methoden und Eigenschaften eine ausgewählte Zugriffsschnittstelle auf das Objekt definiert werden.

Dadurch werden bei diesem Programmieransatz Objekte entwickelt, die autark und unabhängig von spezifischen Rahmenbedingungen wiederverwendet werden können. Dabei können die Elemente beispielsweise in einer oder vielen Applikationen unverändert zum Einsatz kommen.

Vorteile der objektorientierten Programmierung

Die Verwendung der objektorientierten Programmierweise bietet viele Vorteile.

Durch die Einteilung der Software in Objekte kann eine **übersichtliche**, gut **strukturierte** Applikation entwickelt werden. Dadurch sind die Applikation und die einzelnen Elemente optimalerweise leicht und verständlich **lesbar** sowie leicht **erweiterbar**. Zusammen mit der geschaffenen **Wiederverwendbarkeit** von Programmierobjekten können bei der Entwicklung und Wartung von Applikationen letztlich **Zeit** und **Kosten** gespart werden.

Allgemeine Hinweise/Empfehlungen

- Bei der Verwendung der objektorientierten Programmierung ist darauf zu achten, für die jeweilige Applikation den richtigen „Grad der Objektorientierung“ zu finden.
  - Mit Hilfe von OOP wird die Software in Objekte unterteilt, sodass eine übersichtliche, strukturierte, wiederverwendbare Software entwickelt werden kann.
  - Falls die Einteilung der Objekte jedoch exzessiv und sehr detailliert vorgenommen wird, leidet die Verständlichkeit des Programms.
  - Beispiel: Mit Hilfe der Vererbung können Deklarationen und Implementierungen vererbt und somit von der Unterklasse wiederverwendet werden. Zudem kann durch die Einteilung in eine generelle und eine spezifische Klasse das Verständnis für die einzelnen Programmierobjekte unterstützt werden. Dies ist bei der Vererbung dann der Fall, wenn die Einteilung in Basis und Erweiterung schlüssig ist und sich beispielsweise in der realen Welt gleichermaßen oder ähnlich wiederfinden lässt (z. B. Basismotor und speziellerer Motor mit Zusatzfunktionen). Falls jedoch sehr viele Vererbungsebenen verwendet werden und die Funktion der einzelnen Vererbungslevel dadurch unklar bzw. unpräzise wird, ist dieser Teil der Applikation schwer zu verstehen und zu warten.
- Wenn Methoden Rückgabewerte liefern, sollten diese in der Methode gesetzt und an der Aufrufstelle der Methode ausgewertet werden.
- Wenn Methoden, Funktionen oder Eigenschaften strukturierte Rückgabetypen liefern (z. B. eine Struktur oder einen Funktionsbaustein), sollten diese Rückgabetypen als „REFERENCE TO <structured type>“ deklariert werden.
  - Das Zurückliefern von strukturierten Daten als direkter Rückgabetyp („<structured type>“) ist ineffizient, da die Daten mehrfach kopiert werden.
  - Wenn stattdessen „REFERENCE TO <structured type>“ verwendet wird, ist dies zum einen effizienter, da die Daten nicht kopiert werden, und zum anderen kann auf ein einzelnes Element des strukturierten Datentyps direkt beim Aufruf der Methode/Funktion/Eigenschaft zugegriffen werden.
  - Weitere Informationen und ein Beispiel finden Sie u. a. in der Beschreibung Objekt Methode [▶ 182].
- Die Programmierung von objektorientierten Implementierungen sollte eventbasiert sein.
  - Programmelemente sollten nicht grundlos zyklisch aufgerufen werden.
  - In der Regel ist ein Aufruf eines Programmelements dann sinnvoll, wenn ein bestimmtes Ereignis eingetreten ist.

- Ein Methoden- oder Eigenschaftenaufruf über eine nicht belegte Schnittstellenvariable gleicht einem NullPointer-Aufruf. Vor der Verwendung der Schnittstellenvariablen muss ihr eine entsprechende Funktionsbausteininstanz zugewiesen werden (Instanz eines Funktionsbausteins, der die Schnittstelle implementiert).
  - Um sicherzugehen, dass die Schnittstellenvariable keinem NullPointer entspricht, kann die Schnittstellenvariable vor der Verwendung auf ungleich 0 überprüft werden (z. B. `IF (iSample <> 0) THEN iSample.METH(); END_IF`).

OOP und UML

Müssen die objektorientierte Programmierung (OOP) und UML stets zusammen verwendet werden?

- Die kombinierte Nutzung von OOP und UML bietet viele Vorteile (s. nächste Frage), ist allerdings nicht zwangsläufig nötig. Es ist auch ohne die Verwendung von UML möglich, eine Applikation objektorientiert zu programmieren. Genauso kann UML auch in SPS-Projekten verwendet werden, die nicht objektorientiert programmiert werden bzw. wurden.

Welche Vorteile bietet es, OOP und UML zusammen zu verwenden?

- Um die Vorteile der OOP optimal zu nutzen, sollte die Struktur einer objektorientierten Software vor Implementierungsbeginn konzipiert bzw. erstellt werden (z. B.: Welche Klassen sind vorhanden, in welcher Beziehung stehen sie zueinander, welche Funktionalitäten stellen sie bereit, etc.). Vor, während und nach der Programmierung helfen Dokumentationen dabei, die Software zu verstehen, zu analysieren und zu warten.
- Als Analyse-, Design- und Dokumentationswerkzeug von Software bietet UML entsprechende Möglichkeiten, um die Applikation zu planen, zu erzeugen und zu dokumentieren. Dabei eignet sich die Verwendung von UML besonders bei objektorientierten Implementierungen, da vor allem modular aufgebaute Software mithilfe einer grafischen Sprache dargestellt werden kann.
- So wird beispielsweise das Klassendiagramm zum Analysieren und Erzeugen der Programmstruktur verwendet – und je modularer die Software aufgebaut ist, desto einfacher und effizienter kann das Klassendiagramm genutzt werden (z. B.: Grafische Darstellung von separaten Funktionsblöcken mit einzelnen Methoden zur Bereitstellung der Funktionalitäten, etc.).
- Das Zustandsdiagramm kann genutzt werden, um den Ablauf eines ereignisdiskreten Systems zu spezifizieren – und je konsequenter die Software objekt- bzw. ereignisorientiert aufgebaut ist, desto übersichtlicher und effektiver können Zustandsmaschinen entworfen werden (z. B.: Das Verhalten von Modulen/Systemen basiert auf einem Zustandsmodell mit Zuständen (wie Aufstarten, Produktion, Pausezustand) und innerhalb der Zustände werden entsprechende Funktionalitäten aufgerufen, die in Methoden (wie Aufstarten, Ausführen, Pausieren) gekapselt sind, etc.).

Siehe auch:

- Dokumentation zu: TF1910 TC3 UML
- Dokumentation Samples: Beispiel: Objektorientiertes Programm zur Steuerung einer Sortieranlage

7.20.1 Objekt Funktionsbaustein

Ein Funktionsbaustein ist eine POU [▶ 80], die bei der Ausführung einen oder mehrere Werte liefert. Die Werte der Ausgabevariablen und der internen Variablen bleiben nach einer Ausführung bis zur nächsten erhalten. Dies bedeutet, dass der Funktionsbaustein bei mehrmaligem Aufruf mit denselben Eingabevariablen nicht unbedingt dieselben Ausgabewerte liefert.

Im SPS-Projektbaum haben Funktionsbaustein-POUs das Suffix (FB). Der Editor eines Funktionsbausteins besteht aus dem Deklarationsteil und dem Implementierungsteil.

Sie rufen einen Funktionsbaustein immer über eine Instanz auf, die eine Kopie des Funktionsbausteins ist.

Zusätzlich zu der in der IEC 61131-3 beschriebenen Funktionalität können Sie Funktionsbausteine in TwinCAT auch für folgende Funktionalitäten der objektorientierten Programmierung verwenden:

- Erweitern eines Funktionsbausteins (Erweitern eines Funktionsbausteins [▶ 201])
- Implementieren von Schnittstellen (Implementieren einer Schnittstelle [▶ 208])
- Methoden (Objekt Methode [▶ 182])

- Eigenschaften (Objekt Eigenschaft [▶ 189])

Die oberste Zeile des Deklarationsteils enthält folgende Deklaration:

FUNCTION_BLOCK <access specifier> <function block> | EXTENDS <function block> |
IMPLEMENTS <comma-separated list of interfaces>

> **8-Byte-Alignment**
>
> Mit TwinCAT 3 wurde ein 8-Byte-Alignment eingeführt. Achten Sie auf ein passendes Alignment, wenn Daten als gesamter Speicherblock mit anderen Steuerungen oder Softwarekomponenten ausgetauscht werden (siehe Alignment [▶ 826]).

Automatisches Anlegen von Schnittstellenelementen in einem Funktionsbaustein

Es gibt zwei Möglichkeiten, wie Sie die Elemente einer Schnittstelle, die ein Funktionsbaustein implementiert, automatisch in diesem Funktionsbaustein erzeugen lassen können.

1. Wenn Sie beim Anlegen eines neuen Funktionsbausteins in dem Feld **Implementiert** des Dialogs **Hinzufügen** eine Schnittstelle angeben, fügt TwinCAT diesem Funktionsbaustein auch automatisch die Methoden und Eigenschaften der Schnittstelle hinzu.
2. Wenn ein bestehender Funktionsbaustein eine Schnittstelle implementiert, können Sie den Befehl **Schnittstellen implementieren** verwenden, um die Schnittstellenelemente in dem Funktionsbaustein erzeugen zu lassen. Den Befehl **Schnittstellen implementieren** finden Sie im Kontextmenü eines Funktionsbausteins im Projektbaum.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Schnittstellen implementieren [▶ 1112]
- Implementieren einer Schnittstelle: Implementieren einer Schnittstelle [▶ 208]

Funktionsbaustein aufrufen

Der Aufruf erfolgt immer über eine Instanz des Funktionsbausteins. Beim Aufruf eines Funktionsbausteins ändern sich nur die Werte der jeweiligen Instanz.

Deklaration der Instanz:

```
<instance> : <function block>;
```

Auf eine Variable des Funktionsbausteins greifen Sie im Implementierungsteil wie folgt zu:

```
<instance>.<variable>
```

- Sie können von außerhalb der Funktionsbaustein-Instanz nur auf Eingabe- und Ausgabevariablen eines Funktionsbausteins zugreifen, nicht auf die internen Variablen.
- Der Zugriff auf eine Funktionsbaustein-Instanz ist auf die POU begrenzt, in der die Instanz deklariert ist, außer Sie haben die Instanz global deklariert.
- Sie können beim Aufruf der Instanz den Funktionsbausteinvariablen die gewünschten Werte zuweisen.

Beispiel:

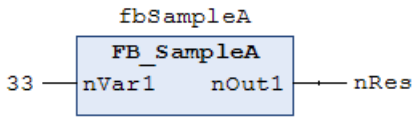
Der Funktionsbaustein FB_SampleA hat die Eingabevariable nVar1 vom Typ INT und die Ausgabevariable nOut1. Im Folgenden wird die Variable nVar1 aus dem Programm MAIN aufgerufen.

ST:

```
PROGRAM MAIN
VAR
    fbSampleA : FB_SampleA;
END_VAR
```

```
fbSampleA.nVar1 := 33; (* FB_SampleA is called and the value 33 is assigned to the variable nVar1 *)
fbSampleA(); (* FB_SampleA is called, that's necessary for the following access to the output variab
le *)
nRes := fbSampleA.nOut1 (* the output variable nOut1 of the FB1 is read *)
```

FUP:



Variablenwerte beim Aufruf zuweisen:

In den textuellen Sprachen AWL und ST können Sie Werte beim Aufruf des Funktionsbausteins direkt an Eingabe- und/oder Ausgabevariablen zuweisen.

Die Zuweisung eines Werts an einen Eingabevariable erfolgt mit :=

Die Zuweisung eines Wert an eine Ausgabevariable erfolgt mit =>

Beispiel:

Die Instanz fbTimer des Timer-Funktionsbausteins wird mit Zuweisungen für die Eingabevariable IN und PT aufgerufen. Anschließend wird die Ausgabevariable Q des Timers der Variablen bVarA zugewiesen

```
PROGRAM MAIN
VAR
    fbTimer : TOF;
    bIn     : BOOL;
    bVarA   : BOOL;
END_VAR
```

```
fbTimer(IN := bIn, PT := t#300ms);
bVarA := fbTimer.Q;
```

Wenn Sie eine Funktionsbaustein-Instanz über die Eingabehilfe einfügen und im Dialog **Eingabehilfe** die Option **Mit Argumenten einfügen** aktiviert ist, fügt TwinCAT den Aufruf mit allen Eingabe- und Ausgabevariablen ein. Sie müssen dann nur die gewünschten Wertzuweisungen einfügen. Im obigen Beispiel fügt TwinCAT den Aufruf wie folgt ein: CMD_TMR (IN:= , PT:= , Q=>).

Mit Hilfe des Attributs **'is_connected'** auf einer lokalen Variablen können Sie zur Zeit des Aufrufs in der Funktionsbaustein-Instanz feststellen, ob ein bestimmter Eingang eine Zuweisung von außen erhält.

7.20.2 Objekt Methode

Symbol: 

Das Objekt dient der objektorientierten Programmierung.

Eine Methode enthält eine Abfolge von Anweisungen, ist jedoch im Gegensatz zu einer Funktion keine unabhängige POU, sondern muss einem Funktionsbaustein oder einem Programm zugeordnet sein.

Zur Organisation von Methoden können Sie Schnittstellen verwenden.

Hinweise zu Methoden

- Alle Daten einer Methode sind temporäre Daten und sind nur während der Ausführung einer Methode gültig (Stack-Variablen). Das bedeutet, dass TwinCAT alle Variablen und Funktionsbausteine, die Sie in einer Methode deklariert haben, bei jedem Aufruf der Methode neu initialisiert.
- Methoden können ebenso wie Funktionen einen Rückgabewert zurückliefern.
- Gemäß der Norm IEC 61131-3 können Methoden ebenso wie normale Funktionen zusätzliche Ein- und Ausgänge besitzen. Die Ein- und Ausgänge weisen Sie beim Methodenaufruf zu.
  - **Ab TwinCAT 3.1.4026:** Eingänge ohne explizit vorgegebenen Initialwert müssen beim Aufruf der Methode zugewiesen werden. Eingänge mit explizit vorgegebenen Initialwert können optional zugewiesen werden oder beim Aufruf der Methode unbeachtet bleiben.
- Im Implementierungsteil einer Methode ist der Zugriff auf die Funktionsbausteininstanz- oder Programmvariablen erlaubt.
- Um auf die eigene Instanz zu verweisen, verwenden Sie den THIS-Zeiger.

- Sie können auf VAR_TEMP-Variablen des Funktionsbausteins in einer Methode nicht zugreifen.
- Sie können VAR_INST-Variablen deklarieren, für welche bei erneuten Methodenaufrufen keine Neuinitialisierung stattfindet (siehe auch Kapitel Instanzvariablen).
- Durch Verwendung des Rückgabetyps „REFERENCE TO <structured type>“ können Sie direkt beim Aufruf der Methode auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode zurückgeliefert wird, zugreifen. Weitere Informationen dazu finden Sie im Abschnitt „Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf [▶ 187]“.
- Auf VAR_IN_OUT-Variablen des Funktionsbausteins kann in einer Methode prinzipiell zugegriffen werden. Da dieser Zugriff potenziell riskant ist, sollte er mit Bedacht ausgeführt werden. Weitere Informationen dazu finden Sie im Abschnitt „Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition [▶ 188]“.
- Methoden, die in einer Schnittstelle definiert sind, dürfen nur Eingabe-, Ausgabe- und VAR_IN_OUT-Variablen definieren, aber keine Implementierung enthalten.

Beispiel:

Der Code im folgenden Beispiel bewirkt, dass TwinCAT den Rückgabewert und die Ausgänge der Methode auf lokal deklarierte Variablen schreibt.

Deklarationsteil der Methode „Method1“ des Funktionsbausteins FB_Sample:

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

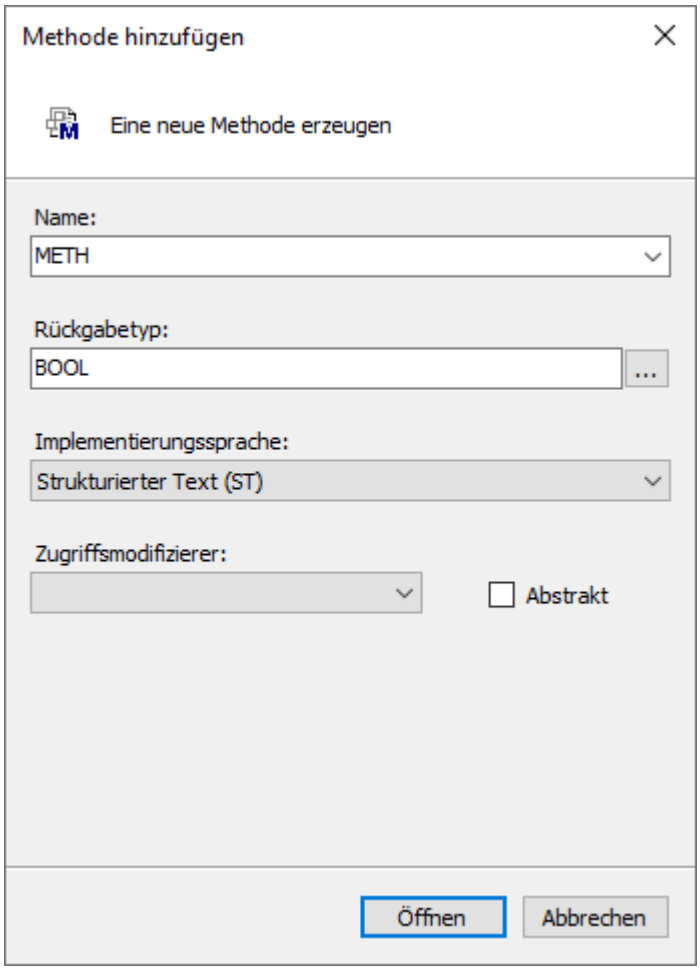
```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

Objekt Methode anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein oder ein Programm.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Methode…**
   ⇨ Der Dialog **Methode hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie einen Rückgabetyp sowie die Implementierungssprache und optional einen Zugriffsmodifizier aus
4. Klicken Sie auf **Öffnen**.

⇨ Das Objekt wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet. Der Editor besteht aus dem Deklarationseditor im oberen Teil und dem Implementierungsteil im unteren Teil.

Dialog Methode hinzufügen



Name	Name der Methode Die Standardmethoden FB_Init und FB_Exit werden in einer Auswahlliste angeboten, wenn sie nicht bereits unterhalb des Bausteins eingefügt sind. Wenn es sich um einen abgeleiteten Funktionsbaustein handelt, bietet die Auswahlliste außerdem alle Methoden des Basisbausteins an.
Rückgabetyp	Typ des Wertes, der zurückgegeben wird
Implementierungssprache	Auswahlliste für die Implementierungssprache

Zugriffsmodifizierer

Zugriffsmodifizierer	Regelt den Zugriff auf die Daten • **PUBLIC**: Der Zugriff ist nicht eingeschränkt (entspricht der Angabe keines Zugriffsmodifizierers). • **PRIVATE**: Der Zugriff auf die Methode ist auf den Funktionsbaustein bzw. das Programm beschränkt. • **PROTECTED**: Der Zugriff auf die Methode ist auf das Programm bzw. den Funktionsbaustein und seine Ableitungen beschränkt. • **INTERNAL**: Der Zugriff auf die Methode ist auf den Namensraum (die Bibliothek) beschränkt. Zusätzlich zu diesen Zugriffsmodifizierern können Sie manuell den Modifizierer FINAL zu einer Methode hinzufügen: • **FINAL**: Überschreibung der Methode in einer Ableitung des Funktionsbausteins ist nicht erlaubt. Das bedeutet, dass die Methode in einer möglicherweise vorhandenen Unterklasse nicht überschrieben/ erweitert werden darf.
Abstrakt	☑ : Kennzeichnet, dass die Methode keine Implementierung hat und die Implementierung durch den abgeleiteten FB bereitgestellt wird. Hintergrundinformationen zu dem Schlüsselwort ABSTRACT finden Sie unter ABSTRACT-Konzept [▶ 212].

Methoden mit einem anderen Zugriffsmodifizierer als PUBLIC werden im Projektmappen-Explorer im SPS-Projektbaum mit einem Signalsymbol gekennzeichnet:

Zugriffsmodifizierer	Objektsymbol	Signalsymbol
PRIVATE		(Schloss)
PROTECTED		(Stern)
INTERNAL		(Herz)

Wenn Sie eine Methode von einer POU zu einer Schnittstelle kopieren oder verschieben, löscht TwinCAT die enthaltenen Implementierungen automatisch.

Spezielle Methoden für einen Funktionsbaustein

FB_init	Deklarationen automatisch implizit. Auch explizite Deklaration möglich. Enthält Initialisierungscode für den Funktionsbaustein, wie im Deklarationsteil des Funktionsbausteins definiert ist. (Methoden FB_init, FB_reinit und FB_exit [▶ 887])
FB_reinit	Explizite Deklaration notwendig. Aufruf, nachdem die Instanz des Funktionsbausteins kopiert wurde (wie während eines Online-Change), und reinitialisiert das neue Instanzmodul. (Methoden FB_init, FB_reinit und FB_exit [▶ 887])
FB_exit	Explizite Deklaration notwendig. Aufruf für jede Instanz des Funktionsbausteins vor einem erneuten Download oder einem Reset oder während eines Online-Change für alle verschobenen oder gelöschten Instanzen. (Methoden FB_init, FB_reinit und FB_exit [▶ 887])
Eigenschaften und Schnittstelleneigenschaften	Bestehen jeweils aus einer Set- und/oder Get-Accessor-Methode. (Objekt Schnittstelleneigenschaft [▶ 201], Objekt Eigenschaft [▶ 189])

Methode aufrufen

Syntax:

```
<return value variable> := <POU name>.<method name>(<method input name> :=
<variable name> (, <further method input name> := <variable name> )* );
```

Beim Methodenaufruf weisen Sie an die Eingabevariablen der Methode Übergabeparameter zu. Beachten Sie dabei die Deklaration. Es genügt, wenn Sie die Namen der Eingangsvariablen angeben, ohne deren Reihenfolge in der Deklaration zu beachten.

Beispiel:

Der Code im folgenden Beispiel bewirkt, dass TwinCAT den Rückgabewert und die Ausgänge der Methode auf lokal deklarierte Variablen schreibt.

Deklarationsteil der Methode „Method1“ des Funktionsbausteins FB_Sample:

```
METHOD Method1 : BOOL
VAR_INPUT
    nIn1  : INT;
    bIn2  : BOOL;
END_VAR
VAR_OUTPUT
    fOut1 : REAL;
    sOut2 : STRING;
END_VAR
```

```
// <method implementation code>
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample      : FB_Sample;
    bReturnValue  : BOOL;
    nLocalInput1  : INT;
    bLocalInput2  : BOOL;
    fLocalOutput1 : REAL;
    sLocalOutput2 : STRING;
END_VAR
```

```
bReturnValue := fbSample.Method1(nIn1  := nLocalInput1,
                                 bIn2  := bLocalInput2,
                                 fOut1 => fLocalOutput1,
                                 sOut2 => sLocalOutput2);
```

Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf

Um direkt bei einem Methoden-, Funktions- oder Eigenschaftenaufruf auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode/Funktion/Eigenschaft zurückgeliefert wird, zugreifen zu können, kann folgende Umsetzung verwendet werden. Ein strukturierter Datentyp ist beispielsweise eine Struktur oder ein Funktionsbaustein.

1. Der Rückgabetyp der Methode/Funktion/Eigenschaft wird als „REFERENCE TO <structured type>“ definiert (anstelle von lediglich „<structured type>“).
2. Bei einem solchen Rückgabetyp ist zu beachten, dass – falls beispielsweise eine FB-lokale Instanz des strukturierten Datentyps zurückgeliefert werden soll – der Referenzoperator REF= anstatt des „normalen“ Zuweisungsoperators := verwendet werden muss.

Die Erklärungen und das Beispiel dieses Abschnitts beziehen sich auf den Aufruf einer Eigenschaft. Sie sind aber genauso auf andere Aufrufe übertragbar, die Rückgabewerte liefern (z. B. Methoden oder Funktionen).

Beispiel

Deklaration der Struktur ST_Sample (strukturierter Datentyp):

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

Deklaration des Funktionsbausteins FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

Deklaration der Eigenschaft FB_Sample.MyProp mit dem Rückgabetyp „REFERENCE TO ST_Sample“:

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

Implementierung der Get-Methode von der Eigenschaft FB_Sample.MyProp:

```
MyProp REF= stLocal;
```

Implementierung der Set-Methode von der Eigenschaft FB_Sample.MyProp:

```
stLocal := MyProp;
```

Aufruf der Get- und Set-Methoden im Hauptprogramm MAIN:

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

Durch die Deklaration des Rückgabetyps der Eigenschaft MyProp als „REFERENCE TO ST_Sample“ und durch die Verwendung des Referenzoperators REF= in der Get-Methode dieser Eigenschaft, kann direkt beim Aufruf der Eigenschaft auf ein einzelnes Element des zurückgelieferten strukturierten Datentyps zugegriffen werden.

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

Wenn der Rückgabetyp nur als „ST_Sample“ deklariert wäre, müsste die von der Eigenschaft zurückgelieferte Struktur zunächst einer lokalen Strukturinstanz zugewiesen werden. Die einzelnen Strukturelemente könnten dann anhand der lokalen Strukturinstanz abgefragt werden.

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition/Eigenschaft

Auf die VAR_IN_OUT-Variablen eines Funktionsbausteins kann in einer Methode, einer Transition oder einer Eigenschaft des Funktionsbausteins prinzipiell zugegriffen werden. Bei einem solchen Zugriff ist folgendes zu beachten:

- Wird der Rumpf oder eine Aktion des Funktionsbausteins von außerhalb des FBs aufgerufen, stellt der Compiler sicher, dass bei diesem Aufruf die VAR_IN_OUT-Variablen des Funktionsbausteins zugewiesen werden.
- Bei dem Aufruf einer Methode, Transition oder Eigenschaft des Funktionsbausteins ist dies nicht der Fall, da die VAR_IN_OUT-Variablen des FBs nicht innerhalb eines Methoden-, Transitions- oder Eigenschaftenaufrufs zugewiesen werden können. Es könnte daher gegebenenfalls ein Zugriff auf die VAR_IN_OUT-Variablen stattfinden, indem die Methode/Transition/Eigenschaft aufgerufen wird, bevor die VAR_IN_OUT-Variablen einer gültigen Referenz zugewiesen wurden. Da dies einem ungültigen Zugriff während der Laufzeit entsprechen würde, ist es potentiell riskant, auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft zuzugreifen.

Aus diesem Grund wird bei einem Zugriff auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft folgende Warnung mit der ID C0371 ausgegeben:

„Warning: Access to VAR_IN_OUT <Var> declared in <POU> from external context <Method/Transition/Property>”

Eine adäquate Reaktion auf diese Warnung ist beispielsweise die Überprüfung der VAR_IN_OUT-Variablen innerhalb der Methode/Transition/Eigenschaft, bevor auf sie zugegriffen wird. Diese Überprüfung ist mithilfe des Operators __ISVALIDREF möglich, mit dem geprüft werden kann, ob eine Referenz auf einen gültigen Wert verweist. Ist diese Prüfung vorhanden, kann zum einen davon ausgegangen werden, dass sich der Anwender des Risikos bewusst ist, das potentiell vorhanden ist, wenn auf die VAR_IN_OUT-Variablen des FBs in einer Methode/Transition/Eigenschaft zugegriffen wird. Zum anderen liegt durch die Überprüfung der Referenz ein angemessener Umgang mit diesem Risiko vor. Somit kann die Warnung für diesen überprüften Bereich mittels Attribut 'warning disable' unterdrückt werden.

Die dazugehörige Beispielimplementierung einer Methode ist im Folgenden dargestellt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

Methode FB_Sample.MyMethod:

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
```

```
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

Siehe auch:

- Objektorientiert programmieren [▶ 178]
- Objekt Schnittstelle [▶ 196]
- Implementieren einer Schnittstelle [▶ 208]
- Erweitern eines Funktionsbausteins [▶ 201]
- Methodenaufruf [▶ 210]
- ABSTRACT-Konzept [▶ 212]
- Referenz Programmierung > Instanzvariablen - VAR_INST [▶ 724]

7.20.3 Objekt Eigenschaft

Symbol: 

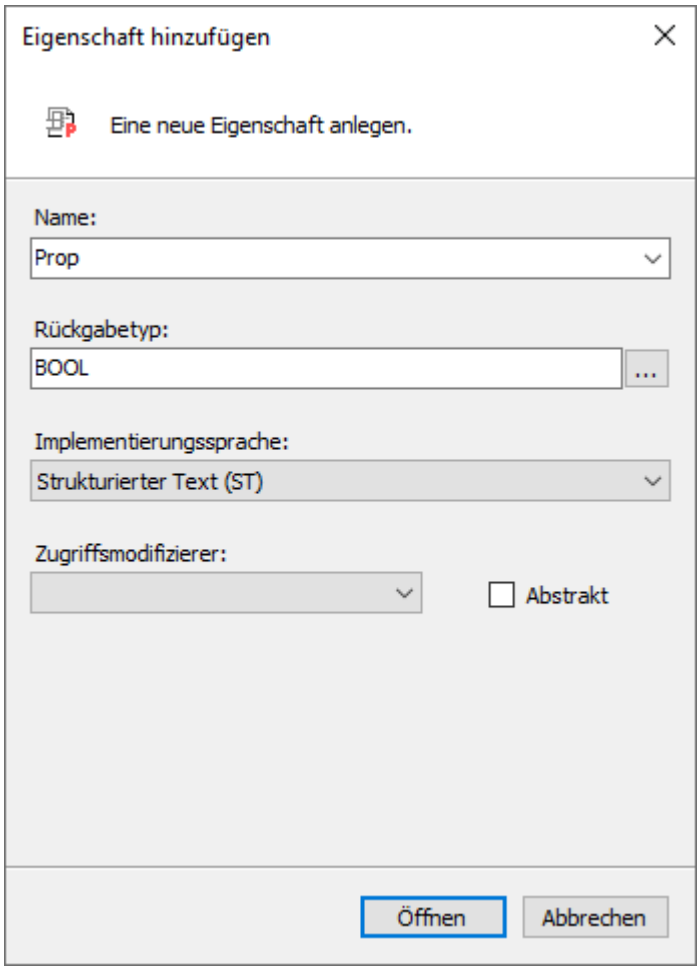
Eine Eigenschaft ist eine Erweiterung der Norm IEC 61131-3 und ist ein Mittel der objektorientierten Programmierung. Sie besteht aus den Accessor-Methoden Get und Set. TwinCAT ruft diese Methoden automatisch auf, sobald ein Lese- oder Schreibzugriff auf den Funktionsbaustein erfolgt, der die Eigenschaft implementiert.

Objekt Eigenschaft anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein oder ein Programm.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Eigenschaft…**
   ⇨ Der Dialog **Eigenschaft hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie einen Rückgabetyp sowie die Implementierungssprache und optional einen Zugriffsmodifizier aus.
4. Klicken Sie auf **Öffnen**.

⇨ Das Objekt wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog Eigenschaft hinzufügen



Name	Name der Eigenschaft
Rückgabetyp	Typ des Wertes, der zurückgegeben wird (Standardtyp oder strukturierter Typ)
Implementierungssprache	Auswahlliste für die Implementierungssprache

Zugriffsmodifizierer

Zugriffsbezeichner	Regelt den Zugriff auf die Daten • **PUBLIC**: Der Zugriff ist nicht eingeschränkt (entspricht der Angabe keines Zugriffsmodifizierers). • **PRIVATE**: Der Zugriff auf die Eigenschaft ist auf den Funktionsbaustein bzw. das Programm beschränkt. • **PROTECTED**: Der Zugriff auf die Eigenschaft ist auf das Programm bzw. den Funktionsbaustein und seine Ableitungen beschränkt. • **INTERNAL**: Der Zugriff auf die Eigenschaft ist auf den Namensraum (die Bibliothek) beschränkt. Zusätzlich zu diesen Zugriffsmodifizierern können Sie manuell den Modifizierer FINAL zu einer Eigenschaft hinzufügen: **FINAL**: Überschreibung der Eigenschaft in einer Ableitung des Funktionsbausteins ist nicht erlaubt. Das bedeutet, dass die Eigenschaft in einer möglicherweise vorhandenen Unterklasse nicht überschrieben/ erweitert werden darf.
Abstrakt	☑ : Kennzeichnet, dass die Eigenschaft keine Implementierung hat und die Implementierung durch den abgeleiteten FB bereitgestellt wird. Hintergrundinformationen zu dem Schlüsselwort ABSTRACT finden Sie unter ABSTRACT-Konzept [▶ 212].

Eigenschaften mit einem anderen Zugriffsmodifizierer als PUBLIC werden im Projektmappen-Explorer im SPS-Projektbaum mit einem Signalsymbol gekennzeichnet:

Zugriffsmodifizierer	Objektsymbol	Signalsymbol
PRIVAT		 (Schloss)
PROTECTED		 (Stern)
INTERNAL		 (Herz)

Eine Eigenschaft kann zusätzliche lokale Variablen enthalten. Eine Eigenschaft kann aber keine zusätzlichen Eingänge enthalten und, im Gegensatz zu einer Funktion oder Methode, keine zusätzlichen Ausgänge.

Wenn Sie eine Eigenschaft von einer POU zu einer Schnittstelle kopieren oder verschieben, löscht TwinCAT die enthaltenen Implementierungen automatisch.

Get- und Set-Accessoren

Die Accessor-Methoden Get und Set fügt TwinCAT automatisch im SPS-Projektbaum unterhalb des Eigenschaftenobjekts ein. Mithilfe des Befehls **Hinzufügen** können Sie sie auch explizit hinzufügen.

TwinCAT ruft den Set-Accessor auf, wenn auf die Eigenschaft schreibend zugegriffen wird, das bedeutet, Sie verwenden den Namen der Eigenschaft als Eingabeparameter.

TwinCAT ruft den Get-Accessor auf, wenn auf die Eigenschaft lesend zugegriffen wird, das bedeutet, Sie verwenden den Namen der Eigenschaft als Ausgabeparameter.

Bitte beachten Sie, dass Sie die Accessor-Methoden implementieren müssen, um auf die Eigenschaft zuzugreifen.

Beispiel für die Implementierung

Deklaration des Funktionsbausteins FB_Sample

```
FUNCTION_BLOCK FB_Sample
VAR
    nVar : INT;
END_VAR
```

```
nVar := nVar + 1;
```

Deklaration der Eigenschaft nValue

```
PROPERTY PUBLIC nValue : INT
```

Implementierung der Accessormethode FB_Sample.nValue.Set

```
nVar := nValue;
```

Implementierung der Accessormethode FB_Sample.nValue.Get

```
nValue := nVar;
```

Aufruf der Eigenschaft nValue

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
END_VAR
```

```
fbSample();
If fbSample.nValue > 500 THEN
    fbSample.nValue := 0;
END_IF;
```

Wenn Sie die Eigenschaft nur für den Lesezugriff oder nur für den Schreibzugriff verwenden wollen, können Sie den Accessor löschen, den Sie nicht verwenden.

Sie können den Accessor-Methoden an folgenden Stellen Zugriffsbezeichner hinzufügen:

- Im Deklarationsteil des Accessors durch manuelles Eintragen.
- Im Dialog **'get'-Accessor hinzufügen** oder **'set'-Accessor hinzufügen**, wenn Sie den Accessor explizit mit dem Befehl **Hinzufügen** einfügen.

Kompatibilitätswarnung bei Eigenschaften vom Typ REFERENCE

Zu TC3.1 Build 4022 ändert sich das Aufrufverhalten von Eigenschaften, die mit dem Rückgabetyp 'REFERENCE TO <…>' definiert sind.

Bei Schreibzugriffen mittels ':=' wird mit Versionen < 3.1.4022.0 der Set-Accessor aufgerufen, sodass die Referenz geschrieben wird. Mit Versionen >= 3.1.4022.0 wird hingegen der Get-Accessor aufgerufen, sodass der Wert geschrieben wird. Um mit Versionen >= 3.1.4022.0 die Referenz zuzuweisen, muss der Referenz-Zuweisungsoperator 'REF= [▶ 661]' verwendet werden.

Monitoring für Eigenschaften im Onlinebetrieb

Für das Monitoring für Eigenschaften im Onlinebetrieb stehen folgende Pragmas zur Verfügung, die Sie an oberster Stelle in der Definition der Eigenschaft einfügen (Attribut 'monitoring' [▶ 846]):

- {attribute 'monitoring':= 'variable'}: Bei jedem Zugriff auf die Eigenschaft speichert TwinCAT den Istwert in einer Variablen und stellt den Wert dieser Variablen dar. Dieser Wert kann unter Umständen veralten, wenn im Code kein Zugriff mehr auf die Eigenschaft erfolgt.
- {attribute 'monitoring' := 'call'}: Bei jeder Darstellung des Werts ruft TwinCAT den Code des Get-Accessors auf. Wenn dieser Code einen Seiteneffekt enthält, dann wird der Seiteneffekt durch das Monitoring ausgeführt.

Sie können eine Eigenschaft mithilfe folgender Funktionalitäten monitoren:

- Inline-Monitoring
  Voraussetzung: In den TwinCAT-Optionen in der Kategorie **TwinCAT > SPS Programmierumgebung > Texteditor** ist in der Registerkarte **Monitoring** die Option **Inline-Monitoring aktivieren** aktiviert.
- Überwachungsliste (Überwachungslisten verwenden [▶ 239])

Zugriff auf ein einzelnes Element eines strukturierten Rückgabetyps beim Methoden-/Funktions-/Eigenschaftenaufruf

Um direkt bei einem Methoden-, Funktions- oder Eigenschaftenaufruf auf ein einzelnes Element des strukturierten Datentyps, welcher von der Methode/Funktion/Eigenschaft zurückgeliefert wird, zugreifen zu können, kann folgende Umsetzung verwendet werden. Ein strukturierter Datentyp ist beispielsweise eine Struktur oder ein Funktionsbaustein.

1. Der Rückgabetyp der Methode/Funktion/Eigenschaft wird als „REFERENCE TO <structured type>“ definiert (anstelle von lediglich „<structured type>“).
2. Bei einem solchen Rückgabetyp ist zu beachten, dass – falls beispielsweise eine FB-lokale Instanz des strukturierten Datentyps zurückgeliefert werden soll – der Referenzoperator REF= anstatt des „normalen“ Zuweisungsoperators := verwendet werden muss.

Die Erklärungen und das Beispiel dieses Abschnitts beziehen sich auf den Aufruf einer Eigenschaft. Sie sind aber genauso auf andere Aufrufe übertragbar, die Rückgabewerte liefern (z. B. Methoden oder Funktionen).

Beispiel

Deklaration der Struktur ST_Sample (strukturierter Datentyp):

```
TYPE ST_Sample :
STRUCT
    bVar  : BOOL;
    nVar  : INT;
END_STRUCT
END_TYPE
```

Deklaration des Funktionsbausteins FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR
    stLocal     : ST_Sample;
END_VAR
```

Deklaration der Eigenschaft FB_Sample.MyProp mit dem Rückgabetyp „REFERENCE TO ST_Sample“:

```
PROPERTY MyProp : REFERENCE TO ST_Sample
```

Implementierung der Get-Methode von der Eigenschaft FB_Sample.MyProp:

```
MyProp REF= stLocal;
```

Implementierung der Set-Methode von der Eigenschaft FB_Sample.MyProp:

```
stLocal := MyProp;
```

Aufruf der Get- und Set-Methoden im Hauptprogramm MAIN:

```
PROGRAM MAIN
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
    stGet       : ST_Sample;
    bSet        : BOOL;
    stSet       : ST_Sample;
END_VAR
```

```
// Get - single member and complete structure possible
nSingleGet := fbSample.MyProp.nVar;
stGet      := fbSample.MyProp;

// Set - only complete structure possible
IF bSet THEN
    fbSample.MyProp REF= stSet;
    bSet            := FALSE;
END_IF
```

Durch die Deklaration des Rückgabetyps der Eigenschaft MyProp als „REFERENCE TO ST_Sample“ und durch die Verwendung des Referenzoperators REF= in der Get-Methode dieser Eigenschaft, kann direkt beim Aufruf der Eigenschaft auf ein einzelnes Element des zurückgelieferten strukturierten Datentyps zugegriffen werden.

```
VAR
    fbSample    : FB_Sample;
    nSingleGet  : INT;
END_VAR
```

```
nSingleGet := fbSample.MyProp.nVar;
```

Wenn der Rückgabetyp nur als „ST_Sample“ deklariert wäre, müsste die von der Eigenschaft zurückgelieferte Struktur zunächst einer lokalen Strukturinstanz zugewiesen werden. Die einzelnen Strukturelemente könnten dann anhand der lokalen Strukturinstanz abgefragt werden.

```
VAR
    fbSample    : FB_Sample;
    stGet       : ST_Sample;
    nSingleGet  : INT;
END_VAR
```

```
stGet      := fbSample.MyProp;
nSingleGet := stGet.nVar;
```

Zugriff auf VAR_IN_OUT-Variablen des Funktionsbausteins in einer Methode/Transition/Eigenschaft

Auf die VAR_IN_OUT-Variablen eines Funktionsbausteins kann in einer Methode, einer Transition oder einer Eigenschaft des Funktionsbausteins prinzipiell zugegriffen werden. Bei einem solchen Zugriff ist folgendes zu beachten:

- Wird der Rumpf oder eine Aktion des Funktionsbausteins von außerhalb des FBs aufgerufen, stellt der Compiler sicher, dass bei diesem Aufruf die VAR_IN_OUT-Variablen des Funktionsbausteins zugewiesen werden.
- Bei dem Aufruf einer Methode, Transition oder Eigenschaft des Funktionsbausteins ist dies nicht der Fall, da die VAR_IN_OUT-Variablen des FBs nicht innerhalb eines Methoden-, Transitions- oder Eigenschaftenaufrufs zugewiesen werden können. Es könnte daher gegebenenfalls ein Zugriff auf die VAR_IN_OUT-Variablen stattfinden, indem die Methode/Transition/Eigenschaft aufgerufen wird, bevor die VAR_IN_OUT-Variablen einer gültigen Referenz zugewiesen wurden. Da dies einem ungültigen Zugriff während der Laufzeit entsprechen würde, ist es potentiell riskant, auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft zuzugreifen.

Aus diesem Grund wird bei einem Zugriff auf die VAR_IN_OUT-Variablen des FBs in einer Methode, einer Transition oder einer Eigenschaft folgende Warnung mit der ID C0371 ausgegeben:

„Warning: Access to VAR_IN_OUT <Var> declared in <POU> from external context <Method/Transition/Property>”

Eine adäquate Reaktion auf diese Warnung ist beispielsweise die Überprüfung der VAR_IN_OUT-Variablen innerhalb der Methode/Transition/Eigenschaft, bevor auf sie zugegriffen wird. Diese Überprüfung ist mithilfe des Operators __ISVALIDREF möglich, mit dem geprüft werden kann, ob eine Referenz auf einen gültigen Wert verweist. Ist diese Prüfung vorhanden, kann zum einen davon ausgegangen werden, dass sich der Anwender des Risikos bewusst ist, das potentiell vorhanden ist, wenn auf die VAR_IN_OUT-Variablen des FBs in einer Methode/Transition/Eigenschaft zugegriffen wird. Zum anderen liegt durch die Überprüfung der Referenz ein angemessener Umgang mit diesem Risiko vor. Somit kann die Warnung für diesen überprüften Bereich mittels Attribut 'warning disable' unterdrückt werden.

Die dazugehörige Beispielimplementierung einer Methode ist im Folgenden dargestellt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut : BOOL;
END_Var
```

Methode FB_Sample.MyMethod:

```
METHOD MyMethod
VAR_INPUT
END_VAR
```

```
// The warning can be disabled here as the user is aware of the risk that the reference may not be
valid by checking its validity
{warning disable C0371}

// Checking the VAR_IN_OUT reference, leave the current method in case of invalid reference
IF NOT __ISVALIDREF(bInOut) THEN
     RETURN;
```

```
END_IF

// Access to VAR_IN_OUT reference (only if the reference was confirmed as valid before)
bInOut := NOT bInOut;

// The warning may be restored at the end of the access area
{warning restore C0371}
```

Initialisierungsbeispiel:

Im folgenden Beispiel werden eine Eingangsvariable und eine Eigenschaft von einem Funktionsbaustein initialisiert, welcher über eine FB_init-Methode [▶ 888] mit einem zusätzlichen Parameter verfügt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

Methode FB_Sample.FB_init:

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

Eigenschaft FB_Sample.nMyProperty und die zugehörige Set-Funktion:

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

Initialisierungsergebnis:

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8
  - nLocalInitParam = 7
  - nLocalProp = 9

Siehe auch:

- Objekt Schnittstelleneigenschaft [▶ 201]
- Objektorientiert programmieren [▶ 178]

- Erweitern eines Funktionsbausteins [▶ 201]
- Referenz Programmierung > Methoden FB_init, FB_reinit und FB_exit [▶ 887]

7.20.4 Objekt Schnittstelle

Symbol: 

Schlüsselwort: INTERFACE

Eine Schnittstelle ist ein Mittel der objektorientierten Programmierung. Das Objekt **Schnittstelle** beschreibt ein Set von Methoden- und Eigenschaft-Prototypen. Prototyp bedeutet in diesem Zusammenhang, dass die Methoden und Eigenschaften nur Deklarationen und keine Implementierung enthalten.

Auf diese Weise können Sie verschiedene Funktionsbausteine, die gemeinsame Eigenschaften haben, gleichartig nutzen.

Dem Objekt **Schnittstelle** können Sie die Objekte **Schnittstelleneigenschaft** und **Schnittstellenmethode** hinzufügen.

Objekt Schnittstelle anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Ordner.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Schnittstelle…**
   ⇨ Der Dialog **Schnittstelle hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein und wählen Sie optional eine Schnittstelle aus, welche erweitert werden soll.
4. Klicken Sie auf **Öffnen**.

⇨ Die Schnittstelle wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet.

Dialog Schnittstelle hinzufügen

Name	Schnittstellenname

Vererbung

Erweitert	 : Erweitert die Schnittstelle, die Sie im Eingabefeld eingeben oder über die Eingabehilfe  auswählen. Dies bedeutet, dass alle Methoden der Schnittstelle, die die neue Schnittstelle erweitert, auch in der neuen Schnittstelle verfügbar sind.

Anwendungsfälle einer Schnittstelle

1. Prüfung der Schnittstellenvereinbarung mittels Compiler

- In einer Schnittstelle (z. B. I_Sample) deklarieren Sie die gewünschten Methoden und Eigenschaften (einschließlich Rückgabetyp, Eingänge etc.), die zu dieser Schnittstelle gehören sollen.
- In den Funktionsbausteinen, die dieser Schnittstelle entsprechen sollen und somit die zugehörigen Methoden und Eigenschaften bereitstellen sollen, implementieren Sie die Schnittstelle I_Sample.
  - Der Funktionsbaustein enthält die Schnittstelle dabei in seiner IMPLEMENTS-Liste innerhalb seines Deklarationsteils (z. B. FB_Sample IMPLEMENTS I_Sample).
  - Ein Funktionsbaustein kann eine oder mehrere Schnittstellen implementieren (z. B. FB_Sample IMPLEMENTS I_Sample1, I_Sample2).
- Ein Funktionsbaustein, der eine Schnittstelle implementiert, muss alle Methoden und Eigenschaften enthalten, die in dieser Schnittstelle definiert sind (Schnittstellenmethoden und -eigenschaften). Dabei muss die Deklaration der Methoden und Eigenschaften exakt der Deklaration in der Schnittstelle entsprechen (Name, Rückgabetype, Eingänge, Ausgänge).

- Die Funktionsbausteine fügen den Schnittstellenmethoden und Schnittstelleneigenschaften Funktionsbaustein-spezifischen Code hinzu. Wenn eine Schnittstelle von mehreren Funktionsbausteinen implementiert wird, können Sie die gleiche Methode mit identischen Parametern, aber unterschiedlichem Implementierungscode in verschiedenen Funktionsbausteinen verwenden.
- Für die Funktionsbausteine, die eine oder mehrere Schnittstellen implementieren, überprüft der Compiler, ob sich die Funktionsbausteine an die jeweiligen Schnittstellenvereinbarungen halten. Wenn sich die Deklaration der Elemente in der Schnittstelle und in dem Funktionsbaustein unterscheiden oder wenn die Schnittstelle über weitere Elemente verfügt, die der Funktionsbaustein nicht enthält, meldet der Compiler einen Fehler.

2. Aufruf von Methoden und Eigenschaften einer Funktionsbausteininstanz über eine Schnittstellenvariable

Neben der automatischen Prüfung der Schnittstellenvereinbarung mittels Compiler können Sie Schnittstellen verwenden, um über eine Schnittstellenvariable eine Schnittstellenmethode oder Schnittstelleneigenschaft einer Funktionsbausteininstanz aufzurufen.

- Sie instanziieren zunächst sowohl die Schnittstelle (z. B. `iSample : I_Sample;`) als auch den bzw. die Funktionsbausteine, die die Schnittstelle ordnungsgemäß implementieren (siehe Anwendungsfall 1: Prüfung der Schnittstellenvereinbarung mittels Compiler).
- Anschließend können Sie der Schnittstellenvariablen jede Instanz eines Funktionsbausteins zuweisen, der die Schnittstelle ordnungsgemäß implementiert. Wenn für eine Schnittstellenvariable noch keine Zuweisung erfolgt ist, enthält die Variable im Onlinebetrieb den Wert 0.
- Im letzten Schritt können Sie über die Schnittstellenvariable eine Schnittstellenmethode oder -eigenschaft aufrufen, wobei die Methode bzw. Eigenschaft für die Funktionsbausteininstanz aufgerufen wird, auf die die Schnittstelle verweist.
- Über eine solche Umsetzung können Sie verschiedene, aber gleichartige Funktionsbausteine über die Schnittstellenvariable einheitlich verwenden. Abhängig vom Projektzustand können Sie der Schnittstellenvariablen beispielsweise eine bestimmte Funktionsbausteininstanz zuweisen, sodass der Aufruf der Schnittstellenmethoden und -eigenschaften identisch ist, aber je nach Projektzustand eine andere Funktionsbausteininstanz verwendet wird.

Einer Variablen vom Typ einer Schnittstelle müssen Sie die Instanz eines Funktionsbausteins zuweisen, bevor eine Methode oder Eigenschaft über die Schnittstellenvariable aufgerufen werden kann.

Hinweise

- Sie dürfen keine Variablen innerhalb einer Schnittstelle deklarieren. Eine Schnittstelle hat keinen Implementierungsteil und keine Aktionen. Es wird nur eine Sammlung von Methoden und Eigenschaften definiert, welche Deklarationscode, aber keinen Implementierungscode enthalten.
- Eine Variable vom Typ einer Schnittstelle ist eine Referenz auf Instanzen von Funktionsbausteinen. TwinCAT behandelt Variablen, die mit dem Typ einer Schnittstelle deklariert sind, immer als Referenzen.

Schnittstellenreferenzen und Online-Change

- Ab TwinCAT 3.0 Build 3100 werden Schnittstellenreferenzen automatisch umadressiert, sodass in jedem Fall, also auch bei einem Online-Change, die korrekte Schnittstelle referenziert wird. Zusätzlicher Code und mehr Zeit werden dafür benötigt und somit muss abhängig von der Anzahl der betroffenen Objekte mit zeitlichen Problemen gerechnet werden. Daher wird dem Programmierer vor Ausführung des Online-Change die Anzahl der betroffenen Variablen und Schnittstellenreferenzen angezeigt, und er muss dann entschieden, ob der Online-Change durchgeführt oder abgebrochen werden soll.

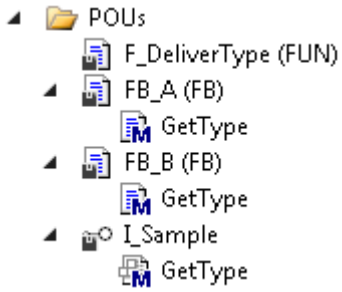
Prüfen von Schnittstellenvariablen

- Den Operator __ISVALIDREF können Sie nur für Operanden vom Typ REFERENCE TO verwenden. Die Prüfung von Schnittstellenvariablen ist mit diesem Operator nicht möglich. Um zu überprüfen, ob einer Schnittstellenvariable bereits eine Funktionsbausteininstanz zugewiesen wurde, können Sie die Schnittstellenvariable auf ungleich 0 prüfen (IF iSample <> 0 THEN …).

Erweitertes Monitoring einer Schnittstellenvariablen

- Ab TwinCAT 3.1 Build 4024 stehen erweiterte Möglichkeiten zum Monitoring/Debugging einer Schnittstellenvariablen zur Verfügung. Dabei kann eine Schnittstellenvariable im Monitoring-Bereich (Deklarationseditor, Überwachungsliste) aufgeklappt werden. Unterhalb der Schnittstellenvariablen werden dann der Symbolpfad und die Onlinedaten der aktuell zugewiesenen FB-Instanz angezeigt.

Beispiel 1



Deklaration der Schnittstelle:

- Sie haben die Schnittstelle I_Sample zu Ihrem Projekt hinzugefügt. Der Schnittstelle fügen Sie die Methode GetType mit dem Rückgabetyp STRING hinzu.
- I_Sample und GetType enthalten keinen Implementierungscode. Die Methode GetType enthält lediglich die benötigten (Variablen-) Deklarationen (z. B. Rückgabetyp). Die Methode GetType können Sie später im Funktionsbaustein ausprogrammieren, der die Schnittstelle I_Sample implementiert.

```
INTERFACE I_Sample
```

Methode I_Sample.GetType:

```
METHOD GetType : STRING
```

Implementierung der Schnittstelle:

- Wenn Sie anschließend einen Funktionsbaustein zum Projekt hinzufügen und in dem Dialog **Hinzufügen** in dem Feld **Implementiert** die Schnittstelle I_Sample angeben, fügt TwinCAT diesem Funktionsbaustein auch automatisch die Methode GetType hinzu. Hier können Sie in den Methoden funktionsbaustein-spezifischen Code implementieren.
- Die Funktionsbausteine FB_A und FB_B implementieren jeweils die Schnittstelle I_Sample:

```
FUNCTION_BLOCK FB_A IMPLEMENTS I_Sample
```

```
FUNCTION_BLOCK FB_B IMPLEMENTS I_Sample
```

- Beide Funktionsbausteine müssen daher eine Methode mit dem Namen GetType und dem Rückgabetyp STRING enthalten. Ansonsten meldet der Compiler einen Fehler (siehe Anwendungsfall 1 im Abschnitt Anwendungsfälle einer Schnittstelle [▶ 196]).

Methode FB_A.GetType:

```
METHOD GetType : STRING
```

```
GetType := 'FB_A';
```

Methode FB_B.GetType:

```
METHOD GetType : STRING
```

```
GetType := 'FB_B';
```

Verwendung der Schnittstelle:

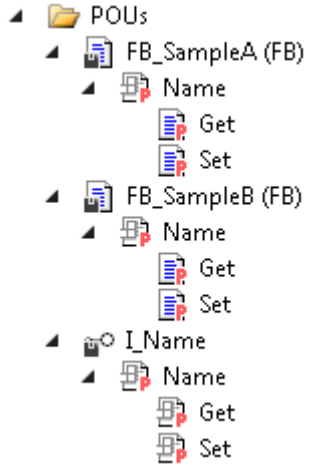
- Eine Funktion F_DeliverType enthält die Deklaration einer Eingangsvariablen vom Typ der Schnittstelle I_Sample. Innerhalb der Funktion wird die Schnittstellenmethode GetType über die Schnittstellenvariable iSample aufgerufen. In diesem Fall hängt es vom übergebenen Funktionsbausteintyp ab, ob FB_A.GetType oder FB_B.GetType aufgerufen wird (siehe Anwendungsfall 2 im Abschnitt Anwendungsfälle einer Schnittstelle [▶ 196]).

```
FUNCTION F_DeliverType : STRING
VAR_INPUT
    iSample : I_Sample;
END_VAR

F_DeliverType := iSample.GetType();
```

- Instanzen von Funktionsbausteinen, die die Schnittstelle I_Sample implementieren (z. B. FB_A und FB_B), können der Eingangsvariablen der Funktion F_DeliverType zugewiesen werden.
- Beispiele für Funktionsaufrufe:
  - Wenn der Funktion F_DeliverType die Funktionsbausteininstanz fbA übergeben wird, wird innerhalb der Funktion die Methode fbA.GetType aufgerufen, da die Schnittstellenvariable iSample auf die Funktionsbausteininstanz fbA zeigt. Dieser Methodenaufruf liefert den Rückgabewert 'FB_A', welcher wiederum von der Funktion F_DeliverType zurückgegeben wird und im Hauptprogramm der Variablen sResultA zugewiesen wird.
  - Entsprechend erhält sResultB den Wert 'FB_B', da innerhalb der Funktion F_DeliverType die Methode fbB.GetType aufgerufen wird.

```
PROGRAM MAIN
VAR
    fbA       : FB_A;
    fbB       : FB_B;
    sResultA  : STRING;
    sResultB  : STRING;
END_VAR

sResultA := F_DeliverType(iSample := fbA); // call with instance of type FB_A
sResultB := F_DeliverType(iSample := fbB); // call with instance of type FB_B
```

Beispiel 2



Deklaration der Schnittstelle:

- Sie haben die Schnittstelle I_Name zu Ihrem Projekt hinzugefügt. Der Schnittstelle fügen Sie die Eigenschaft Name mit dem Rückgabetyp STRING hinzu. Die Eigenschaft besitzt die Accessor-Methoden Get und Set. Der Get-Accessor soll dazu dienen, den Namen irgendeines Objekts aus einem Funktionsbaustein auszulesen, der die Schnittstelle implementiert. Den Set-Accessor verwenden Sie dazu, den Namen in diesen Funktionsbaustein zu schreiben.
- I_Name und Name enthalten keinen Implementierungscode. Die Eigenschaft Name enthält lediglich die benötigte Deklaration (Rückgabetyp). Die Get- und Set-Methoden können Sie innerhalb der Schnittstellendefinition somit nicht bearbeiten, jedoch später im Funktionsbaustein, der die Schnittstelle I_Name implementiert.

```
INTERFACE I_Name
```

Eigenschaft I_Name.Name:

```
PROPERTY Name : STRING
```

Implementierung der Schnittstelle:

- Die Funktionsbausteine FB_SampleA und FB_SampleB implementieren die Schnittstelle I_Name.

- Wenn die Schnittstelle beispielsweise beim Anlegen der Funktionsbausteine im Dialog **Hinzufügen** angegeben wird, fügt TwinCAT die Eigenschaft Name mit den Get- und Set-Methoden automatisch unterhalb der Funktionsbausteine FB_SampleA und FB_SampleB ein.
- Unterhalb der Funktionsbausteine können Sie die Accessor-Methoden editieren, beispielsweise so, dass die Variable sVar1 gelesen wird und Sie somit den Namen eines Objekts erhalten. In FB_SampleB, der dieselbe Schnittstelle I_Name implementiert, können Sie in der Get-Methode Code implementieren, der dann den Namen eines anderen Objekts liefert. Die Set-Methode verwenden Sie, um den Namen, den das Programm MAIN liefert ('abc'), in den Funktionsbaustein FB_SampleB zu schreiben.
- Die Funktionsbausteine FB_SampleA und FB_SampleB implementieren jeweils die Schnittstelle I_Name:

```
FUNCTION_BLOCK FB_SampleA IMPLEMENTS I_Name
VAR
    sVar1 : STRING := 'My name is A.';
END_VAR

FUNCTION_BLOCK FB_SampleB IMPLEMENTS I_Name
VAR
    sVar2 : STRING := 'My name is B.';
END_VAR
```

- Beide Funktionsbausteine müssen daher eine Eigenschaft mit dem Namen Name und dem Rückgabetyp STRING enthalten. Die Eigenschaft muss über eine Get- und eine Set-Methode verfügen. Ansonsten meldet der Compiler einen Fehler (siehe Anwendungsfall 1 im Abschnitt Anwendungsfälle einer Schnittstelle [▶ 196]).

FB_SampleA.Name.Get:

```
Name := sVar1;
```

FB_SampleA.Name.Set:

```
sVar1 := Name;
```

FB_SampleB.Name.Get:

```
Name := sVar2;
```

FB_SampleB.Name.Set:

```
sVar2 := Name;
```

Verwendung der Schnittstelle:

- Auf die Eigenschaften der Funktionsbausteine kann sowohl über die entsprechenden Funktionsbausteininstanzen als auch über eine Schnittstellenvariable vom Typ I_Name zugegriffen werden. Voraussetzung für den Zugriff über eine Schnittstellenvariable ist, dass dieser Variablen zuvor eine konkrete Funktionsbausteininstanz zugewiesen wurde, die die Schnittstelle I_Name implementiert.

```
PROGRAM MAIN
VAR
    iName     : I_Name;

    fbSampleA : FB_SampleA;
    sNameA    : STRING;       // will be 'My name is A.'

    fbSampleB : FB_SampleB;
    sNameB    : STRING;       // will be 'My name is B.' after first cycle
                              // and will be 'New name' afterwards
END_VAR

// assign FB instance fbSample1 to interface variable
iName := fbSampleA;

// access to name property of fbSample1 via interface variable (Get)
sNameA := iName.Name;

// access to name property of fbSample2 via FB instance (Get and Set)
sNameB := fbSampleB.Name;
fbSampleB.Name := 'New name';
```

Siehe auch:

- Implementieren einer Schnittstelle [▶ 208]
- Erweitern einer Schnittstelle [▶ 208]

7.20.4.1 Objekt Schnittstellenmethode

Symbol: 

Eine Schnittstellenmethode ist ein Mittel der objektorientierten Programmierung. Das Objekt **Schnittstellenmethoden** fügen Sie einer Schnittstelle über den Befehl **Hinzufügen > Method…** hinzu.

Wenn eine Methode unterhalb einer Schnittstelle eingefügt ist, können Sie in dieser Methode nur Variablendeklarationen (Eingabe-, Ausgabe- und Ein/Ausgabe-Variablen) hinzufügen und instanziieren.

Der Methode können Sie erst dann Programmcode hinzufügen, wenn ein Funktionsbaustein die Schnittstelle, zu der die Methode gehört, „implementiert“. TwinCAT fügt die Methode dann unterhalb des Funktionsbausteins ein.

Siehe auch:

- Objekt Schnittstelle [▶ 196]
- Objekt Methode [▶ 182]
- Implementieren einer Schnittstelle [▶ 208]

7.20.4.2 Objekt Schnittstelleneigenschaft

Symbol: 

Eine Schnittstelleneigenschaft ist ein Mittel der objektorientierten Programmierung. Das Objekt **Schnittstelleneigenschaft** fügen Sie einer Schnittstelle über den Befehl **Hinzufügen > Property…** hinzu, um die Beschreibung der Schnittstelle mit den Accessor-Methoden Get und/oder Set zu erweitern. Für die Accessor-Methoden ist in der Schnittstelleneigenschaft kein Implementierungscode enthalten. Wenn Sie den Set-Accessor löschen, dann besteht nur ein lesender Zugriff auf die Eigenschaft, und kein schreibender Zugriff.

Der Get-Accessor dient dem Lesezugriff auf die Eigenschaft.
Der Set-Accessor dient dem Schreibzugriff auf die Eigenschaft.

Wenn die Eigenschaft kein Get und/oder Set hat, fügen Sie diesen Accessor der Schnittstelleneigenschaft mit dem Befehl **Hinzufügen** hinzu.

Wenn Sie einen Funktionsbaustein oder ein Programm mit einer Schnittstelle erweitern, die Eigenschaften enthält, fügt TwinCAT diese und ihre zugehörigen Get- und/oder Set-Accessoren automatisch im SPS-Projektbaum unterhalb der POU im SPS-Projektbaum ein. Anschließend können Sie in den Get- und/oder Set-Accessoren den Code implementieren.

Siehe auch:

- Objekt Schnittstelle [▶ 196]
- Objekt Eigenschaft [▶ 189]
- Implementieren einer Schnittstelle [▶ 208]

7.20.5 Erweitern eines Funktionsbausteins

Die Erweiterung eines Funktionsbausteins basiert auf dem Konzept der Vererbung in der objektorientierten Programmierung. Ein abgeleiteter Funktionsbaustein „erweitert“ zu diesem Zweck einen Basis-Funktionsbaustein und erhält („erbt“) dadurch prinzipiell die Eigenschaften und Funktionalitäten des Basis-Funktionsbausteins – zusätzlich zu seinen eigenen Eigenschaften und Funktionalitäten.

Die Bezeichnung „Funktionsbaustein“ kann in TwinCAT synonym zur Bezeichnung „Klasse“ verwendet werden. Der abgeleitete Funktionsbaustein kann somit auch als „Unterklasse“ und der Basis-Funktionsbaustein als „Basisklasse“ bezeichnet werden.

Beachten Sie unbedingt die folgenden Informationen:

- Vererbungsprinzip [▶ 203]
- Anwendungsfälle von geerbten Elementen [▶ 205]

Anzahl der Erweiterungen pro Basis-Funktionsbaustein

Die Anzahl der Erweiterungen pro Basis-Funktionsbaustein ist nicht beschränkt. Sie können also einen Funktionsbaustein durch mehrere andere Funktionsbausteine erweitern und dadurch auf verschiedene Arten spezialisieren.

- Möglich:
  FUNCTION_BLOCK FB_Sub1 EXTENDS FB_Base
  FUNCTION_BLOCK FB_Sub2 EXTENDS FB_Base
  FUNCTION_BLOCK FB_Sub3 EXTENDS FB_Base

Anzahl der Vererbungsebenen

Die Anzahl der Vererbungsebenen ist nicht beschränkt. Sie können also einen Funktionsbaustein, der einen anderen Funktionsbaustein erweitert, in einem weiteren Funktionsbaustein spezialisieren etc.

- Möglich:
  FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
  FUNCTION_BLOCK FB_SubSub EXTENDS FB_Sub
  FUNCTION_BLOCK FB_SubSubSub EXTENDS FB_SubSub

Mehrfachvererbung nicht erlaubt

Bei Funktionsbausteinen ist Mehrfachvererbung nicht erlaubt. Es ist nicht möglich, dass ein Funktionsbaustein mehr als einen anderen Funktionsbaustein erweitert.

Ausnahme: Ein Funktionsbaustein kann mehrere Schnittstellen implementieren und eine Schnittstelle kann mehrere andere Schnittstellen erweitern.

- Nicht möglich:
  FUNCTION_BLOCK FB_Sub EXTENDS FB_Base1, FB_Base2
- Möglich:
  FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample1, I_Sample2
- Möglich:
  INTERFACE I_Sub EXTENDS I_Base_1, I_Base_2

Überladen

Das Überladen von Methoden ist nicht möglich. Wenn Sie in einer Unterklasse also eine Methode der Basisklasse überschreiben oder erweitern (gleicher Methodenname), muss die Methodendeklaration der Deklaration in der Basisklasse entsprechen (Zugriffsmodifizierer, Rückgabetyp, Variablenschnittstelle).

Erweitern eines Basis-Funktionsbausteins durch einen neuen Funktionsbaustein

✓ Das aktuell geöffnete Projekt besitzt einen Basis-Funktionsbaustein, zum Beispiel FB_Sample, der durch einen neuen Funktionsbaustein erweitert werden soll.

1. Selektieren Sie das SPS-Projektobjekt oder einen Unterordner im SPS-Projektbaum und wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU…**

   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
2. Geben Sie einen Namen für den neuen Funktionsbaustein in das Eingabefeld Name ein, beispielsweise FB_SampleEx.
3. Wählen Sie **Funktionsbaustein**.

4. Wählen Sie **Erweitert** und klicken Sie auf die Schaltfläche ... .
5. Wählen Sie in der Eingabehilfe aus der Kategorie **Funktionsbausteine** unter dem Projekt die POU(FB) aus, die als Basis-Funktionsbaustein dienen soll, zum Beispiel „FB_Sample“ und klicken Sie auf **OK**.
6. Optional können Sie einen **Zugriffsmodifizierer** für den neuen Funktionsbaustein aus der Combobox auswählen.
7. Wählen Sie aus der Combobox **Implementierungsprache** zum Beispiel „Strukturierter Text (ST)“ aus.
8. Klicken Sie auf **Öffnen**.

⇨ TwinCAT fügt den Funktionsbaustein FB_SampleEx in den SPS-Projektbaum ein und der Editor öffnet sich. In der ersten Zeile steht:
FUNCTION_BLOCK FB_SampleEx EXTENDS FB_Sample
Der Funktionsbaustein FB_SampleEx erweitert den Basis-Funktionsbaustein FB_Sample.

Erweitern eines Basis-Funktionsbausteins durch einen bestehenden Funktionsbaustein

✓ Das aktuell geöffnete Projekt besitzt einen Basis-Funktionsbaustein, zum Beispiel „FB_Sample“, und einen weiteren Funktionsbaustein, zum Beispiel „FB_SampleEx“, der noch nicht vom Basis-Funktionsbaustein abgeleitet ist. Der Funktionsbaustein FB_SampleEx soll den Basis-Funktionsbaustein erweitern, das bedeutet: FB_SampleEx soll FB_Sample erweitern.

1. Doppelklicken Sie im SPS-Projektbaum auf den Funktionsbaustein FB_SampleEx.
   ⇨ Der Editor der POU öffnet sich.
2. Erweitern Sie den bestehenden Eintrag der obersten Zeile FUNCTION_BLOCK FB_SampleEx mit EXTENDS FB_Sample.

⇨ Der Funktionsbaustein FB_SampleEx erweitert den Basis-Funktionsbaustein FB_Sample.

Siehe auch:

- Objekt Funktionsbaustein [▶ 180]
- Objekt Eigenschaft [▶ 189]
- Objekt Methode [▶ 182]
- Objekt Aktion [▶ 89]
- Objekt Transition [▶ 90]
- Referenz Programmierung: SUPER [▶ 728]
- Referenz Programmierung: THIS [▶ 730]

7.20.5.1 Vererbungsprinzip

Inhalt der Vererbung

Ein abgeleiteter Funktionsbaustein erbt alle Daten, Methoden, Eigenschaften, Aktionen und Transitionen, die im Basis-Funktionsbaustein definiert sind. Beachten Sie die Zugriffsmöglichkeiten auf geerbte Elemente, die mithilfe des Zugriffsmodifizierers definiert werden.

Zugriffsmöglichkeiten auf geerbte Elemente

Inwieweit eine Unterklasse innerhalb ihres Gültigkeitsbereichs auf geerbte Methoden oder Eigenschaften zugreifen kann, hängt von dem Zugriffsmodifizierer ab, mit dem die Methode oder Eigenschaft in der Basisklasse definiert ist.

Methoden und Eigenschaften, die in der Basisklasse mit dem Zugriffsmodifizierer PRIVATE deklariert sind, können innerhalb des Gültigkeitsbereich der Unterklasse weder aufgerufen werden, noch können sie von der Unterklasse überschrieben oder erweitert werden.
Private Methoden und Eigenschaften stehen der Unterklasse nur insofern zur Verfügung, als dass sie für die Instanz der Unterklasse ausgeführt werden, falls sie innerhalb der Implementierung der Basisklasse aufgerufen werden.

Beispiel:

Die Basisklasse verfügt über eine PUBLIC- und eine PRIVATE-Methode. Die PUBLIC-Methode ruft in ihrer Implementierung die PRIVATE-Methode auf. Die PUBLIC-Methode kann von der Unterklasse aufgerufen werden, sodass dabei implizit die PRIVATE-Methode mit aufgerufen wird. Die PRIVATE-Methode kann von der Unterklasse allerdings nicht aktiv aufgerufen, überschrieben oder erweitert werden.

Zur Festlegung der Zugriffsmöglichkeiten auf eine Methode oder eine Eigenschaft stehen folgende Zugriffsmodifizierer zur Verfügung:

PUBLIC	Entspricht der Angabe keines Zugriffsmodifizierers. Das Element (Methode oder Eigenschaft) kann von außerhalb des Funktionsbausteins aufgerufen werden. Somit kann auch eine Unterklasse auf das Element zugreifen
PRIVATE	Der Zugriff auf das Element ist auf den Funktionsbaustein beschränkt. Ein Zugriff von außerhalb des Funktionsbausteins ist in keiner Weise möglich. Auch eine Unterklasse kann nicht auf das Element zugreifen. Die Unterklasse kann das Element somit weder aufrufen noch überschreiben oder erweitern
PROTECTED	Der Zugriff auf das Element ist auf den Funktionsbaustein und seine Ableitungen beschränkt. Eine Unterklasse kann auf das Element zugreifen und es somit sowohl aufrufen als auch erweitern oder überschreiben. Ein Zugriff von außerhalb dieser "Vererbungsfamilie" ist nicht möglich.
INTERNAL	Der Zugriff auf das Element ist auf den Namensraum (die Bibliothek) beschränkt. Ein Zugriff von außerhalb des Namensraums ist nicht möglich. Das Element kann von außerhalb des Namensraums somit weder aufgerufen noch überschrieben oder erweitert werden.

Erweitern bzw. Überschreiben von geerbten Elementen

- Ein abgeleiteter Funktionsbaustein darf die Methoden, Eigenschaften, Aktionen und Transitionen, die im Basis-Funktionsbaustein definiert sind, erweitern bzw. überschreiben, falls für die Elemente in der Basisklasse ein entsprechender Zugriffsmodifizierer verwendet wird.
- Um ein Element zu erweitern oder zu überschreiben, muss das Element in der Unterklasse auf die gleiche Weise wie in der Basisklasse deklariert werden:
  - gleicher Name
  - gleicher Zugriffsmodifizierer (bei Methoden und Eigenschaften)
  - gleiche Variablenschnittstelle (z. B. Methodenein-/ausgänge)
  - gleicher Rückgabetyp (bei Methoden und Eigenschaften)
- Beim Erweitern bzw. Überschreiben eines Elements wird in der Unterklasse lediglich der Implementierungsteil angepasst, um das Element mit einem veränderten Verhalten zu versehen.

Engineering-Tipp: Für dieses Erweitern bzw. Überschreiben der vom Basisbaustein geerbten Methoden, Eigenschaften, Aktionen und Transitionen erhalten Sie beim Engineering folgendermaßen Unterstützung: Wenn Sie eine Methode, Eigenschaft etc. zu dem abgeleiteten Baustein hinzufügen, erhalten Sie im Feld **Name** des Hinzufügen-Dialogs (z. B. **Methode hinzufügen**, **Eigenschaft hinzufügen**) eine Drop-down-Liste mit einer Auswahl der im Basisbaustein verwendeten Methoden, Eigenschaften etc. Wenn Sie z. B. im Dialog **Methode hinzufügen** eine dieser angebotenen Methoden auswählen, werden die übrigen Deklarationseinstellungen der Methode (Rückgabetyp, Zugriffsmodifizierer) automatisch von der Methodendeklaration der Basisklasse übernommen. Beim Bestätigen des Dialogs wird die Methode gemäß dieser Deklarationen angelegt. Sie können den Implementierungsteil der Methode daraufhin anpassen, sodass sie dem gewünschten Verhalten der Unterklasse entspricht.

Weitere Informationen zum Thema Erweitern bzw. Überschreiben finden Sie im Abschnitt „Anwendungsfälle von geerbten Elementen [▶ 205]“.

Weitere Aspekte der Erweiterung

- Eine Instanz des abgeleiteten Funktionsbausteins kann in jedem Kontext verwendet werden, in dem TwinCAT einen Funktionsbaustein vom Typ des Basis-Funktionsbausteins erwartet.
- Ein abgeleiteter Funktionsbaustein darf keine Funktionsbausteinvariablen mit denselben Namen enthalten, wie Sie der Basis-Funktionsbaustein bereits deklariert hat. Dies meldet der Compiler als Fehler.
  Einzige Ausnahme: Wenn Sie eine Variable im Basis-Funktionsbaustein als VAR_TEMP deklariert

haben, dann darf der abgeleitete Funktionsbaustein eine Variable mit demselben Namen definieren. In diesem Fall kann der abgeleitete Funktionsbaustein auf die Variable des Basis-Funktionsbausteins nicht mehr zugreifen.

- Die Variablen und Elemente (z. B. Methoden, Eigenschaften, Aktionen, Transitionen) des Basis-Funktionsbausteins können innerhalb des Gültigkeitsbereichs des abgeleiteten Funktionsbausteins durch die Verwendung des SUPER-Zeigers direkt angesprochen werden.

7.20.5.2 Anwendungsfälle von geerbten Elementen

Generell können geerbte Elemente, auf die die Unterklasse entsprechenden Zugriff besitzt (siehe Abschnitt „Zugriffsmöglichkeiten auf geerbte Elemente [▶ 203]“), auf drei verschiedene Arten genutzt werden:

- Geerbte Elemente können unverändert genutzt werden.
- Geerbte Elemente können überschrieben werden.
- Geerbte Elemente können erweitert werden.

Diese drei Anwendungsfälle werden im Folgenden am Beispiel des Elements "Methode" erläutert.

Unveränderte Nutzung

- Voraussetzung: Die Unterklasse benötigt genau die gleichen Implementierungen, die in der Methode der Basisklasse bereits programmiert wurden.
- Umsetzung: In diesem Fall wird die Methode für die Unterklasse **nicht** angelegt.
- Folge: Die Unterklasse nutzt die Methodenimplementierung der Basisklasse.
- Beispiel:
  - Die Basisklasse muss für die Ausführung eines Prozesses eine Achse ansteuern.
  - Für die Unterklasse gilt die gleiche Anforderung: die Unterklasse muss ebenfalls die Achse ansteuern.
  - In diesem Fall wird die Methode ExecuteProcess für die Unterklasse **nicht** angelegt. Wenn für eine Instanz der Unterklasse die Methode aufgerufen wird (fbSub.ExecuteProcess(…)), wird automatisch die Basisimplementierung der Methode aufgerufen (FB_Base.ExecuteProcess). Dadurch profitiert die Unterklasse von den Implementierungen, die bereits in der Basisklasse umgesetzt wurden.

Funktionsbaustein FB_Base:

```
FUNCTION_BLOCK FB_Base
VAR
    fbAxis  : FB_Axis;
END_VAR
```

Methode FB_Base.ExecuteProcess:

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling axis module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbAxis.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the axis module
ExecuteProcess := NOT fbAxis.Error;
```

Funktionsbausteinstein FB_Sub:

```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
VAR
END_VAR
```

Methode FB_Sub.ExecuteProcess:

[existiert nicht]

Überschreibung

- Voraussetzung: Die Unterklasse benötigt in der Methode im Vergleich zur Basisklasse komplett andere Anweisungen.
- Umsetzung: In diesem Fall wird die Methode für die Unterklasse angelegt und im Implementierungsteil mit entsprechend anderen Anweisungen gefüllt. Im Vergleich zur Methode der Basisklasse unterscheidet sich nur der Implementierungsteil – der Deklarationsteil muss identisch sein.
- Folge: Die Unterklasse nutzt ihre eigene Implementierung der Methode. Die Unterklasse hat die Methode der Basisklasse überschrieben.
- Beispiel:
  - Die Basisklasse muss für die Ausführung eines Prozesses eine Achse ansteuern.
  - Die Unterklasse muss bei der Prozessausführung hingegen keine Achse, sondern einen Zylinder ansteuern.
  - In diesem Fall wird die Methode ExecuteProcess für die Unterklasse angelegt. Der Implementierungsteil der Methode wird mit den benötigten Anweisungen programmiert, die verglichen mit der Basisimplementierung eine komplett andere Wirkung erzielen.

Funktionsbaustein FB_Base:

```
FUNCTION_BLOCK FB_Base
VAR
    fbAxis  : FB_Axis;
END_VAR
```

Methode FB_Base.ExecuteProcess:

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling axis module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbAxis.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the axis module
ExecuteProcess := NOT fbAxis.Error;
```

Funktionsbausteinstein FB_Sub:

```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
VAR
    fbCylinder  : FB_Cylinder;
END_VAR
```

Methode FB_Sub.ExecuteProcess:

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling cylinder module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbCylinder.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the cylinder module
ExecuteProcess := NOT fbCylinder.Error;
```

Erweiterung

- Voraussetzung: Die Unterklasse benötigt sowohl die Implementierung, wie sie in der Basisklasse bereits umgesetzt wurde, als auch zusätzliche Anweisungen, die spezifisch für die Unterklasse sind.
- Umsetzung: In diesem Fall wird die Methode für die Unterklasse angelegt und im Implementierungsteil mit den zusätzlich benötigten Anweisungen gefüllt. An der gewünschten Stelle innerhalb der Unterklassenmethode wird die Methode der Basisklasse mittels SUPER^.SampleMethod(…) aufgerufen. Mithilfe dieses Aufrufs wird die ursprüngliche Methode der Basisklasse ausgeführt. Durch die zusätzlichen Anweisungen innerhalb der Unterklassenmethode werden zudem weitere Anweisungen ausgeführt, die speziell für die Unterklasse benötigt werden.

- Folge: Die Unterklasse nutzt sowohl ihre eigene, zusätzliche Implementierung als auch die Implementierung der Basisklasse (mittels Aufruf über den SUPER-Zeiger). Die Unterklasse hat die Methode der Basisklasse erweitert.
- Beispiel:
  - Die Basisklasse muss für die Ausführung eines Prozesses eine Achse ansteuern.
  - Die Unterklasse muss bei der Prozessausführung ebenfalls eine Achse ansteuern. **Zusätzlich** muss die Unterklasse allerdings noch einen Zylinder ansteuern.
  - In diesem Fall wird die Methode ExecuteProcess für die Unterklasse angelegt. Der Implementierungsteil der Methode wird mit den benötigten, zusätzlichen Anweisungen programmiert. An geeigneter Stelle im Ablauf der Unterklassenmethode wird die Methode der Basisklasse (FB_Base.ExecuteProcess) mithilfe des SUPER-Zeigers aufgerufen (SUPER^.ExecuteProcess(…)). Dadurch profitiert die Unterklasse von den Implementierungen, die bereits in der Basisklasse umgesetzt wurden – sie kann die Implementierungen aber zusätzlich mit Anweisungen erweitern/spezialisieren, die von der Unterklasse benötigt werden.

Funktionsbaustein FB_Base:

```
FUNCTION_BLOCK FB_Base
VAR
    fbAxis  : FB_Axis;
END_VAR
```

Methode FB_Base.ExecuteProcess:

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess  : BOOL;
END_VAR
```

```
// Calling axis module by passing input parameter "bExecuteProcess" of this method to the input
parameter "bExecute" of method "Execute"
fbAxis.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the axis module
ExecuteProcess := NOT fbAxis.Error;
```

Funktionsbausteinstein FB_Sub:

```
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
VAR
    fbCylinder  : FB_Cylinder;
END_VAR
```

Methode FB_Sub.ExecuteProcess:

```
METHOD ExecuteProcess : BOOL
VAR_INPUT
    bExecuteProcess    : BOOL;
END_VAR
```

```
// Extension: Calling cylinder module by passing input parameter "bExecuteProcess" of this method to
the input parameter "bExecute" of method "Execute"
fbCylinder.Execute(bExecute := bExecuteProcess);

// Setting the return value of this method as inverted error signal of the cylinder module PLUS
calling the base method and analyzing its return value
ExecuteProcess := NOT fbCylinder.Error AND SUPER^.ExecuteProcess(bExecuteProcess :=
bExecuteProcess);
```

7.20.6 Erweitern einer Struktur

Sie können Strukturen ebenso wie Funktionsbausteine erweitern. Die Struktur erhält dann zusätzlich zu den eigenen Variablen die Variablen der Basis-Struktur.

Erzeugen einer Struktur, die eine andere Struktur erweitert:

1. Selektieren Sie das SPS-Projektobjekt oder einen Unterordner im SPS-Projektbaum.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > DUT…**
   ⇨ Der Dialog **DUT hinzufügen** öffnet sich.
3. Geben Sie einen Namen und wählen Sie **Struktur** als Datentyp.

4. Aktivieren Sie die Option **Erweitert** und klicken Sie auf die Schaltfläche ... .
   ⇨ Die **Eingabehilfe** öffnet sich.
5. Wählen Sie aus der Kategorie **Datentypen** die Struktur aus, die durch die neue Struktur erweitert werden soll.

⇨ Die Struktur erweitert die Basis-Struktur.

Mehrfachvererbung nicht erlaubt

Bei Strukturen ist Mehrfachvererbung nicht erlaubt. Es ist nicht möglich, dass eine Struktur mehr als eine andere Struktur erweitert.

- Nicht möglich:
  TYPE ST_Sub EXTENDS ST_Base1, ST_Base2 :
  STRUCT
  …

Siehe auch:

- Objekt DUT [▶ 77]

7.20.7 Erweitern einer Schnittstelle

Sie können Schnittstellen ebenso wie Funktionsbausteine erweitern. Die Schnittstelle erhält dann auch die Schnittstellen-Methoden und Schnittstellen-Eigenschaften der Basis-Schnittstelle, zusätzlich zu ihren eigenen.

Erzeugen einer Schnittstelle, die eine andere Schnittstelle erweitert:

1. Selektieren Sie das SPS-Projektobjekt oder einen Unterordner im SPS-Projektbaum.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Schnittstelle…**
   ⇨ Der Dialog **Schnittstelle hinzufügen** öffnet sich.
3. Geben Sie einen Namen für die neue Schnittstelle ein.
4. Aktivieren Sie die Option **Erweitert** und klicken Sie auf die Schaltfläche ... .
5. Die **Eingabehilfe** öffnet sich.
6. Wählen Sie aus der Kategorie **Schnittstellen** die Schnittstelle aus, die durch die neue Schnittstelle erweitert werden soll.

⇨ Die Schnittstelle erweitert die Basis-Schnittstelle.

Mehrfachvererbung erlaubt

Bei Schnittstellen ist Mehrfachvererbung erlaubt. Es ist möglich, dass eine Schnittstelle mehr als eine andere Schnittstelle erweitert.

- Möglich:
  INTERFACE I_Sub EXTENDS I_Base1, I_Base2

Siehe auch:

- Objekt Schnittstelle [▶ 196]

7.20.8 Implementieren einer Schnittstelle

Die Implementierung von Schnittstellen basiert auf dem Konzept der objektorientierten Programmierung. Über gemeinsame Schnittstellen können Sie verschiedene, aber gleichartige Funktionsbausteine auch gleichartig verwenden.

Ein Funktionsbaustein, der eine Schnittstelle implementiert, muss alle Methoden und Eigenschaften enthalten, die in dieser Schnittstelle definiert sind (Schnittstellen-Methoden und Schnittstellen-Eigenschaften). Das bedeutet: Name, Rückgabetyp, Eingänge und Ausgänge der jeweiligen Methode oder Eigenschaft müssen exakt gleich sein. Wenn sich die Deklaration der Elemente in der Schnittstelle und in dem Funktionsbaustein unterscheiden oder wenn die Schnittstelle über weitere Elemente verfügt, die der Funktionsbaustein nicht enthält, meldet der Compiler einen Fehler.

Weiterführende Informationen zu den Anwendungsfällen einer Schnittstelle sowie ein Beispiel finden Sie im Abschnitt „Objekt Schnittstelle [▶ 196]“

Wenn Sie einen neuen Funktionsbausteins anlegen, der eine Schnittstelle implementiert, fügt TwinCAT automatisch alle Methoden und Eigenschaften dieser Schnittstelle unterhalb des neuen Funktionsbausteins im Baum ein.

Wenn Sie der Schnittstelle danach weitere Methoden oder Eigenschaften hinzufügen, fügt TwinCAT diese Elemente nicht automatisch auch in den betreffenden Funktionsbausteinen hinzu. Für die Aktualisierung müssen Sie explizit den Befehl **Schnittstellen implementieren** auswählen.

Bei Ausführung dieses Befehls werden die automatisch angelegten Methoden oder Eigenschaften mit einem Pragmaattribut versehen, welches Übersetzungsfehler oder -warnungen provoziert. Dadurch werden Sie dahingehend unterstützt, dass automatisch angelegte Elemente nicht unbeabsichtigt leer bleiben. Weitere Informationen hierzu entnehmen Sie bitte der Hilfeseite zum Befehl Schnittstellen implementieren [▶ 1112].

Implementieren einer Schnittstelle in einen neuen Funktionsbaustein

✓ Das aktuell geöffnete Projekt besitzt mindestens ein Schnittstellen-Objekt.

1. Selektieren Sie das SPS-Projektobjekt oder einen Unterordner im SPS-Projektbaum und wählen Sie im Kontextmenü den Befehl **Hinzufügen > POU…**

   ⇨ Der Dialog **POU hinzufügen** öffnet sich.
2. Geben Sie einen Namen für den neuen Funktionsbaustein in das Eingabefeld **Name** ein, zum Beispiel „FB_SampleImp“.
3. Wählen Sie Funktionsbaustein.
4. Wählen Sie **Implementiert** und klicken Sie auf die Schaltfläche  .
5. Wählen Sie in der Eingabehilfe aus der Kategorie **Schnittstellen** die Schnittstelle zum Beispiel „I_Itf1“ und klicken Sie auf **OK**.
6. Um eine weitere Schnittstellen einzufügen, klicken Sie erneut auf  und wählen Sie eine weitere Schnittstelle aus.
7. Optional können Sie einen **Zugriffsmodifizierer** für den neuen Funktionsbaustein aus der Auswahlliste auswählen.
8. Wählen Sie aus der Auswahlliste **Implementierungsprache** zum Beispiel „Strukturierter Text (ST)“ aus.
9. Klicken Sie auf **Öffnen**.

⇨ TwinCAT fügt den Funktionsbaustein FB_SampleImp in den SPS-Projektbaum ein und der Editor öffnet sich. In der ersten Zeile steht:
FUNCTION_BLOCK FB_SampleImp IMPLEMENTS I_Itf1
Die Methoden und Eigenschaften der Schnittstelle sind jetzt im SPS-Projektbaum unter dem Funktionsbaustein eingefügt und Sie können nun Programmcode im Implementierungsteil der Methoden und Eigenschaften eingeben.

Implementieren einer Schnittstelle in einen bestehenden Funktionsbaustein

✓ Das aktuell geöffnete Projekt besitzt einen Funktionsbaustein, zum Beispiel „FB_SampleImp“ und mindestens ein Schnittstellen-Objekt, zum Beispiel „I_Itf1“.

1. Doppelklicken Sie im SPS-Projektbaum auf die POU FB_SampleImp.

   ⇨ Der Editor der POU öffnet sich.
2. Erweitern Sie den bestehenden Eintrag der obersten Zeile FUNCTION_BLOCK FB_SampleImp mit IMPLEMENTS I_Itf1.

   ⇨ Der Funktionsbaustein FB_SampleImp implementiert die Schnittstelle I_Itf1.

3. Um die Elemente (Methoden oder Eigenschaften), die in der Schnittstelle definiert sind, aber im Funktionsbaustein noch nicht vorhanden sind, im Funktionsbaustein automatisch erzeugen zu lassen, können Sie den Befehl **Schnittstellen implementieren** ausführen, der im Kontextmenü des Funktionsbausteins im Projektbaum verfügbar ist.

⇨ Die Methoden und Eigenschaften der Schnittstelle sind jetzt im SPS-Projektbaum unter dem Funktionsbaustein eingefügt und Sie können nun Programmcode im Implementierungsteil der Methoden und Eigenschaften eingeben.

Siehe auch:

- Objekt Funktionsbaustein [▶ 180]
- Dokumentation TC3 User Interface: Befehl Schnittstellen implementieren [▶ 1112]

7.20.9 Methodenaufruf

Um einen Methodenaufruf zu implementieren, werden den Schnittstellenvariablen die tatsächlichen Parameter (Argumente) übergeben. Dabei kann alternativ auf die Parameternamen verzichtet werden.

Je nach deklariertem Zugriffsmodifizierer kann eine Methode nur innerhalb des eigenen Namensraums (INTERNAL), nur innerhalb des eigenen Programmierbausteins und seinen Ableitungen (PROTECTED) oder nur innerhalb des eigenen Programmierbausteins (PRIVATE) aufgerufen werden. Bei PUBLIC kann die Methode überall aufgerufen werden.

Innerhalb der Implementierung kann eine Methode sich selbst rekursiv aufrufen: entweder direkt mit Hilfe des THIS-Pointers oder mit Hilfe einer lokalen Variablen für den zugeordneten Funktionsbaustein.

Methodenaufruf als virtueller Funktionsaufruf

Durch Vererbung kann es zu virtuellen Funktionsaufrufen kommen. Virtuelle Funktionsaufrufe ermöglichen, dass derselbe Aufruf in einem Programm-Quellcode während der Laufzeit verschiedene Methoden aufruft.

In folgenden Fällen wird der Methodenaufruf dynamisch gebunden:

- Sie rufen eine Methode über einen Pointer auf einen Funktionsbaustein auf (zum Beispiel pFB^.SampleMethod).
  Der Pointer kann in dieser Situation auf Instanzen vom Typ des Funktionsbausteins und auf Instanzen von allen abgeleiteten Funktionsbausteinen zeigen.
- Sie rufen eine Methode einer Schnittstellen-Variablen auf (zum Beispiel iSample.SampleMethod).
  Die Schnittstelle kann auf alle Instanzen von Funktionsbausteinen verweisen, die diese Schnittstelle implementieren.
- Eine Methode ruft eine andere Methode desselben Funktionsbausteins auf. Die Methode kann in dem Fall auch die Methode eines erweiterten Funktionsbausteins mit gleichem Namen aufrufen.
- Der Aufruf einer Methode erfolgt über eine Referenz auf einen Funktionsbaustein. Die Referenz kann in dieser Situation auf Instanzen vom Typ des Funktionsbausteins und auf Instanzen von allen abgeleiteten Funktionsbausteinen zeigen.
- Sie weisen VAR_IN_OUT-Variablen eines Basis-Funktionsbaustein-Typen einer Instanz eines abgeleiteten FB-Typen zu.
  Die Variable kann in dieser Situation auf Instanzen vom Typ des Funktionsbausteins und auf Instanzen von allen abgeleiteten Funktionsbausteinen zeigen.

Beispiel

- Die Funktionsbausteine FB_Sub1 und FB_Sub2 erweitern jeweils den Funktionsbaustein FB_Base.
- FB_Base implementiert die Schnittstelle I_Base, die die Methode Method1 definiert.
- FB_Base und FB_Sub1 stellen jeweils die Methode Method1 zur Verfügung. FB_Sub1 überschreibt bzw. erweitert somit die Methode der Basisklasse FB_Base.
- FB_Sub2 stellt die Methode nicht zur Verfügung. Der Funktionsbaustein nutzt unverändert die Methode der Basisklasse FB_Base.

- In dem Programm MAIN wird einer Schnittstellenvariablen und einer Referenz auf die Basisklasse jeweils eine Instanz der Basisklasse FB_Base, eine Instanz der Unterklasse FB_Sub1 oder eine Instanz der Unterklasse FB_Sub2 zugewiesen. Welche Instanz zugewiesen wird, hängt von dem Wert der Variablen nVar ab.
- Über die Schnittstellenvariable und über die Referenzvariable wird die Methode Method1 aufgerufen. Dieser Methodenaufruf ist dynamisch und kann für unterschiedliche Instanzen (fbBase, fbSub1, fbSub2) ausgeführt werden, wobei sich die dahinterstehenden Methodenimplementierungen zwischen den Instanzen unterscheiden.
  - Für die Instanz fbBase wird die Implementierung von FB_Base.Method1 ausgeführt.
  - Für die Instanz fbSub1 wird die Implementierung von FB_Sub1.Method1 ausgeführt, da FB_Sub1 die Methode der Basisklasse überschreibt bzw. erweitert.
  - Für die Instanz fbSub2 wird die Implementierung von FB_Base.Method1 ausgeführt, da die Unterklasse FB_Sub2 die Methode der Basisklasse weder überschreibt noch erweitert, sondern unverändert nutzt.

Schnittstelle I_Base mit der Methode Method1:

```
INTERFACE I_Base
METHOD Method1
```

Funktionsbaustein FB_Base mit der Methode Method1:

```
FUNCTION_BLOCK FB_Base IMPLEMENTS I_Base
METHOD Method1
```

Funktionsbaustein FB_Sub1 mit der Methode Method1:

```
FUNCTION_BLOCK FB_Sub1 EXTENDS FB_Base
METHOD Method1
```

Funktionsbaustein FB_Sub2 ohne eigene Methode:

```
FUNCTION_BLOCK FB_Sub2 EXTENDS FB_Base
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    nVar       : INT;
    fbBase     : FB_Base;
    fbSub1     : FB_Sub1;
    fbSub2     : FB_Sub2;
    iBase      : I_Base;
    refBase    : REFERENCE to FB_Base;
END_VAR
(* Choosing the desired instances via value of nVar:
    0 => fbBase
    1 => fbSub1
    2 => fbSub2 *)

IF nVar = 0 THEN
    iBase     := fbBase;
    refBase REF= fbBase;

ELSIF nVar = 1 THEN
    iBase     := fbSub1;
    refBase REF= fbSub1;

ELSIF nVar = 2 THEN
    iBase     := fbSub2;
    refBase REF= fbSub2;
END_IF

// Regarding each of the following two calls via interface and via reference:
    // If nVar is 0, FB_Base.Method1 will be called for instance fbBase
    // If nVar is 1, FB_Sub1.Method1 will be called for instance fbSub1
    // If nVar is 2, FB_Base.Method1 will be called for instance fbSub2

iBase.Method1();
refBase.Method1();
```

Zusätzliche Ausgänge

Gemäß der Norm IEC 61131-3 können Methoden sowie normale Funktionen zusätzliche Ausgänge deklariert haben. Beim Methodenaufruf weisen Sie den zusätzlichen Ausgängen Variablen zu.

Genaue Informationen hierzu finden Sie unter Objekt Funktion [▶ 83].

Methode rekursiv aufrufen

HINWEIS

Verwenden Sie Rekursionen vorwiegend zur Bearbeitung von rekursiven Datentypen wie beispielsweise verketteten Listen. Allgemein ist es ratsam, bei der Verwendung von Rekursion vorsichtig zu sein. Bei einer unerwartet tiefen Rekursion kann es zu einem Stacküberlauf und damit zu einem Maschinenstillstand kommen.

Innerhalb ihrer Implementierung kann eine Methode sich selbst aufrufen:

- direkt mit Hilfe des THIS-Pointers
- indirekt mit Hilfe einer lokalen Funktionsbaustein-Instanz des Basisfunktionsbausteins

Üblicherweise wird bei einem solchen rekursiven Aufruf eine Compilerwarnung ausgegeben. Wenn die Methode mit dem Pragma {attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'} versehen ist, wird die Compilerwarnung unterdrückt. Im Kapitel Attribut 'estimated-stack-usage' [▶ 837] finden Sie ein Implementierungsbeispiel.

Siehe auch:

- Objekt Methode [▶ 182]
- Objekt Funktion [▶ 83]
- Erweitern eines Funktionsbausteins [▶ 201]
- Referenz Programmierung: SUPER [▶ 728]
- Referenz Programmierung: THIS [▶ 730]

7.20.10 ABSTRACT-Konzept

Das Schlüsselwort ABSTRACT ist für Funktionsbausteine, Methoden und Eigenschaften verfügbar. Es ermöglicht die Implementierung eines SPS-Projektes mit Abstraktionsebenen.
Die Abstraktion ist ein Schlüsselkonzept der Objektorientierten Programmierung. Verschiedene Abstraktionsebenen enthalten dabei allgemeine oder spezifische Implementationsaspekte.

Verfügbar ab TC3.1 Build 4024

Anwendung der Abstraktion

Es bietet sich an, Basisfunktionen oder Gemeinsamkeiten verschiedener Klassen in einer abstrakten Basisklasse zu implementieren. Spezifische Aspekte implementieren Sie in nicht-abstrakten Unterklassen. Das Prinzip ähnelt damit der Verwendung einer Schnittstelle. Schnittstellen entsprechen rein abstrakten Klassen, die nur abstrakte Methoden und Eigenschaften enthalten. Eine abstrakte Klasse kann auch nicht-abstrakte Methoden und Eigenschaften enthalten.

Regeln für die Nutzung des Schlüsselworts ABSTRACT

- Abstrakte Funktionsbausteine können nicht instanziiert werden.
- Abstrakte Funktionsbausteine können abstrakte und nicht-abstrakte Methoden und Eigenschaften enthalten.
- Abstrakte Methoden oder Eigenschaften enthalten keine Implementierung (nur die Deklaration).
- Wenn ein Funktionsbaustein eine abstrakte Methode oder Eigenschaft enthält, muss er selbst auch abstrakt sein.

- Abstrakte Funktionsbausteine müssen erweitert werden, um die abstrakten Methoden oder Eigenschaften implementieren zu können.
- Daraus folgt: Ein abgeleiteter FB muss die Methoden/Eigenschaften seines Basis-FBs implementieren oder er muss ebenfalls als abstrakt definiert werden.

Beispiel

Abstrakte Basisklasse:

```
FUNCTION_BLOCK ABSTRACT FB_System_Base
```

In dieser abstrakten Basisklasse werden die Gemeinsamkeiten aller Systemmodule implementiert. Dafür enthält sie die nicht-abstrakte Eigenschaft „nSystemID“ und die abstrakte Methode „Execute“:

```
PROPERTY nSystemID : UINT
```

```
METHOD ABSTRACT Execute
```

Während die Implementierung von „nSystemID“ für alle Systeme gleich ist, unterscheidet sich die Implementierung der Methode „Execute“ für die einzelnen Systeme.

Nicht-abstrakte Unterklasse:

```
FUNCTION_BLOCK FB_StackSystem EXTENDS FB_System_Base
```

Für die spezifischen Systeme werden nicht-abstrakte Klassen implementiert, die von der Basisklasse abgeleitet werden. Diese Unterklasse repräsentiert einen Stack. Da sie nicht abstrakt ist, muss sie die Methode „Execute“ implementieren, die die spezifische Stack-Ausführung definiert:

```
METHOD Execute
```

7.20.11 Beispiele

Basis-OOP-Beispiel

Beschreibung	Dieses SPS-Beispiel zeigt einige Basisfunktionalitäten der objektorientierten Programmierung (OOP). Die enthaltenen Elemente/Funktionalitäten sind: • Funktionsbausteine (FB) • Methoden • Properties • FB-Vererbung/-Erweiterung • Interface (ITF) -Implementierung und -Verwendung
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644034443.zip
Weiterführende Informationen	In der Dokumentation PLC: Objektorientiert programmieren

Erweitertes-OOP-Beispiel

Beschreibung	Dieses SPS-Beispiel enthält ein objektorientiertes Programm zur Steuerung einer Sortieranlage. Die Anwendung kann über eine integrierte Visualisierung gesteuert werden.
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644036107.zip
Weiterführende Informationen	In der Dokumentation PLC: Objektorientiertes Programm zur Steuerung einer Sortieranlage

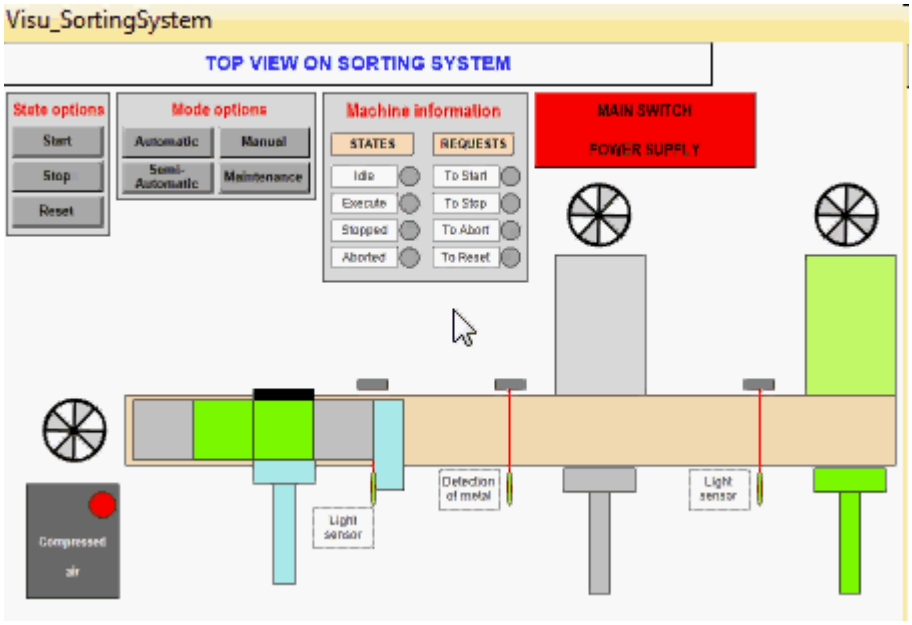
7.20.11.1 Objektorientiertes Programm zur Steuerung einer Sortieranlage

Die Programmiermittel der Objektorientierung sind sehr vielseitig und können auf unterschiedlichste Weise angewendet werden. Das folgende Beispiel verdeutlicht die Vorteile der Objektorientierung und zeigt einige Ideen auf, wie die Objektorientierung bei der SPS-/Maschinenprogrammierung verwendet werden kann. Das

bei diesem Beispiel vorgestellte objektorientierte Konzept und die dahinter stehenden Ansätze erheben keinen Anspruch auf Vollständigkeit und können nicht pauschal auf jede beliebige Applikation übertragen werden.

Das Beispielprogramm zur Veranschaulichung der objektorientierten Programmierung (OOP) dient zur Steuerung einer Sortieranlage, die Metall- und Plastikkästen gemäß ihrer Materialart sortiert. Dabei beinhaltet der Sortiervorgang eine Vereinzelung und zwei Ausschleusungen. Diese Vorgänge werden mit Hilfe von Zylindern realisiert, von denen unterschiedlich komplexe Ausführungen existieren. Als Beispielanforderung gilt, dass es möglich sein soll, die an der Anlage eingesetzten Zylinder flexibel durch einen anderen Zylindertypen auszutauschen.

Anhand eines Sensors zur Metallerkennung kann die Anlage die verschiedenen Materialarten unterscheiden, sodass Metall- (grau) und Plastikkästen (grün) auf unterschiedlichen Nebenförderbändern abtransportiert werden. Das folgende Video verdeutlicht den Sortiervorgang der Anlage, die von oben zu sehen ist.

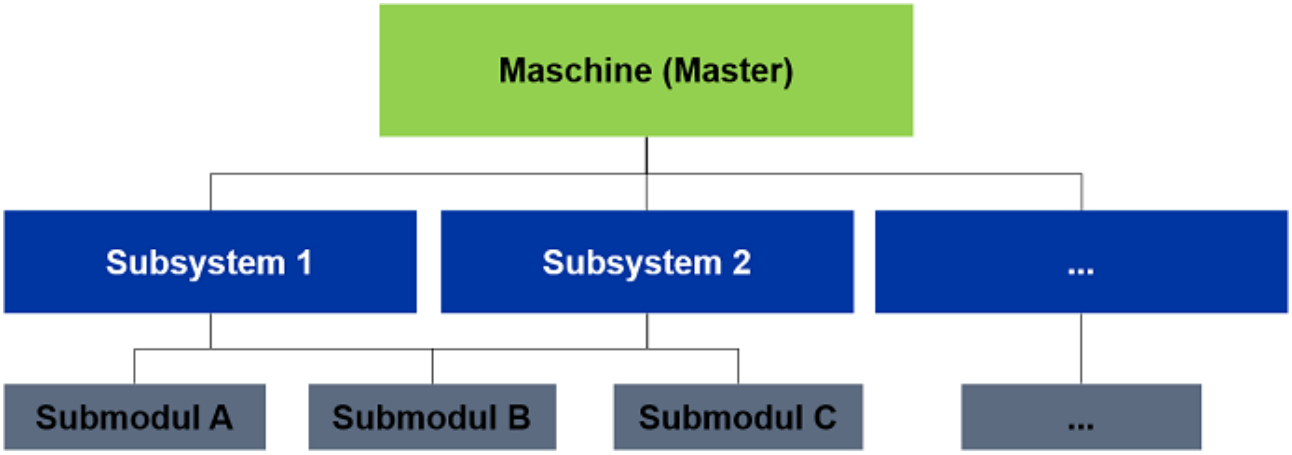


Generelles Konzept der OOP

Ein allgemeines Ziel der objektorientierten Programmierung ist es beispielsweise, Automatisierungsmodule zu entwickeln. Bei diesen Objekten handelt es sich um applikationsneutrale, vorgefertigte Funktionseinheiten, die nur einmal entwickelt werden, aber mehrfach unverändert verwendet werden können. Automatisierungsmodule werden demnach nicht für eine bestimmte Anlage entwickelt, sondern stellen allgemein eine Anlagenfunktionalität zur Verfügung, die i.d.R. in mehreren Anlagen zum Einsatz kommt. So wird beispielsweise eine Klasse „FB_Axis“ zur Steuerung einer Achse einmal entwickelt und in mehreren Anlagen instanziiert und verwendet.

Mit Hilfe von Automatisierungsmodulen wird vor allem der Implementierungsaufwand bei der Programmierung einer Anlage erheblich reduziert. Gleichzeitig sind die Qualität und die Zuverlässigkeit der verwendeten Objekte vergleichsweise hoch, da die Module auch in anderen Anlagen eingesetzt und damit getestet und ggf. verbessert werden. Sind der Programmierstil und das Konzept bei der Implementierung von verschiedenen Modulen identisch, kann ein einheitliches „Look and Feel“ für die Verwendung der Objekte geschaffen werden. Dadurch kann eine Applikationsstandardisierung erreicht werden.

Für eine sinnvolle Entwicklung und Verwendung von Automatisierungsmodulen werden eine modulare Konzeption und eine modulare Programmierung der Anlage vorausgesetzt. Die Unterteilung der Anlage in Objekte kann beispielsweise so konzipiert werden, dass die Anlage aus verschiedenen Subsystemen besteht und diese Subsysteme wiederum verschiedene Submodule beinhalten. Die Maschine, ihre Subsysteme und deren Submodule werden als Automatisierungsmodule implementiert und agieren somit als separate, autarke Objekte.



Des Weiteren ist es denkbar, die Daten und Funktionalitäten, die alle Submodule gemeinsam haben, in einer Submodul-Basisklasse zu generalisieren. Dieser Ansatz ist durch die Entwicklung einer Subsystem-Basisklasse ebenfalls auf die Subsysteme übertragbar, falls Gemeinsamkeiten zwischen den Subsystemen bestehen. Dadurch würde zum einen der Programmieraufwand erheblich reduziert und zum anderen müsste bei Änderungsbedarf der gemeinsamen Implementierungen nur die Basisklasse angepasst werden.

Sortieranlage – Vorgehen

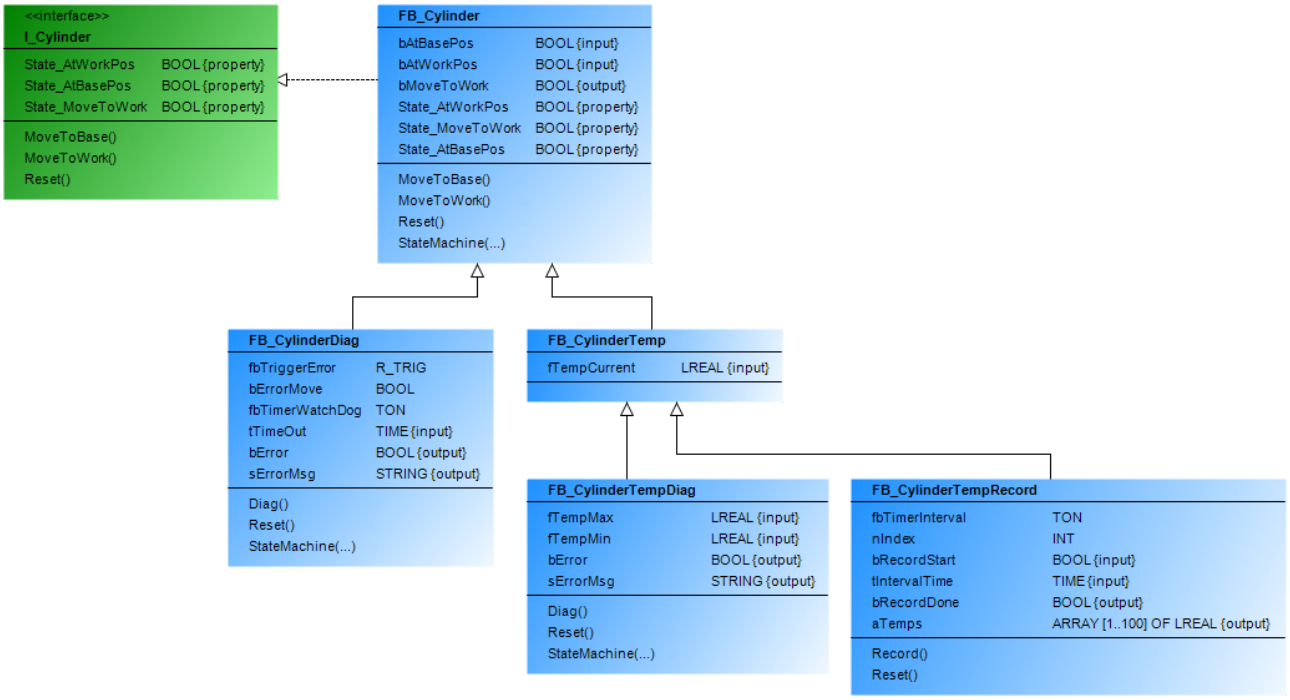
Die Programmierung der Sortieranlage unter Verwendung der Objektorientierung wird im Folgenden ausführlich an dem Beispiel der Zylinder erläutert. Die Implementierung der anderen Submodule, wie den Antrieben und den Sensoren, lässt sich aus der Programmierweise der Zylinder ableiten. Zudem wird die Zusammenfassung von Submodulen zu Subsystemen am Ende kurz beschrieben. Bei der Implementierung der Zylinder als Submodule wird folgendermaßen vorgegangen:

- Planung der Softwarestruktur
- Implementierung des Softwarekonzepts
- Instanziierung der Programmelemente
- Zuweisung von Funktionsblockinstanzen zu einer Interfaceinstanz
- Verwendung einer Funktionsblockinstanz über eine Interfaceinstanz
- Weiteres Softwarekonzept: Zusammenfassung von Submodulen zu Subsystemen

Planung der Softwarestruktur

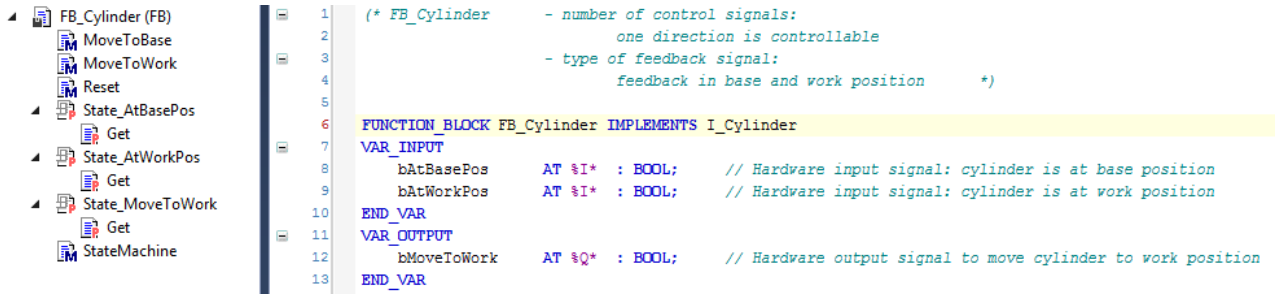
Die geforderten Zylindertypen umfassen einfache Zylinder, die lediglich in Grund- und Arbeitsstellung verfahren können, und Zylinder mit zusätzlichen Funktionalitäten. Diese komplexeren Zylinder können das Erreichen ihrer Endlagen überwachen, ihre Temperatur aufzeichnen oder diagnostizieren, wenn die Temperatur nicht mehr in einem vorgegebenen Intervall liegt. Die einzelnen Zylinder der Maschine sollen nicht auf einen bestimmten Zylindertyp beschränkt sein, sondern durch andere Typen ausgetauscht werden können. Auf diese Weise soll an der Anlage beispielsweise ein einfacher Zylinder durch einen Zylinder mit Zusatzfunktionalität ersetzt werden können.
Unabhängig davon, über welche Zusatzfunktion sie verfügen, müssen alle Zylinder die Grundfunktionalitäten des Ein- und Ausfahrens ausführen können. Da es sich hierbei um eine Anforderung handelt, die im Sinne der Applikation an jeden Zylinder gestellt wird, werden die zu der Anforderung gehörigen Methoden und Properties in einer Schnittstelle definiert. In einer Schnittstelle sind dabei keine Implementierungen, sondern nur die Definition der Methoden und Properties zu finden. Sie stellt somit lediglich eine Vereinbarung für die Funktionsblöcke dar, die die Schnittstelle einbinden, und wird erfüllt, wenn die definierten Methoden und Properties von den Bausteinen zur Verfügung gestellt werden. In den einbindenden Funktionsbausteinen werden letztlich die Implementierungen der in der Schnittstelle definierten Programmelemente umgesetzt. Durch die Definition des Interfaces *I_Cylinder* und dessen Einbindung in den Zylinderbausteinen ist somit sichergestellt, dass die Zylinder den gestellten Anforderungen entsprechen und über die dazugehörigen Programmelemente zum Verfahren verfügen. Des Weiteren ist es durch die Definition und Einbindung einer Schnittstelle möglich, Bausteininstanzen über Interfaceinstanzen anzusprechen. Auf diese Weise ergibt sich die geforderte Austauschbarkeit der Zylinder.
Da alle Zylinder die Grundfunktionalitäten des Ein- und Ausfahrens bereitstellen müssen, ist es neben der Definition einer „Schnittstellen-Vereinbarung“ außerdem sinnvoll, diese Funktionalitäten zu generalisieren und ihre Implementierung über eine Basisklasse zur Verfügung zu stellen. Aus diesem Grund wird der

Zylinder *FB_Cylinder* geplant, der Methoden zur Zylinderbewegung besitzt und die Basisklasse aller geforderten Zylinder darstellt. Dadurch verfügen alle abgeleiteten Klassen über die Grundfunktionalitäten des Ein- und Ausfahrens, obwohl diese nur einmal in dem Basiszylinder implementiert werden.
Von der Oberklasse *FB_Cylinder* wird der Funktionsblock *FB_CylinderDiag* abgeleitet, der zusätzlich das Erreichen der Endlagen überwacht und den Basiszylinder um diese Funktionalität erweitert. Er stellt die Lösung eines der geforderten Zylinder dar.
Da die Zylinder mit Aufzeichnung bzw. Überwachung der Temperatur den Wert der aktuellen Temperatur benötigen, wird von der Basisklasse *FB_Cylinder* ein Funktionsblock mit dieser Komponente abgeleitet. Durch den Zusatz der Temperatur heißt die Klasse *FB_CylinderTemp* und stellt die Oberklasse für die Zylinder mit Temperaturaufzeichnung und -überwachung dar. Diese beiden Unterklassen mit den Namen *FB_CylinderTempDiag* und *FB_CylinderTempRecord* werden um ihr spezifisches Verhalten und die Variablen erweitert, die für diese Funktionalitäten benötigt werden.



Implementierung des Softwarekonzepts

Die geplanten Funktionsblöcke können nun implementiert werden. Um das Vorgehen aufzuzeigen, werden nachfolgend Teile der Basisklasse *FB_Cylinder* und der abgeleiteten Klasse *FB_CylinderDiag* beschrieben. Der Funktionsblock *FB_Cylinder* bindet das Interface *I_Cylinder* mit Hilfe des Schlüsselworts IMPLEMENTS ein, wodurch die Methoden und Properties der Schnittstelle in der Oberklasse automatisch erstellt werden können. Dadurch ist garantiert, dass der Funktionsblock und seine Ableitungen über die vom Interface geforderten Elemente verfügen. Die durch das Interface bedingte Bausteinstruktur und die Deklaration des Funktionsblocks *FB_Cylinder* sind in folgender Abbildung dargestellt.

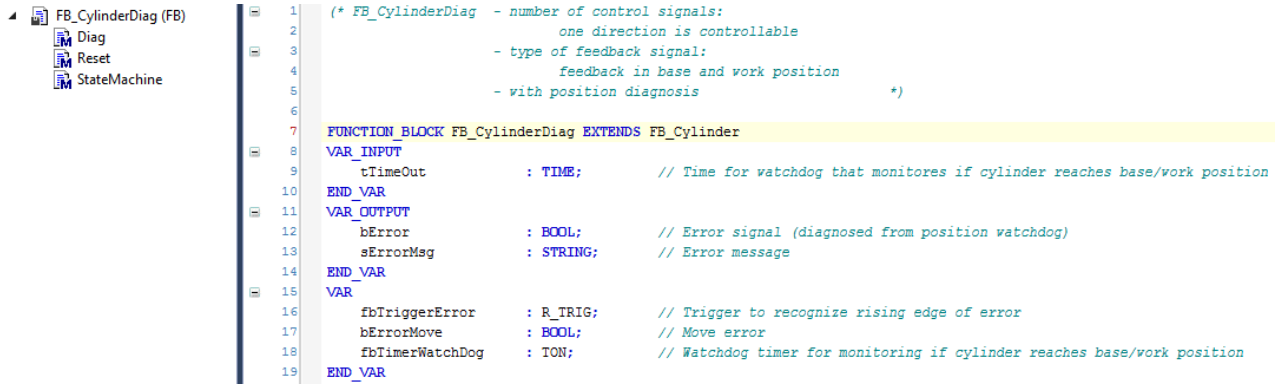


Die Methode *Reset*, die den Zylinderausgang *bMoveToWork* des Funktionsblocks zurücksetzt, ist beispielhaft aufgeführt.

```
// ======================================================
// *** Method Reset of FB_Cylinder ***
```

```
    bMoveToWork := FALSE;

// ========================================================
```

Mit Hilfe des Schlüsselworts EXTENDS wird der Funktionsblock *FB_CylinderDiag* zu einer Ableitung von *FB_Cylinder*. Dadurch können (abhängig vom Zugriffsmodifizierer) die Variablen, Methoden und Properties der Basisklasse genutzt werden. Da die Unterklasse die Methoden *Reset* und *StateMachine* der Basisklasse erweitern soll, werden die Methoden in der Unterklasse eingefügt und können so verändert werden. Außerdem wird die zusätzliche Methode *Diag* integriert, über die die Basisklasse nicht verfügt. Die für die Diagnose-Funktionalität benötigten zusätzlichen Variablen werden in *FB_CylinderDiag* deklariert. Die Struktur und die Deklaration von *FB_CylinderDiag* sehen wie folgt aus:



In den Methoden *Reset* und *StateMachine* der abgeleiteten Klasse können die gleichnamigen Methoden der Oberklasse erweitert oder überschrieben werden. Um eine Methodenerweiterung zu erhalten, wird die Methode der Basisklasse aufgerufen, indem das Schlüsselwort SUPER verwendet wird. Bei SUPER handelt es sich um einen Funktionsblockzeiger, der auf die Funktionsblockinstanz der Basisklasse zeigt. Durch weitere Anweisungen kann in der Methode der Unterklasse das Verhalten der Basisklasse erweitert werden, wodurch die Methode für den Zylinder *FB_CylinderDiag* angepasst wird. Die beispielhafte Methode *Reset* von *FB_CylinderDiag*, die die gleichnamige Methode der Basisklasse *FB_Cylinder* durch weitere Anweisungen erweitert, ist nachfolgend aufgeführt.

```
// ========================================================
// *** Method Reset of FB_CylinderDiag ***

    // Calling method Reset of base class FB_Cylinder via 'SUPER^.'
    SUPER^.Reset();

    // Reset error
    bError     := FALSE;
    sErrorMsg  := '';

// ========================================================
```

Instanziierung der Programmelemente

Die mit Hilfe von Vererbung und den entsprechenden Schlüsselwörtern erstellten Funktionsblöcke, die Zylinder mit verschiedener Funktionalität repräsentieren, können nun instanziiert werden.
Damit ein Zylinder als variabel angesehen und durch jeden Zylindertyp dargestellt werden kann, werden die Funktionsblöcke *FB_Cylinder, FB_CylinderDiag, FB_CylinderTemp, FB_CylinderTempDiag* und *FB_CylinderTempRecord* je einmal instanziiert. Die Interfaceinstanz *iCylinder* ist das Objekt, das zur Laufzeit auf eine der Zylinderbausteininstanzen und damit auf den aktuell gewählten Zylindertyp verweist.
Die Variablen *bCylinderDiag, bCylinderTemp* und *bCylinderRecord* können durch den Benutzer verändert werden und geben an, ob der Zylinder über die Diagnose- bzw. die Temperaturfunktionalitäten verfügen soll.
Die Instanziierung der Funktionsblöcke und des Interfaces sowie die drei booleschen Variablen sind nachfolgend dargestellt.

```
// ===== Variables to enable/disable diagnosis and temperature mode ===
   bCylinderDiag         : BOOL;                  // If true the cylinder has diagnosis
functionality
   bCylinderTemp         : BOOL;                  // If true the cylinder has temperature
functionality
   bCylinderRecord       : BOOL;                  // If true the cylinder has recording
functionality

// ===== Function block instances for cylinder ========================
   fbCylinder            : FB_Cylinder;           // Without diagnosis and temperature mode
```

```
   fbCylinderDiag        : FB_CylinderDiag;        // With diagnosis of states
   fbCylinderTemp        : FB_CylinderTemp;        // With temperature mode
   fbCylinderTempDiag    : FB_CylinderTempDiag;    // With diagnosis of temperature
   fbCylinderTempRecord  : FB_CylinderTempRecord;  // With record of temperatures

// ===== Interface instance for cylinder ==============================
   iCylinder             : I_Cylinder;             // Interface for flexible access to cylinder FBs
```

Zuweisung von Funktionsblockinstanzen zu einer Interface-Instanz

Bei einem gewünschten Austausch des Zylinders müssen nun lediglich die Variablen *bCylinderDiag, bCylinderTemp* und *bCylinderRecord* angepasst werden, da der Interface-Instanz *iCylinder* je nach Zustand dieser drei Variablen die entsprechende Funktionsblockinstanz zugewiesen wird.
Soll der Zylinder beispielsweise über die Funktionalität der Temperaturüberwachung verfügen, besitzen *bCylinderDiag* und *bCylinderTemp* den Wert TRUE und *bCylinderRecord* den Wert FALSE. Der gewünschte Funktionsblock ist demnach *FB_CylinderTempDiag* und die dazugehörige Instanz *fbCylinderTempDiag* wird der Interface-Instanz zugewiesen. Die für diese Klasse spezifischen Ausgänge *bError* und *sErrorMsg* werden separat abgefangen, indem sie lokalen Variablen zugewiesen werden. Dadurch, dass die FB-Instanz *fbCylinderTempDiag* der Interface-Instanz *iCylinder* zugewiesen wird, kann über die Interface-Instanz auf die Funktionsblockinstanz mit Temperaturüberwachung zugegriffen werden.
Besitzen die booleschen Variablen hingegen andere Werte, wird der Interface-Instanz entsprechend eine andere Funktionsblockinstanz zugewiesen. Zwei Beispiele der Funktionsblockzuweisung sind nachfolgend dargestellt.

```
// ========================================================
// Selecting cylinder by checking variables to enable / disable diagnosis and temperature mode

   IF bCylinderDiag THEN
       IF bCylinderTemp THEN
           // =========== FB with diagnosis and temperature mode ==================
           bError     := fbCylinderTempDiag.bError;              // Assigning output variable of
chosen FB to local variable
           sErrorMsg  := fbCylinderTempDiag.sErrorMsg;
           iCylinder  := fbCylinderTempDiag;                     // Assigning chosen FB instance to
interface instance

       ELSE
           // =========== FB with diagnosis and without temperature mode ===========
           fbCylinderDiag.tTimeOut  := tTimeOutCylinder;         // Setting special data for
selected FB
           bCylError                := fbCylinderDiag.bError;    // Assigning output variable of
chosen FB to local variable
           sCylErrorMsg             := fbCylinderDiag.sErrorMsg;
           iCylinder                := fbCylinderDiag;           // Assigning chosen FB instance to
interface instance

       …

   END_IF

// ========================================================
```

Verwendung einer Funktionsblockinstanz über eine Interface-Instanz

Die Interface-Instanz, der eine der FB-Instanzen zugewiesen wurde, kann ihre Methoden mit Hilfe der Punktnotation aufrufen. Bei diesem Methodenaufruf über die Interface-Instanz wird, aufgrund des Interfacepointers, die Methode der Funktionsblockinstanz aufgerufen, auf die das Interface verweist.
Im folgenden Programmausschnitt ist dargestellt, wie der gewählte Zylinderbaustein über die Interfaceinstanz in Grund- bzw. in Arbeitsstellung verfahren wird.

```
// ========================================================
// Manual cylinder control (Calling methods of FB via interface instance)

   // Cylinder to work position
   IF fbButtonCylToWork.bOut THEN
       iCylinder.MoveToWork();
   // Cylinder to base position
   ELSIF fbButtonCylToBase.bOut THEN
       iCylinder.MoveToBase();
   END_IF

// ========================================================
```

Weiteres Softwarekonzept: Zusammenfassung von Submodulen zu Subsystemen

Die vorgestellte Implementierung von Funktionsblöcken und die Verwendung ihrer Instanzen mit Hilfe einer Interface-Instanz stellt die objektorientierte Programmierung im Kleinen dar. Zur optimalen Anwendung der Objektorientierung wird diese um die OOP im Großen ergänzt. Dabei werden z. B. verschiedene Submodule zu Subsystemen zusammengefasst.
Bei der Sortieranlage bietet es sich an, einen Sensor zur Materialerkennung, einen Antrieb, der die Bewegung eines Nebenförderbands regelt, und einen Ausschleusungszylinder zu einem Ausschleusungsmodul zusammenzufassen. Auf diese Weise können aus dieser Klasse beliebig viele Objekte bzw. Sortiermodule instanziiert werden, wobei nur ein Ausschleusungsmodul programmiert wird. Dadurch können auf einfachste Weise verschiedene Anlagen umgesetzt werden, die unterschiedlich viele Sortiermodule besitzen und auf andere Materialarten spezialisiert sind (z. B. Metall, Kunststoff, Folie, Glas…).
Ein anderes Subsystem der Sortieranlage stellt die Vereinzelung dar, die aus einem Sensor zur Boxerkennung, einem Antrieb zur Bewegung der Boxen auf dem Hauptförderband und zwei Zylindern zur Umsetzung der Vereinzelung besteht.
Da die beiden Subsysteme der Vereinzelung und der Ausschleusung über Gemeinsamkeiten verfügen, werden die Subsystem-spezifischen Daten und Funktionalitäten in einer Subsystem-Basisklasse zusammengefasst. Die Vereinzelung und die Ausschleusung werden daraufhin als Unterklasse dieses Funktionsblocks angelegt, wodurch sie die generalisierten Programmelemente von der Subsystem-Basisklasse erben.

Durch die Entwicklung von Submodulen und Subsystemen werden die objektorientierten Programmiermöglichkeiten und -vorteile auf verschiedenen Ebenen eingesetzt und genutzt.

TC3.1-Sourcen

Die gesamten TC3.1-Sourcen zu dem Beispielprogramm können hier entpackt werden: https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644036107.zip

Um das Beispielprogramm zu starten:

- Aktivieren Sie die TwinCAT Konfiguration und starten Sie TwinCAT im Run Mode
- Loggen Sie beide PLCs ein und starten Sie diese (**TC3_SortingSystem_PLC** und **TC3_SortingSystem_Simu**)
- Bedienen Sie die Applikation über die Maschinenvisualisierung. Diese ist zu finden im:
  - PLC-Projekt: TC3_SortingSystem_PLC
  - Ordner: 05_Visu
- Beispielsweise: Starten der Anlage im Automatikmodus durch Betätigen der folgenden Button:
  - “Main Switch Power Supply”
  - “Automatic”
  - “Start”

Siehe auch:

- Dokumentation PLC: Objektorientiert programmieren

8 SPS-Projekt auf die SPS übertragen

Um das SPS-Projekt auf die Steuerung zu übertragen, muss das Programm fehlerfrei übersetzt sein.

8.1 Programmcode erzeugen

Der Programmcode bezeichnet den Maschinencode, den eine Steuerung ausführt, wenn Sie ein SPS-Programm starten. TwinCAT erzeugt den Programmcode aus dem im Entwicklungssystem geschriebenen Quellcode automatisch vor dem Download des SPS-Projekts auf die Steuerung. Dabei wird vor dem Erzeugen des Programmcodes eine Prüfung der Zuweisungen, der Datentypen und der Verfügbarkeit von Bibliotheken durchgeführt. Weiterhin werden beim Erzeugen des Programmcodes die Speicheradressen vergeben.

Bei jedem Download wird zusätzlich das Übersetzungsprotokoll (Compile Info), welches Code und Identifikation des geladenen SPS-Projekts enthält, als Datei auf dem Zielgerät gespeichert.

Meldungen beim Erzeugen des Programmcodes

Da aufgrund der inkrementellen Kompilierung der Speicher nur für geänderte und neue Bausteine und Variablen neu vergeben wird, entstehen Lücken im Speicher. Denselben Effekt haben Online-Changes. Diese Fragmentierung verringert den verfügbaren freien Speicher. Sie können in diesem Fall den Speicher mithilfe des Befehls Bereinigen komplett neu vergeben und somit den freien Speicher wieder vergrößern. Weitere Informationen zu den Meldungen bei der Codegenerierung: Syntaxfehler und Fehler, die TwinCAT während der Codegenerierung und Speichervergabe feststellt, erscheinen im Meldungsfenster (Fehlerliste) Übersetzen. Bei jeder Codegenerierung werden dort außerdem Informationen zur Größe des Codes, zur Größe der Daten (in Bytes), zum Inhalt der belegten Speicherbereiche, und zur höchsten benutzten Adresse (Byte) ausgegeben. Es hängt von der SPS ab, in welchen Speicherbereichen welche Daten und der Code untergebracht werden.

8.2 Programmcode laden, einloggen und SPS starten

Um den Quellcode Ihres SPS-Programms auf die Steuerung zu laden, müssen Sie sich mit dem SPS-Projekt auf der Steuerung einloggen. Falls Sie mehrere SPS-Projekte in Ihrem Projekt haben, müssen Sie das gewünschte SPS-Projekt zuerst aktiv schalten.

Laden des SPS-Projekts und Starten des Programms

✓ Das SPS-Projekt ist fehlerfrei und befindet sich noch nicht auf der Steuerung.

✓ Das SPS-Projekt und die Kommunikation mit der Steuerung sind nicht verschlüsselt.

1. Wählen Sie in der Drop-down-Liste **Active PLC Project** der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** das SPS-Projekt, das geladen und gestartet werden soll.
   ⇨ Das aktive SPS-Projekt erscheint als erster Eintrag in der Drop-down-Liste.
2. Klicken Sie in den **TwinCAT XAE Base Symbolleistenoptionen** auf die Schaltfläche **Activate Configuration**  .
   ⇨ Ein Dialog erscheint mit der Abfrage, ob die Konfiguration aktiviert werden soll.
3. Klicken Sie auf **Ok**.
   ⇨ Ein Dialog erscheint mit der Abfrage, ob TwinCAT im Run-Modus neu gestartet werden soll.
4. Klicken Sie auf **Ok**.
   ⇨ Die Konfiguration wird aktiviert und TwinCAT in den Run-Modus gesetzt. In der Taskleiste erscheint der aktuelle Status:  . Durch die Aktivierung wird auch das SPS-Projekt auf die Steuerung übertragen.
5. Wählen Sie im Menü **PLC** oder in den **TwinCAT SPS Symbolleistenoptionen** den Befehl **Einloggen**  .
   ⇨ Ein Dialog erscheint mit der Abfrage, ob Sie die Applikation auf der Steuerung anlegen und laden wollen.

6. Bestätigen Sie den Dialog mit **YES**.
   ⇨ Das SPS-Projekt wird auf der Steuerung geladen.
7. Wählen Sie im Menü **PLC** oder in den **TwinCAT SPS Symbolleistenoptionen** den Befehl **Start** oder drücken Sie **[F5]**.

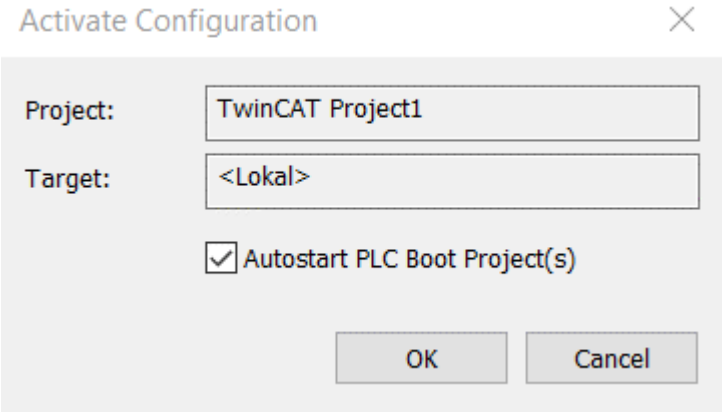
⇨ Das Programm läuft auf der Steuerung.

Siehe auch:

- SPS-Projekt auf der SPS aktualisieren [▶ 262]
- Dokumentation TC3 User Interface: Befehl Einloggen [▶ 1001]
- Dokumentation TC3 User Interface: Befehl Konfiguration aktivieren [▶ 981]
- Monitoring von Werten
- Programmiersprachen und ihre Editoren

8.3 Programm automatisch laden

Wenn Sie in Ihrem PLC-Projekt die Konfiguration aktivieren , öffnet sich ein Dialogfenster, in dem Sie sehen oder einstellen können, ob der Autostart PLC Boot Project(s) aktiviert ist.



Die Checkbox des Dialogs zeigt an, ob die für PLCs auf dem Target aktuell der Autostart aktiviert ist oder nicht:

- Checked: Alle Autostarts auf dem Target werden gesetzt.
- Unchecked: Alle Autostarts auf dem Target werden entfernt.

Um die Autostart-Funktion für Ihre PLCs zu verändern, setzen oder entfernen Sie den Haken in der Checkbox. Wenn Sie Ihre Auswahl mit **OK** bestätigen, wird Ihre Eingabe gespeichert und Sie gelangen in den Run Mode von TwinCAT.

Die Option entspricht dem Befehl **Autostart Boot Project** aus der Registerkarte **Projekt** des PLC-Projekts im Projektmappen-Explorer.

9 SPS-Projekt testen und Fehler beheben

TwinCAT 3 PLC bietet Ihnen verschiedene Möglichkeiten, Ihre Anwendung zu testen und Fehler zu finden. So können Sie im Simulationsbetrieb Ihre Anwendung auch ohne angeschlossene Hardware starten. Mit Haltepunkten und Befehlen zum schrittweisen Abarbeiten des Programms können Sie ganz gezielt bestimmte Programmstellen untersuchen. Durch das Schreiben von Variablenwerten können Sie das laufende Programm beeinflussen.

Es stehen Ihnen Befehle zur Verfügung, die Ihre Anwendung unterschiedlich stark zurücksetzen. Dies reicht vom Rücksetzen der nur nichtpersistenten Variablen bis zum kompletten Zurücksetzen der Steuerung in den Auslieferungszustand.

9.1 Haltepunkte verwenden

Haltepunkte (Breakpoints) werden üblicherweise zur Fehlersuche im Programm genutzt. Sie können Haltepunkte an bestimmten Positionen im Programm setzen, um dort einen Ausführungsstopp zu erzwingen und die Variablenwerte zu beobachten. TwinCAT 3 PLC unterstützt Haltepunkte in allen IEC-Editoren.

Der Stopp am Haltepunkt kann an zusätzliche Bedingungen geknüpft werden. Sie können Haltepunkte auch zu Ausführungspunkten umdefinieren, an denen das Programm nicht stoppt, sondern bestimmter Code abgearbeitet wird.

Die Ansicht **Haltepunkte** (Menü **PLC > Fenster**) gibt eine Übersicht über alle definierten Haltepunkte. Darin stehen Ihnen zusätzliche Befehle zum gleichzeitigen Ändern mehrerer Haltepunkte zur Verfügung.

Im Editor wird der Status von Haltepunkten und Ausführungspunkten mit folgenden Symbolen markiert:

- Haltepunkt aktiviert
- Haltepunkt deaktiviert
- Haltepunkt ist in einer anderen Instanz des gerade im Editor geöffneten Bausteins gesetzt.
- Stopp an Haltepunkt
- Haltepunkt mit Bedingung aktiviert
- Haltepunkt mit Bedingung deaktiviert
- Ausführungspunkt aktiviert
- Ausführungspunkt deaktiviert
- Ausführungspunkt mit Bedingung aktiviert
- Ausführungspunkt mit Bedingung deaktiviert

Siehe auch:

- Dokumentation TC3 User Interface: PLC [▶ 983]
- Dokumentation TC3 User Interface: Debuggen [▶ 973]
- Dokumentation TC3 User Interface: Befehl Aufrufliste [▶ 989]

Haltepunkte in SPS-Projekten mit mehreren Tasks

Wenn bei der Ausführung eines SPS-Projekts ein Haltepunkt erreicht wird, wird in diesem SPS-Projekt kein Code mehr von irgendeiner Task ausgeführt. Code, der sich außerhalb dieses SPS-Projekts befindet, wird weiterhin ausgeführt.

Wenn das Programm auf der SPS an einem Haltepunkt steht, erzeugt ein Online-Change oder Download ein Stoppen aller Tasks. Das bedeutet ein Stopp der SPS. In diesem Fall zeigt TwinCAT eine entsprechende Meldung an und Sie können entscheiden, ob Sie mit dem Login fortfahren wollen oder nicht.

Setzen eines einfachen Haltepunkts (Beispiel ST-Editor)

✓ Das Projekt ist im Onlinebetrieb.

1. Öffnen Sie eine POU in der Sprache ST im Editor.
2. Setzen Sie den Cursor in die Zeile, in der ein Haltepunkt gesetzt werden soll.
3. Wählen Sie im Menü **Debuggen** oder im Kontextmenü den Befehl **Haltepunkt umschalten** oder drücken Sie die Taste **[F9]**.

   ⇨ Die Zeile wird rot markiert und mit dem Icon (Haltepunkt aktiviert) gekennzeichnet. Wenn das Programm an dem Haltepunkt steht, wird die Zeile mit dem Icon (Stopp an Haltepunkt) gekennzeichnet. Die Abarbeitung des Programms stoppt.
4. Wählen Sie im Menü **PLC** oder in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** den Befehl **Start** oder drücken Sie die Taste **[F5]**.

   ⇨ Das Programm läuft weiter.
5. Setzen Sie weitere Haltepunkte und überprüfen Sie an den Haltepositionen die Werte von Variablen.
6. Setzen Sie den Cursor in eine Zeile, in der ein Haltepunkt gelöscht werden soll.
7. Wählen Sie im Menü **Debuggen** oder im Kontextmenü den Befehl **Haltepunkt umschalten** oder drücken Sie die Taste **[F9]**.

⇨ Die Markierung verschwindet. Der Haltepunkt ist gelöscht.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Haltepunkt umschalten [▶ 978]

Festlegen einer Haltepunkt-Bedingung (Beispiel ST-Editor)

✓ Das Projekt ist im Onlinebetrieb.

1. Öffnen Sie eine POU in der Sprache ST im Editor.
2. Wählen Sie im Menü **PLC > Fenster** den Befehl **Haltepunkte**.

   ⇨ Die Ansicht **Haltepunkte** öffnet sich.
3. Wählen Sie den Befehl **Neu** in der Symbolleiste.

   ⇨ Der Dialog **Neuer Haltepunkt** öffnet sich. Die Registerkarte **Ort** ist sichtbar. Alternativ können Sie den Dialog über den Befehl **Neuer Haltepunkt** im Menü **Debuggen** öffnen.
4. Wählen Sie die POU und die Position des neuen Haltepunkts.
5. Wählen Sie die Registerkarte **Bedingung**.
6. Wählen Sie im Abschnitt **Trefferzahl** die Option **Anhalten, wenn die Trefferzahl ein Vielfaches von** und geben Sie im Feld rechts daneben den Wert 5 ein.
7. Definieren Sie zusätzlich noch eine boolesche Bedingung, wann der Haltepunkt aktiv sein soll. Aktivieren Sie dazu die Option **Halt, wenn TRUE**. Geben Sie im Feld rechts daneben eine boolesche Variable ein.
8. Aktivieren Sie die Option **Haltepunkt sofort aktivieren**.
9. Schließen Sie den Dialog.

⇨ Die Zeile wird rot markiert und mit dem Icon gekennzeichnet.

Beobachten Sie nun das laufende Programm. Solange die boolesche Variable für die Bedingung FALSE ist, ist die Bedingung für den Haltepunkt nicht erfüllt und das Programm läuft. Wenn Sie die Variable auf TRUE setzen, ist die Bedingung erfüllt und das Programm bleibt bei jedem 5. Durchlauf an diesem Haltepunkte stehen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Haltepunkte [▶ 987]

- Dokumentation TC3 User Interface: Befehl Neuer Haltepunkt [▶ 973]

Festlegen eines Ausführungspunkts (Beispiel ST-Editor)

✓ Das Projekt ist im Onlinebetrieb.

1. Öffnen Sie eine POU in der Sprache ST im Editor.
2. Wählen Sie im Menü **PLC > Fenster** den Befehl **Haltepunkte**.
   ⇨ Die Ansicht **Haltepunkte** öffnet sich.
3. Wählen Sie den Befehl **Neu** in der Symbolleiste.
   ⇨ Der Dialog **Neuer Haltepunkt** öffnet sich. Die Registerkarte **Ort** ist sichtbar.
   Alternativ können Sie den Dialog über den Befehl **Neuer Haltepunkt** im Menü **Debuggen** öffnen.
4. Wählen Sie die POU und die Position des Ausführungspunkts.
5. Wählen Sie die Registerkarte **Ausführungspunkt Einstellungen**.
6. Aktivieren Sie die Option **Ausführungspunkt**.
   Geben Sie im Feld **Folgenden Code ausführen** die gewünschten Anweisungen ein, die beim Erreichen des Ausführungspunktes ausgeführt werden sollen. Zum Beispiel `nCounter := nCounter + 1;`, falls die Variable nCounter verfügbar ist.
7. Schließen Sie den Dialog.

⇨ Die Zeile wird rot markiert und mit dem Icon ● gekennzeichnet

Wenn das Programm den Ausführungspunkt erreicht, bleibt es nicht stehen, sondern es wird der oben definierte Code ausgeführt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Haltepunkte [▶ 987]
- Dokumentation TC3 User Interface: Befehl Neuer Haltepunkt [▶ 973]

9.2 Schrittweises Abarbeiten eines Programms (Stepping)

Sie können ein SPS-Projekt Schritt für Schritt ausführen und dabei durch den Code navigieren. Das ist hilfreich, um den Status Ihres Codes zur Laufzeit zu ermitteln. Sie können dabei den Aufrufablauf untersuchen, Variablenwerte verfolgen oder Fehler ermitteln. Dafür stehen Ihnen im Menü **Debug** Schrittbefehle zur Verfügung. Die Befehle werden verfügbar, wenn das Programm an einem definierten Programmschritt steht (Debugbetrieb). Während des Debugbetriebs wird die aktuelle Halteposition gelb hinterlegt und in den Texteditoren mit dem Symbol gekennzeichnet.

In den Debugbetrieb wechseln

✓ Das SPS-Projekt ist im Onlinebetrieb.

1. Setzen Sie in den POUs Haltepunkte an den Stellen im Code, die Sie untersuchen wollen.
2. Starten Sie das Programm.
   ⇨ Das Programm startet, der Code wird bis zum ersten Haltepunkt abgearbeitet. Das Projekt ist nun im Debugbetrieb.
   ⇨ Der Editor mit der aktuellen Halteposition ist geöffnet. Die Codezeile mit aktivem Haltepunkt, an der die Programmausführung angehalten wurde, ist gelb hinterlegt und mit dem Symbol (Stopp an Haltepunkt) gekennzeichnet. Die Anweisung wurde noch nicht ausgeführt.
   ⇨ Sie können die verschiedenen Schrittbefehle wählen oder sich die Aufrufliste anzeigen lassen.

Verhalten der Schrittbefehle

- Befehl Prozedurschritt [▶ 978] („Step Over“)
  Die Anweisung an der Halteposition wird ausgeführt. Vor der nächsten Anweisung im Programmierbaustein wird angehalten.

Wenn in der Anweisung ein Aufruf ist (von einem Programm, einer Funktionsbaustein-Instanz, einer Funktion, einer Methode oder einer Aktion), wird der untergeordnete Programmierbaustein in einem Schritt vollständig durchlaufen.

- Befehl Einzelschritt [▶ 979] („Step Into“) 

  Die Anweisung an der Halteposition wird ausgeführt. Vor der nächsten Anweisung wird angehalten.

  Wenn in der Anweisung ein Aufruf ist (von einem Programm, einer Funktionsbaustein-Instanz, einer Funktion, einer Methode oder einer Aktion), wird in diesen untergeordneten Programmierbaustein gesprungen. Die erste Anweisung dort wird ausgeführt und vor der nächsten Anweisung wird angehalten. Die neue aktuelle Halteposition ist dann im aufgerufenen Programmierbaustein.

- Befehl Ausführen bis Rücksprung [▶ 979] („Step Out“) 

  Der Befehl führt den Programmierbaustein von der aktuellen Halteposition bis zum Bausteinende aus und springt dann zurück in den aufrufenden Programmierbaustein. An der Aufrufstelle (in der Zeile mit dem Aufruf) wird angehalten.

  Wenn die aktuelle Halteposition im Hauptprogramm ist, wird der Programmierbaustein bis zum Ende durchlaufen. Dann wird zurück an den Anfang (an den Programmstart an die erste Codezeile im Programmierbaustein) gesprungen und dort angehalten.

- Befehl Ausführen bis Cursor [▶ 980] („Run to Cursor“) 

  Setzen Sie zunächst den Cursor an eine beliebige Codezeile und wählen Sie dann den Befehl. Das Programm wird ab der aktuellen Halteposition ausgeführt und hält an der aktuellen Cursorposition an, ohne den Code dieser Zeile auszuführen.

- Befehl Nächste Anweisung festlegen [▶ 980] („Set Next Statement“) 

  Setzen Sie zunächst den Cursor an eine beliebige Codezeile (auch vor der aktuellen Halteposition) und wählen Sie dann den Befehl. Die mit dem Cursor gekennzeichnete Anweisung wird als nächstes ausgeführt. Alle Anweisungen dazwischen werden ignoriert und übersprungen.

- Befehl Nächste Anweisung anzeigen [▶ 980] („Show Next Statement“) 

  Wenn Sie die aktuelle Halteposition nicht sehen, wählen Sie den Befehl. Dann wird das Fenster mit der aktuellen Halteposition aktiv und die Halteposition sichtbar.

Wählen Sie den Befehl PLC > Fenster > Aufrufliste, um für die aktuell in der Programmabarbeitung erreichte Halteposition der bisherige Aufrufbaum vollständig anzuzeigen.

Die Ansicht **Aufrufbaum** zeigt jederzeit, auch schon vor dem Kompilieren (Übersetzen) des SPS-Projekts, wo sich der Baustein in der Aufrufstruktur des Programms befindet.

Sehen Sie dazu auch

- Befehl Einzelschritt [▶ 979]
- Befehl Prozedurschritt [▶ 978]
- Befehl Ausführen bis Rücksprung [▶ 979]
- Befehl Ausführen bis Cursor [▶ 980]
- Befehl Nächste Anweisung anzeigen [▶ 980]

9.3 Forcen und Schreiben von Variablen

Sie können in TwinCAT den Wert von Variablen auf der Steuerung im Onlinebetrieb verändern. Dabei wird der ursprünglich eingetragene Wert der Variablen überschrieben. Hierfür gibt es zwei unterschiedliche Vorgehensweisen, das Forcen und das Schreiben eines vorher vorbereiteten Werts.

⚠ VORSICHT

Sach- und Personenschäden durch unerwartetes Verhalten der Maschine oder Anlage

Das außerordentliche Ändern von Variablenwerten in einem auf der Steuerung laufenden SPS-Programm kann zu einem unerwarteten Verhalten der gesteuerten Maschine führen. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

- Evaluieren Sie vor einem Forcen von Variablenwerten mögliche Gefahren und treffen Sie entsprechende Sicherheitsvorkehrungen.

Das Schreiben erfolgt mit dem Befehl **Werte schreiben**  und setzt die Variable einmalig auf den vorbereiteten Wert. Der Wert kann somit jederzeit durch das Programm überschrieben werden.

Das Forcen erfolgt mit dem Befehl **Werte forcen**  und setzt den vorbereiteten Wert dauerhaft.

Das Vorbereiten eines Wertes für das Forcen oder Schreiben ist an verschiedenen Stellen möglich:

- Deklarationsteil: Feld **Vorbereiteter Wert**
- Implementierungsteil: Inline-Monitoring-Feld
- Monitoring-Fenster: Feld **Vorbereiteter Wert**

Funktionsweise des Forcens

Beim Forcen schreibt TwinCAT den Wert in jedem Zyklus, sodass die Variable dauerhaft auf dem geforcten Wert gehalten wird. Das Forcen muss durch den Nutzer aufgehoben werden. Dabei ist der geforcte Wert der Variablen innerhalb eines PLC-Zyklus veränderbar, wie bei jeder anderen Variablen.

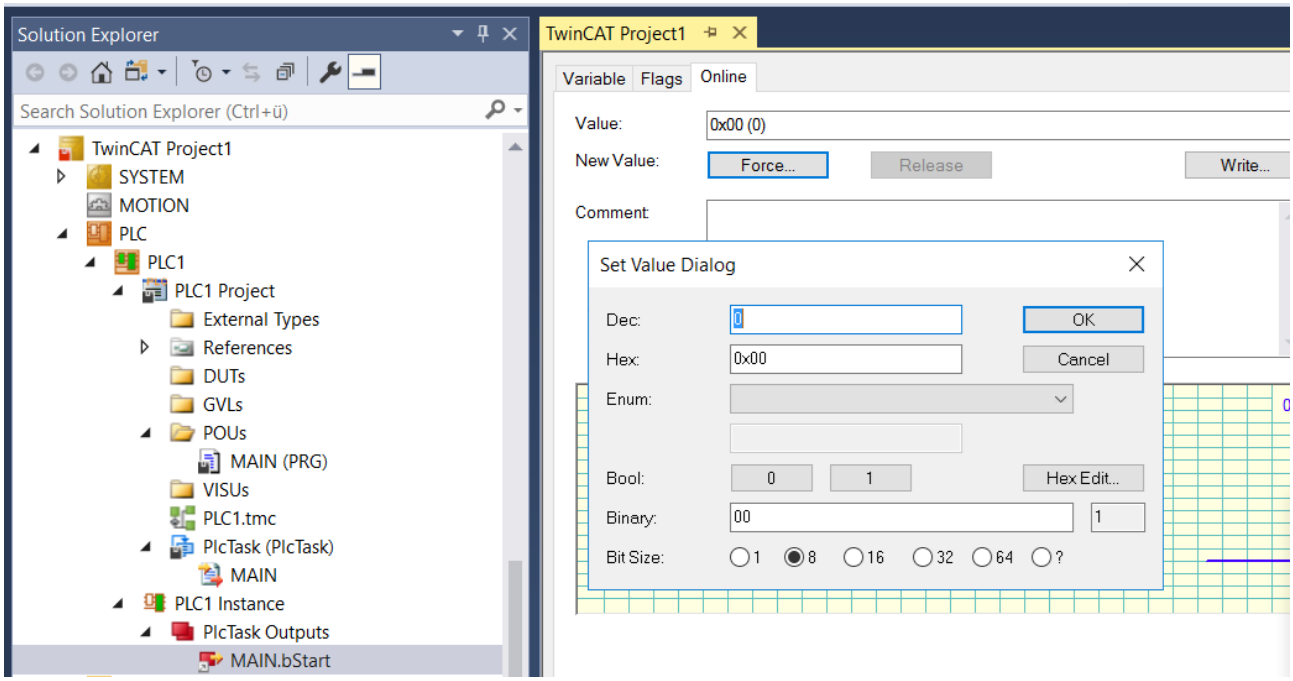
Das Setzen des vorbereiteten Werts auf die betreffende Variable erfolgt jeweils am Beginn und Ende eines Abarbeitungszyklus. Abfolge der Abarbeitung in jedem Zyklus:

1. Eingänge lesen
2. Werte forcen
3. Code abarbeiten
4. Werte forcen
5. Ausgänge schreiben.

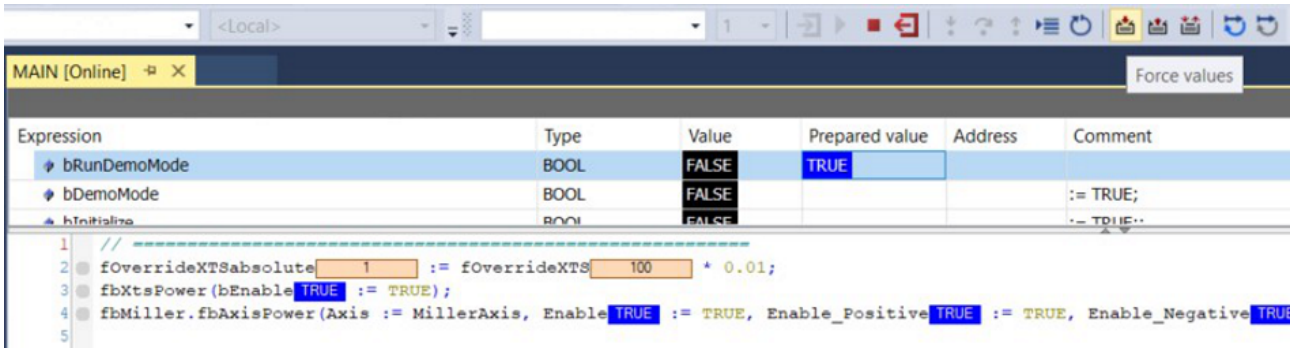
Es ist möglich, dass eine geforcte Variable während der Codeabarbeitung im Zyklus vorübergehend einen anderen Wert bekommt, weil der Code eine Zuweisung durchführt. Die Variable erhält dann erst wieder am Ende des Zyklus den geforcten Wert. Auch durch den Schreibzugriff eines Clients auf Symbole der Applikation kann der Variablenwert mitten im Zyklus überschrieben werden.

Es gibt zwei unterschiedliche Vorgehensweisen, um Werte zu forcen.

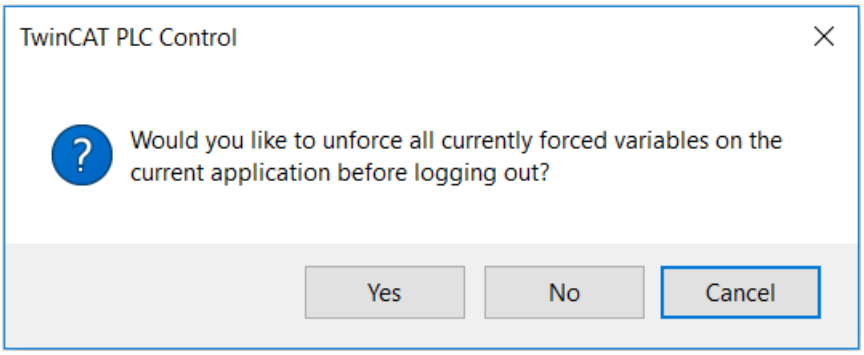
Für die eine Vorgehensweise kann das Projekt vom Projektmappen-Explorer aus geöffnet werden. Von hier aus werden die Werte dann direkt geforct. Ein Einloggen in die Laufzeit ist hierfür nicht erforderlich. Dadurch bleibt der Wert der Variablen auch dann geforct, wenn der Nutzer das Laufzeitsystem verlässt. Es erscheint in diesem Fall weder eine Warnmeldung noch ein Abfragedialog.



Im Gegensatz dazu ist das Forcen einer Variablen in der PLC nur möglich, wenn sich der Nutzer in das Laufzeitsystem einloggt .



Beim Ausloggen aus dem Laufzeitsystem erscheint ein Dialog, der abfragt, ob die Variable weiterhin geforct bleiben soll.



Wählen Sie in diesem Dialog **Ja** aus, werden die geforcten Werte aufgehoben. Wenn Sie hier **Nein** auswählen, werden die geforcten Werte auf dem Laufzeitsystem gespeichert und bleiben entsprechend dauerhaft bestehen. Das heißt, Sie können sich in der Zwischenzeit aus dem Laufzeitsystem ausloggen und die geforcten Werte bestehen noch, wenn Sie sich zu einem anderen Zeitpunkt wieder in das Laufzeitsystem einloggen.

HINWEIS

Sachschäden durch dauerhaft geforcte Variablen

Variable werden dauerhaft auf dem geforcten Wert gehalten, dadurch kann der geforcte Wert länger als erwartet bestehen bleiben und es kann zu Sachschäden kommen. Das gilt insbesondere, wenn die Maschine unbeaufsichtigt läuft.

- Um sicherzustellen, dass die Maschine keine unerwarteten Bewegungen ausführt, geforcte Werte am Ende des Bearbeitungsvorgangs zurücksetzen.

Beachten Sie, dass geforcte Variablen explizit durch den Anwender aufgehoben werden müssen. Der geforcte Wert der Variablen kann allerdings auch nach dem Unforcen bestehen bleiben.

- Heben Sie den Force der Variablen auf.
- Um den geforcten Wert der Variablen sicher aufzuheben, ändern Sie die Werte zurück auf die Ursprungswerte.
- Loggen Sie sich aus TwinCAT aus und bestätigen Sie die Abfrage, ob das Forcen für alle Variablen aufgehoben werden soll, mit **Ja**.

⇨ Die Variable hat jetzt wieder ihren ursprünglichen Wert.

Temporäre Variablen können nicht geforct werden. Geschrieben werden können temporäre Variablen nur dann, wenn die SPS in einem Haltepunkt steht, der sich im Code derselben POU befindet, in dem auch die temporäre Variable definiert ist. Auch im Flow-Control-Modus hat das Schreiben von temporären Variablen keinen Effekt.

Siehe auch:

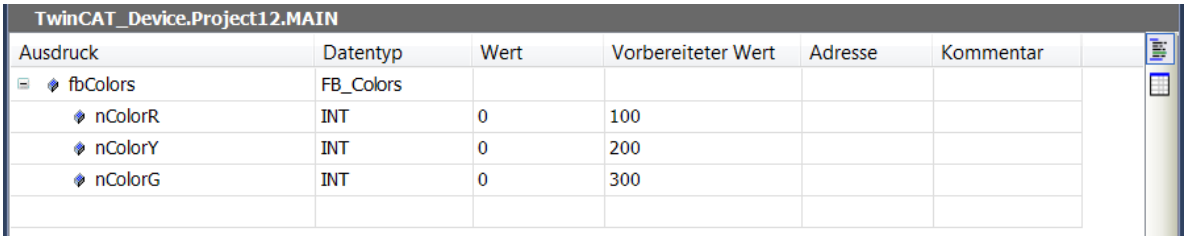
- Dokumentation TC3 User Interface: Befehl Werte forcen [▶ 1005]
- Dokumentation TC3 User Interface: Befehl Werte schreiben [▶ 1007]
- Dokumentation TC3 User Interface: Befehl Forcen aufheben [▶ 1006]

Forcen im Deklarationsteil

✓ Ihr SPS-Projekt besitzt eine POU mit Deklarationen. Die Anwendung befindet sich im Onlinebetrieb.

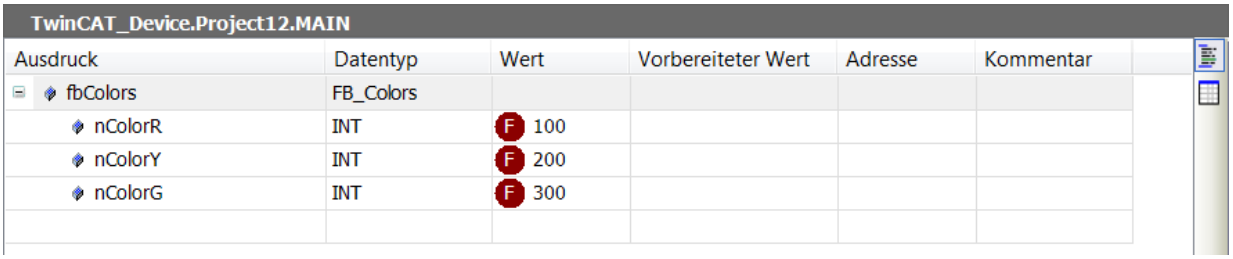
1. Öffnen Sie die POU im Editor durch einen Doppelklick auf das Objekt oder den Befehl **Öffnen** im Menü **Ansicht** oder im Kontextmenü.
2. Führen Sie im Deklarationsteil des Editors einen Doppelklick in der Spalte **Vorbereiteter Wert** einer Variablen aus.

   ⇨ Das Feld wird editierbar und Sie können einen Wert eingeben.



3. Führen Sie Schritt 2 für weitere Variablen aus.
4. Wählen Sie im Menü **PLC** oder in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** den Befehl **Werte forcen** .

⇨ Die Werte der Variablen werden mit den vorbereiteten Werten überschrieben. Die Werte sind mit dem Symbol F gekennzeichnet.



Sie können Werte von Variablen auch in der Ansicht **PLC > Fenster > Überwachungsliste <n>** forcen.

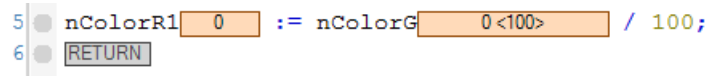
Siehe auch:

- Dokumentation TC3 User Interface: Befehl Werte forcen [▶ 1005]

Forcen im Implementierungsteil

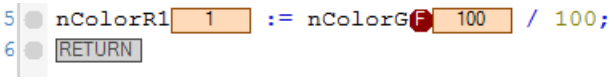
✓ Die Anwendung befindet sich im Onlinebetrieb.

1. Öffnen Sie die POU im Editor durch einen Doppelklick auf das Objekt oder den Befehl **Öffnen** im Menü **Ansicht** oder im Kontextmenü.
2. Führen Sie im Implementierungsteil des Editors einen Doppelklick auf ein Inline-Monitoring-Feld aus.
   ⇨ Der Dialog **Werte vorbereiten** öffnet sich.
3. Geben Sie im Feld **Einen neuen Wert für die nächste Schreib- oder Force-Operation vorbereiten** den neuen Wert ein.
   ⇨ Der vorbereitete Wert erscheint im Inline-Monitoring-Feld.



4. Wählen Sie den Befehl **Werte forcen** im Menü **PLC** oder in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen**.

⇨ Der Wert der Variablen wird mit den vorbereiteten Werten überschrieben. Die Werte sind mit dem Symbol F gekennzeichnet.



Siehe auch:

- Dokumentation TC3 User Interface: Dialog Wert vorbereiten [▶ 1006]
- Dokumentation TC3 User Interface: Befehl Werte forcen [▶ 1005]
- Dokumentation TC3 User Interface: Befehl Forcen aufheben [▶ 1006]

Alle geforcten Variablen in einer Liste sehen und aufheben

✓ Die Anwendung befindet sich im Onlinebetrieb. Mehrere Variablen sind im geforcten Zustand.

1. Wählen Sie im Menü **PLC > Fenster** den Befehl **Alle Forces anzeigen**.
   ⇨ Die Ansicht **Alle Forces anzeigen** erscheint. Sie enthält in Form einer Überwachungsliste alle aktuell geforcten Variablen des SPS-Projekts.
2. Selektieren Sie alle Zeilen der Liste und wählen Sie in der Auswahlliste links oben in der Ansicht **Force aufheben > Forcen aufheben und alle ausgewählten Werte beibehalten**.

⇨ Das Forcen für die Variablen wird aufgehoben, sie erhalten die Werte, die sie vor dem Forcen hatten.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Alle Forces anzeigen [▶ 984]
- Dokumentation TC3 User Interface: Befehl Werte schreiben [▶ 1007]
- Dokumentation TC3 User Interface: Befehl Forcen aufheben [▶ 1006]

Sehen Sie dazu auch

- Befehl Werte schreiben [▶ 1007]
- Befehl Werte forcen [▶ 1005]
- Befehl Forcen aufheben [▶ 1006]

9.4 Reset des SPS-Projekts durchführen

Ein Reset des SPS-Projekts stoppt das Programm und setzt die Variablen auf ihren Initialisierungswert zurück. Abhängig von der Art des Reset werden auch RETAIN-Variablen und PERSISTENT-Variablen zurückgesetzt.

- Reset kalt: Alle Variablen des aktiven SPS-Projekts, mit Ausnahme der remanenten Variablen (RETAIN- und PERSISTENT-Variablen), werden auf ihren Initialisierungswert zurückgesetzt.
- Reset Ursprung: Alle Variablen des aktiven SPS-Projekts, einschließlich der remanenten Variablen (RETAIN- und PERSISTENT-Variablen), werden auf ihren Initialisierungswert zurückgesetzt. Das SPS-Projekt auf der Steuerung wird zurückgesetzt.

Das kleine Beispielprogramm und die nachfolgende Handlungsanweisungen verdeutlichen Ihnen das Verhalten der verschiedenen Resets.

Siehe auch:

- Referenz Programmierung: Remanente Variablen - PERSISTENT, RETAIN [▶ 726]
- Dokumentation TC3 User Interface: Befehl Reset kalt [▶ 1003]
- Dokumentation TC3 User Interface: Befehl Reset Ursprung [▶ 1004]

Beispielprogramm

Deklaration:

```
VAR
    nVar : INT := 0;
END_VAR
VAR RETAIN
    nVarRetain : INT :=0;
END_VAR
VAR PERSISTENT
    nVarPersistent : INT:= 0;
END_VAR
```

Implementierung:

```
nVar := 100;
nVarRetain := 200;
nVarPersistent := 300;
```

1. Führen Sie den Befehl **Erstellen** aus.
2. Laden Sie das SPS-Projekt auf die Steuerung.
3. Wählen Sie im Menü **PLC** oder in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** den Befehl **Einloggen**, um in den Onlinebetrieb zu wechseln.
4. Starten Sie das SPS-Programm.

⇨ Beobachten Sie die Variablen nVar, nVarRetain und nVarPersistent.

Ausführen eines Reset kalt:

1. Wählen Sie den Befehl **Reset kalt** im Menü **PLC** oder in den **TwinCAT SPS Symbolleistenoptionen**.
   ⇨ Eine Abfrage erscheint, ob Sie den Befehl wirklich durchführen wollen.
2. Bestätigen Sie den Dialog mit **Ja**.

⇨ Das SPS-Projekt wird zurückgesetzt. Die Variable nVar wird auf den Initialisierungswert 0 gesetzt. Die RETAIN-Variable nVarRetain und die PERSISTENT-Variable nVarPersistent behalten ihren Wert

Ausführen eines Reset Ursprung:

1. Wählen Sie im Menü **PLC** oder in den **TwinCAT SPS Symbolleistenoptionen** den Befehl **Reset Ursprung**.
   ⇨ Eine Abfrage erscheint, ob Sie den Befehl wirklich durchführen wollen.
2. Bestätigen Sie den Dialog mit **Ja**.

⇨ Das SPS-Projekt wird ausgeloggt. Alle Variablen werden auf ihren Initialisierungswert zurückgesetzt.

9.5 Ablaufkontrolle

Mit der Ablaufkontrolle können Sie die Abarbeitung des Programms verfolgen. Die Ablaufkontrolle ist für die Spracheditoren ST, FUP, KOP und CFC verfügbar.

Mit aktivierter Ablaufkontrolle stellt TwinCAT die Werte von Variablen die Ergebnisse von Funktionsaufrufen und Operationen an der jeweiligen Abarbeitungsposition und zum jeweiligen Abarbeitungszeitpunkt dar. Dabei werden exakt diejenigen Codezeilen bzw. Netzwerke farblich markiert, die im aktuellen Zyklus durchlaufen werden. Zum Vergleich: Beim Standard-Monitioring liefert TwinCAT nur den Wert, den eine Variable zwischen zwei Abarbeitungszyklen hat.

Die Ablaufkontrolle arbeitet in allen gerade sichtbaren Teilen der gerade geöffneten Editorfenster. Dabei wird **Ablaufkontrolle aktiviert** in der Statuszeile angezeigt, solange die Funktion aktiv ist und in einem Editorfenster Ablaufkontrollpositionen (durchlaufene Teile des Codes) sichtbar sind.

Sie können Werte im Deklarationsteil und im Implementierungsteil schreiben. Ein Forcen ist nicht möglich.

Das Schreiben der Werte erfolgt am Ende des aktuellen Zyklus.

Wenn Sie die Ablaufkontrolle aktivieren, verlängert sich die Laufzeit des SPS-Projekts.

Darstellung der Ablaufkontrolle in verschiedenen Spracheditoren

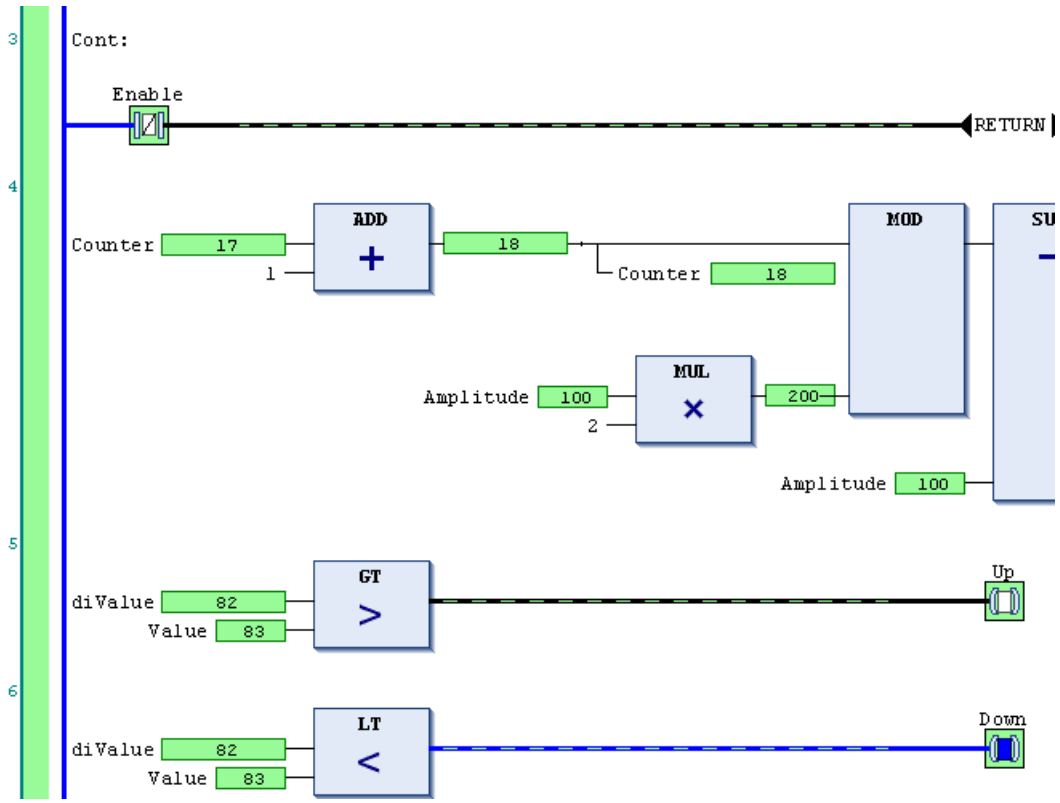
Standardmäßig stellt TwinCAT die Ablaufkontrollposition der durchlaufenen Codeteile als grüne Felder dar. Nichtdurchlaufene Codeteile werden weiß dargestellt.

Beachten Sie, dass der angezeigte Wert einer nicht durchlaufenen Codeposition ein „normaler“ Monitoring-Wert ist. Dies ist der Wert, der zwischen zwei Task-Zyklen vorliegt.

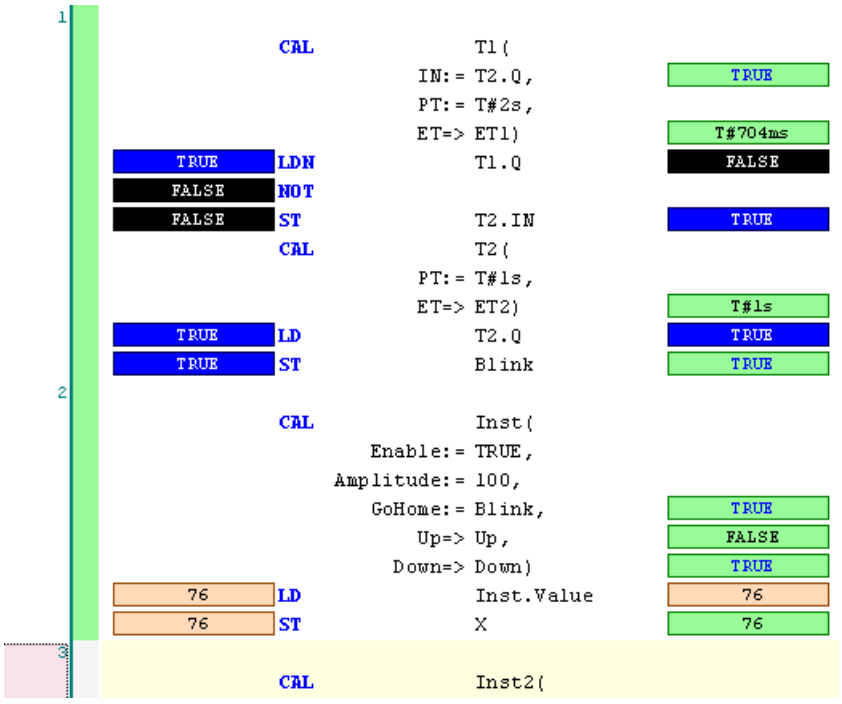
```
1  i 1619  := i 1619  + 1;
2  b 0  := NOT b 0 ;
3  IF str 'abcdefghij  = str1 "  THEN
4  fl2 1.5  := fl 1.23 ;
5  ELSE
6  fl2 1.5 :=1.5;
7  D 6.5E+04 :=B255*B255;
8  END_IF;
9  IF D 6.5E+04 <0.0 THEN
```

In Netzwerkeditoren markiert TwinCAT die durchlaufenen Netzwerke am linken Rand durch Balken in der „Ablaufkontrollfarbe“.

Im KOP stellt TwinCAT die aktuell durchlaufenen Verbindungslinien in grün dar, die anderen in grau. Der Istwert der Verbindung wird ebenfalls dargestellt: TRUE durch fette blaue, FALSE durch fette schwarze Linien, unbekannte oder analoge Werte durch dünne schwarze Linien. Das kann durch Kombination der jeweiligen Informationen zu gestrichelten Linien führen.



In AWL verwendet TwinCAT für jede Anweisung zwei Felder für die Anzeige der Istwerte. Eins links des Operators mit dem aktuellen Akkumulator-Wert, eins rechts des Operanden mit dem Operanden-Wert.



Siehe auch:

- Dokumentation TC3 User Interface: Befehl Ablaufkontrolle [▶ 1004]

9.6 Aktuelle Abarbeitungsposition mit der Aufrufliste bestimmen

Mithilfe der Aufrufliste können Sie die aktuelle Position der Programmabarbeitung bestimmen. Diese Funktion ist bei der schrittweisen Abarbeitung des Programms sehr nützlich.

✓ Das Projekt ist im Onlinebetrieb. Das Programm steht an einem Haltepunkt oder Sie führen es schrittweise aus.

1. Öffnen Sie die Aufrufliste mit dem Befehl **Aufrufliste** im Menü **PLC > Fenster**.

⇨ Die Aufrufliste wird geöffnet. Die Liste zeigt die aktuell erreichte Position mit vollständigem Aufrufpfad.

Die Aufrufliste ist auch im Offlinebetrieb verfügbar oder im normalen Onlinebetrieb (ohne gerade die Debugging-Funktionen zu benützen). In diesem Fall enthält sie die zuletzt während einer schrittweisen Ausführung angezeigte Position in „gegrauter“ Schrift.

Siehe auch:

- Schrittweises Abarbeiten eines Programms (Stepping) [▶ 224]
- Dokumentation TC3 User Interface: Befehl Aufrufliste [▶ 989]

10 SPS-Projekt zur Laufzeit

Wenn das Anwendungsprogramm auf der SPS läuft, gibt es im TwinCAT-3-Entwicklungssystem Funktionalitäten zum Überwachen und Ändern der Variablenwerte, sowie zum Aufzeichnen und Speichern ihres Verlaufs.

10.1 Monitoring von Werten

Sie können zur Laufzeit die aktuellen Werte von Variablen eines Programmierobjekts an verschiedenen Stellen im Projekt beobachten. Dies wird „Monitoring“ genannt:

- Online-Ansicht des Programmiereditor eines Objekts: „Inline-Monitoring“
- Online-Ansicht des Deklarationseditors eines Objekts
- Objektunabhängige, konfigurierbare Überwachungslisten

Sie können die Ergebnisse von Funktionsaufrufen und die aktuellen Werte von Variablen in Objekten des Typs **Eigenschaft** (Property) monitoren, wenn Sie das Pragma {attribute 'monitoring'...} einsetzen.

Siehe auch:

- Referenz Programmierung: Pragmas > Attribut 'monitoring' [▶ 846]

10.1.1 Monitoring in Programmierobjekten aufrufen

✓ Sie haben ein Projekt auf die Steuerung geladen und gestartet.
Sie möchten die aktuellen Werte, die die Variablen auf der Steuerung haben, im TwinCAT-Entwicklungssystem in den Editoren der Programmierbausteine beobachten.

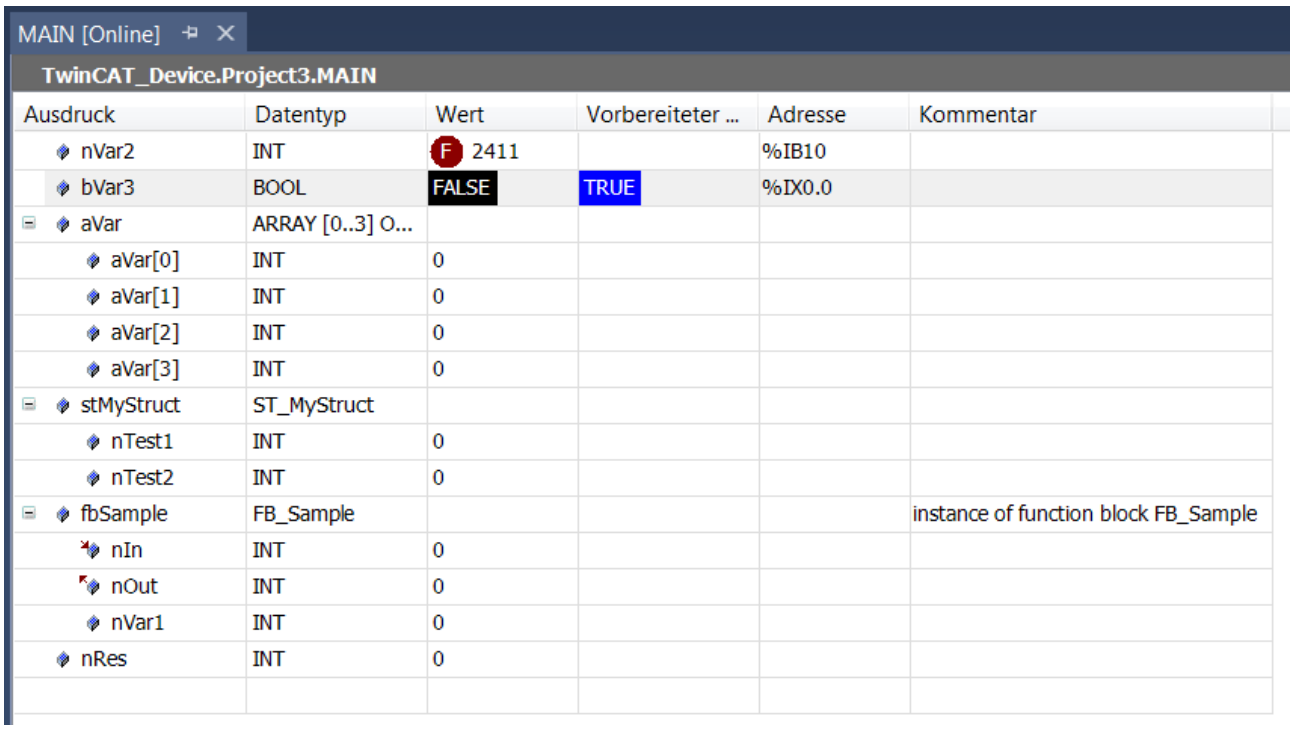
1. Stellen Sie sicher, dass im Menü **Extras > Optionen** in der Kategorie **TwinCAT > SPS-Programmierumgebung > Texteditor** in der Registerkarte **Monitoring** die Option **Inline-Monitoring aktivieren** aktiviert ist.
2. Wählen Sie im Menü **PLC > Darstellung** den Befehl **Dezimal**, um das Darstellungsformat der Werte einzustellen.
3. Öffnen Sie mit Doppelklick auf die POU im SPS-Projektbaum im Projektmappen-Explorer den zugehörigen Editor.

⇨ Sie sehen die Darstellungen der Variablenwerte im Deklarationsteil und Implementierungsteil. Und sie sehen die allgemeine Beschreibung dazu, sowie die Besonderheiten abhängig vom Typ der POU und des Programmiereditors.

Im Deklarationseditor monitoren

Der aktuelle Wert einer Variablen wird in der Spalte **Wert** dargestellt. Sie können den in Spalte

Vorbereiteter Wert eingetragenen Wert schreiben und forcen. Geforcten Werten ist ein rotes Symbol  vorangestellt.



Der Ausdruck einer Schnittstellenreferenz ist aufklappbar. Wenn die Schnittstelle auf eine globale Instanz zeigt, wird diese globale Instanz als erster Eintrag unter der Referenz angezeigt. Wenn sich danach die Schnittstellenreferenz ändert, klappt die angezeigte Referenz zu.

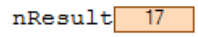
Siehe auch:

- Deklarationseditor [▶ 654]
- Forcen und Schreiben von Variablen [▶ 225]

In der Implementierung monitoren (Inline-Monitoring)

Die Darstellung des aktuellen Variablenwertes im Implementierungsteil wird als Inline-Monitoring bezeichnet. Abhängig von der Programmiersprache gibt es folgende Darstellungen:

- Variablen haben hinter ihrem Namen ein Fenster mit dem aktuellen Wert eingeblendet:

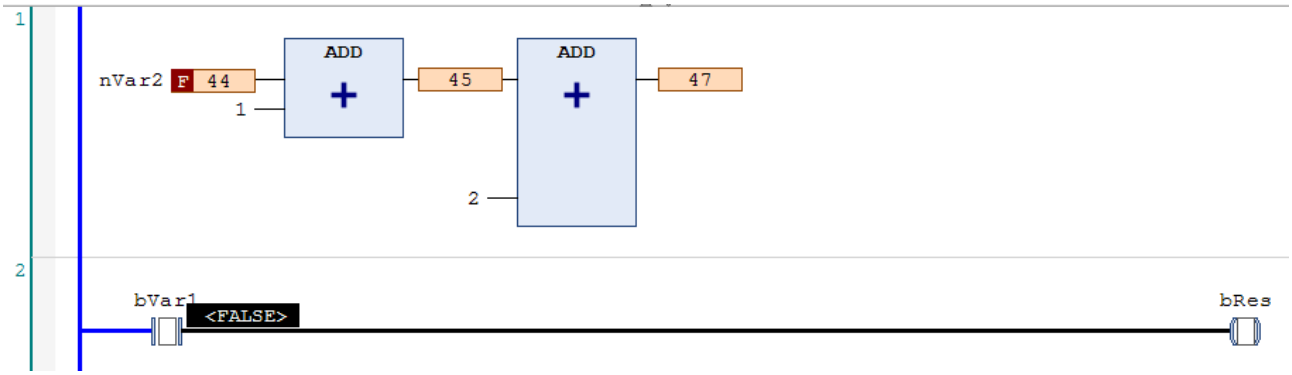
  Wenn Sie für Variablen Werte zum Forcen oder Schreiben vorbereitet haben, werden diese innerhalb des Inline-Monitoring-Fensters hinter dem aktuellen Wert in spitzen Klammern dargestellt.

  Nach dem Forcen werden die betroffenen Werte mit dem Symbol F gekennzeichnet.
- Netzwerkeditoren und CFC:
  Verbindungslinien (Signalleitungen) sind entsprechend ihrem booleschen Istwert farblich gekennzeichnet: blau bedeutet TRUE, schwarz bedeutet FALSE.
- KOP-Editor:
  Zusätzlich sind die Kontakt- und Spulenelemente gekennzeichnet. Bei Kontakten oder Spulen wird ein vorbereiteter Wert (TRUE oder FALSE) in einem kleinen Fenster neben dem Element angezeigt.
- AS-Editor:
  Transitionen mit Wert TRUE sind entsprechend ihrem booleschen Istwert farblich gekennzeichnet: blau bedeutet TRUE, schwarz bedeutet FALSE.
  Aktive Schritte sind blau gekennzeichnet.
  In der Implementierung sind geforcte Transitionswerte rot gekennzeichnet.
- AWL-Tabelleneditor:
  Aktuelle Werte werden in einer eigenen Spalte dargestellt

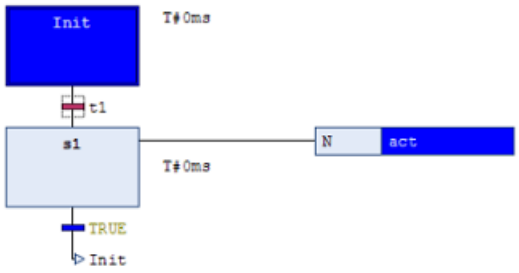
Monitoring im ST-Editor:

```
1  nVar2 44 := nVar2 44 + 1; (*counter*)
2  bVar3 TRUE := TRUE;
3  stMyStruct.nTest1 45 := nVar2 44;
4  aVar[2] 45 := nVar2 44;
5  nRes 0 := fbSample.nOut 0;
6
```

Monitoring im KOP-Editor:



Monitoring im AS-Editor:



Sie können die Inline-Monitoring-Funktion hier deaktivieren: Menü **Extras > Optionen**, Kategorie **TwinCAT > SPS Programmierumgebung > Texteditor**, Registerkarte **Monitoring**.

Siehe auch:

- Referenz Programmierung: ST-Editor im Onlinebetrieb [▶ 657]
- Referenz Programmierung: FUP/KOP/AWL-Editor im Onlinebetrieb [▶ 689]
- Referenz Programmierung: AS-Editor im Onlinebetrieb [▶ 668]
- Referenz Programmierung: CFC-Editor im Onlinebetrieb [▶ 707]

Array teilweise monitoren

Ein Array, das aufgeklappt ist, zeigt bei bis zu 1000 Elementen die Istwerte an. Das kann unübersichtlich sein. Außerdem kann ein Array mehr als 1000 Elemente umfassen. Dann ist es hilfreich, den Bereich der angezeigten Elemente zu begrenzen. Das können Sie während des Onlinebetriebs auf folgende Weise tun.

✓ Sie haben ein Projekt auf die Steuerung geladen, in dem eine mehrdimensionale Arrayvariable mit mehr als 1000 Elementen deklariert ist.
Beispiel: `aSample : ARRAY [1..100, 1..10, 1..20] OF INT;`

1. Klicken Sie bei der Variablen `aSample` in das Feld der Spalte **Datentyp**.
   ⇨ Der Dialog **Monitoringbereich** öffnet sich.
2. Geben Sie bei **Start** den Wert [1, 0, 0] ein.
3. Geben Sie bei **Ende** den Wert [1, 10, 20] ein.

⇨ Die Istwerte von 200 Arrayelementen werden angezeigt. Der Bereich ist begrenzt auf Elemente des Index [1, <i>, <j>].

Funktionsbaustein monitoren

Wenn Sie dabei sind, im Onlinebetrieb die Editoransicht für einen Funktionsbaustein zu öffnen, werden Sie gefragt, ob Sie die Ansicht der Basis-Implementation oder die einer Instanz des Bausteins öffnen möchten. In der Basis-Implementierung ist Monitoring nur möglich, wenn ein Haltepunkt gesetzt ist. Wenn Sie den Haltepunkt in der Basis-Implementierung setzen, wird er in allen Instanzen gesetzt. In der Ansicht der Basis-Implementierung werden dann zunächst die Werte derjenigen Instanz dargestellt, die im Programmablauf zuerst abgearbeitet wird.

Wenn Sie während des Onlinebetriebs auf die Editoransicht eines Funktionsbausteins doppelklicken, erscheint ein Dialog, in dem Sie zwischen der Ansicht der Basisimplementierung oder einer bestimmten Instanz wählen können.

Wenn Sie die Basisimplementierung wählen, dann erscheint im Editor der Code, ohne dass aktuelle Werte angezeigt werden. Setzen Sie nun in der Basisimplementierung einen Haltepunkt. Wenn die Ausführung dort anhält, werden die aktuellen Werte derjenigen Instanz angezeigt, die im Programmablauf zuerst abgearbeitet wird. Sie können nun sukzessive durch alle Instanzen steppen.

Wenn Sie eine der Instanzen wählen, dann öffnet sich der Editor mit dem Code der Funktionsbaustein-Instanz. Die aktuellen Werte werden in der Deklaration und gegebenenfalls in der Implementierung angezeigt und ständig aktualisiert.

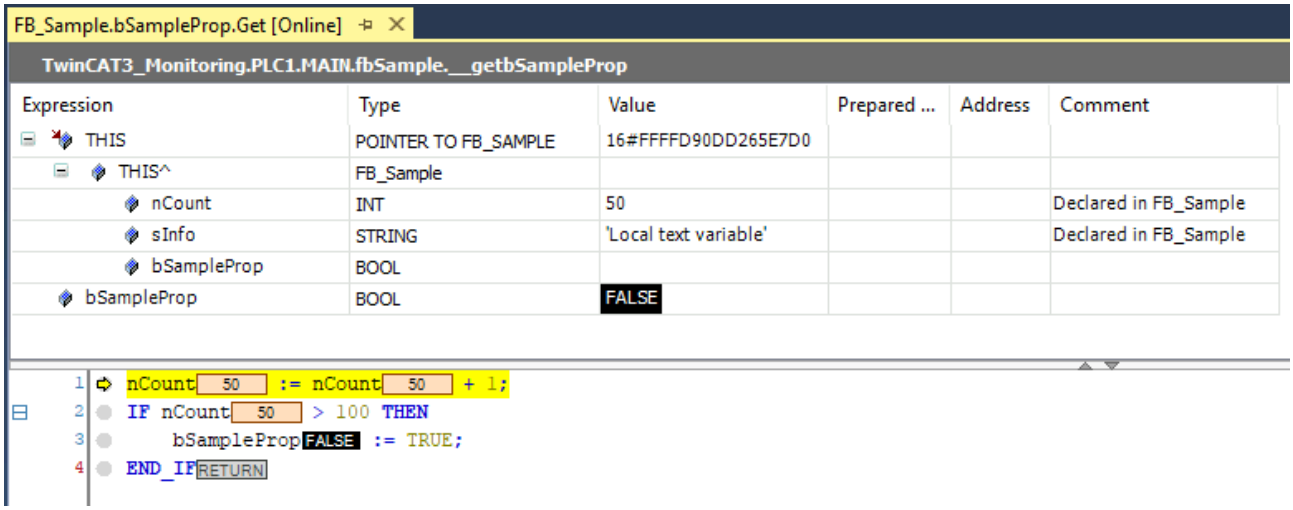
Siehe auch:

- Objekt Funktionsbaustein [▶ 86]
- Haltepunkte verwenden [▶ 222]

Eigenschaft monitoren

Sie können in einem Eigenschaftenobjekt  Variablen monitoren, wenn Sie während des Onlinebetriebs einen Haltepunkt setzen. Wenn dort angehalten wird, werden die aktuellen Werte angezeigt.

Zusätzlich zu den eigenen Werten werden automatisch die Werte der Variablen der übergeordneten Instanz angezeigt. Im Deklarationsteil der Eigenschaft erscheint dafür in der ersten Zeile der Pointer THIS, der auf die übergeordnete Instanz zeigt, mit den Datentypangaben und aktuellen Werten.



Siehe auch:

- Objekt Eigenschaft [▶ 100]

Eigenschaftenzugriff im übergeordneten Programmierobjekt monitoren

Sie können in einem Funktionsbaustein oder Programm zusätzlich zu den Variablenwerten die Werte von untergeordneten Eigenschaften  monitoren.

Fügen Sie dafür in dem untergeordneten Eigenschaftenobjekt in der Deklaration entweder das Pragma {attribute 'monitoring' = 'variable'} oder das Pragma {attribute 'monitoring' = 'call'} ein. Wenn Sie zur Laufzeit das übergeordnete Programm oder die übergeordnete Funktionsbausteininstanz öffnen, dann werden im Editor zusätzlich zu den aktuellen Variablenwerte die aktuellen Eigenschaftenwerte angezeigt.

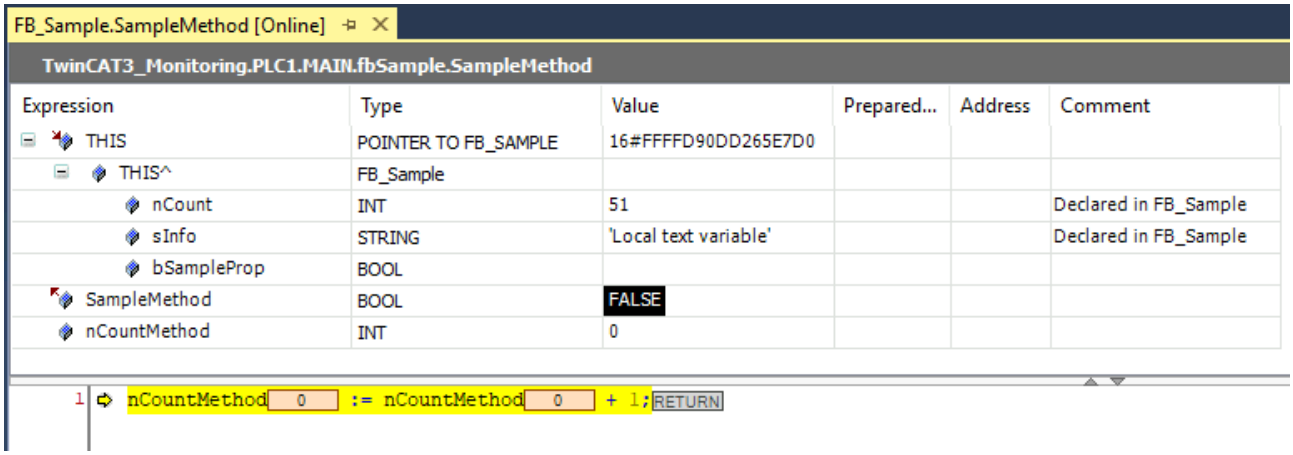
Siehe auch:

- Referenz Programmierung: Pragmas > Attribut 'monitoring' [▶ 846]

Methode monitoren

Sie können in einem Methodenobjekt  Variablen monitoren, wenn Sie während des Onlinebetriebs einen Haltepunkt in der Methode setzen. Wenn dort angehalten wird, werden die aktuellen Werte angezeigt.

Zusätzlich zu den eigenen Werten werden automatisch die Werte der Variablen der übergeordneten Instanz angezeigt. Im Deklarationsteil der Methode erscheint dafür in der ersten Zeile der Pointer THIS, der auf die übergeordnete Instanz zeigt, mit den Datentypangaben und aktuellen Werten.



Siehe auch:

- Objekt Methode [▶ 93]

Funktion monitoren

Sie können in einem Funktionsobjekt Variablen monitoren, wenn Sie während des Onlinebetriebs einen Haltepunkt in der Funktion setzen. Wenn dort angehalten wird, werden die aktuellen Werte angezeigt.

- Objekt Funktion [▶ 83]

Rückgabewert eines Funktionsaufrufs monitoren

Im ST-Editor wird an der Stelle einer POU, an der eine Funktion aufgerufen wird, der aktuelle Rückgabewert der Funktion im Inline-Monitoring angezeigt, wenn Folgendes erfüllt ist:

- es handelt sich um einen Wert, der als 4-byte-numerischer Wert interpretiert werden kann (z. B. INT, SINT oder LINT).
- in der Funktionsdeklaration ist das Pragma {attribute 'monitoring' := 'call'} eingefügt.

Siehe auch:

- Referenz Programmierung: Pragmas > Attribut 'monitoring' [▶ 846]

10.1.2 Überwachungslisten verwenden

Was ist eine Überwachungsliste?

Eine Überwachungsliste ist eine benutzerdefinierte Liste von Projektvariablen, die zum Zweck des Monitorings ihrer Werte in einer Ansicht zusammengefasst werden. Im Onlinebetrieb können Sie in einer Überwachungsliste Variablenwerte schreiben und forcen.

In einem Projekt stehen vier Überwachungslisten **Überwachungsliste <n>** im Menü **PLC > Fenster** zur Befüllung bereit.

Anlegen und Bearbeiten einer Überwachungsliste (Offline- oder Onlinebetrieb)

✓ Ein Projekt ist im Offline- oder Onlinebetrieb geöffnet. In dem Projekt sind Variablen deklariert, die Sie in einer der vier möglichen Überwachungslisten haben möchten.

1. Wählen Sie im Menü **PLC > Fenster** den Befehl **Überwachungsliste <n>**.
   ⇨ Die Ansicht **Überwachungsliste <n>** erscheint. Sie enthält eine leere Tabellenzeile.
2. Geben Sie nach einem Doppelklick auf das Feld in Spalte **Ausdruck** eine zu überwachende Variable ein, händisch oder über die **Eingabehilfe**.
   Syntax: <Gerätename>.<Projektname>.<Objektname>.<Variablenname>
   Beispiel: „TwinCAT_Device.Project5.MAIN.nVar“
   Wenn Sie den Namen einer strukturierten Variablen eintragen, werden im Onlinebetrieb die einzelnen Komponenten automatisch in weiteren Zeilen angezeigt.
3. Definieren Sie nacheinander alle mit dieser Liste zu überwachenden Variablen. Die Reihenfolge können Sie mit drag-and-drop verändern.

⇨ Die Felder **Ausführungszeitpunkt**, **Datentyp**, **Adresse**, **Kommentar** und **Wert** werden automatisch gemäß der Variablendeklaration gefüllt. Das Symbol vor dem Ausdruck zeigt an, ob es sich um eine Eingabevariable, Ausgabevariable oder „normale“ Variable handelt.

Im Onlinebetrieb können Sie eine Überwachungsliste auch mithilfe des Kontextmenübefehls **Zur Überwachungsliste hinzufügen** neu anlegen oder eine bestehende bearbeiten.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Überwachungsliste <n> [▶ 983]

Hinzufügen von Variablen mithilfe des Befehls „Zur Überwachungsliste hinzufügen“ (Onlinebetrieb)

✓ Ein Projekt ist im Betrieb geöffnet. In dem Projekt sind bereits Variablen deklariert, die Sie in eine Überwachungsliste aufnehmen möchten.

1. Öffnen Sie die gewünschte Überwachungsliste mit dem Befehl **Überwachungsliste <n>** im Menü **PLC > Fenster**.
2. Stellen Sie den Cursor im Deklarations- oder Implementierungsteil einer POU auf die Variable und wählen Sie aus dem Kontextmenü den Befehl **Zur Überwachungsliste hinzufügen**.
   ⇨ Der Liste wird ein Eintrag für die gewählte Variable hinzugefügt.
3. Sie können weitere Variablen auf diese Weise hinzufügen, oder durch Eintragen in der Liste im Feld **Ausdruck**, wie oben beschrieben (**Anlegen und Bearbeiten einer Überwachungsliste**).
   ⇨ Die Überwachungslisten werden unmittelbar aktualisiert.

Wenn keine Überwachungsliste geöffnet ist, während Sie den Befehl **Zur Überwachungsliste hinzufügen** auf eine Variable anwenden, wird diese automatisch der Liste „Überwachungsliste 1“ hinzugefügt.

Schreiben und Forcen der Variablenwerte ist auch in den Überwachungslisten möglich. Im Onlinebetrieb erscheint dazu die Spalte **Vorbereiteter Wert**.

Siehe auch:

- Forcen und Schreiben von Variablen [▶ 225]
- Dokumentation TC3 User Interface: Befehl Überwachungsliste <n> [▶ 983]
- Dokumentation TC3 User Interface: Befehl Zur Überwachungsliste hinzufügen [▶ 920]

10.2 Werte ändern mit Rezepturen

Sie verwenden Rezepturen, um gleichzeitig die Werte für einen bestimmten Satz an Variablen (Rezepturdefinition) auf der Steuerung zu verändern oder auszulesen.

Die Grundeinstellungen zu Rezepturen, wie der Speicherort und das Speicherformat, legen Sie im Objekt **Rezepturverwalter** fest. Unterhalb dieses Objekts fügen Sie eine oder mehrere Rezepturdefinitionen ein. Eine **Rezepturdefinition** umfasst eine oder mehrere Rezepturen für die enthaltenen Variablen. Die Rezeptur besteht aus bestimmten Variablenwerten.

Sie können die Rezepturen in Dateien speichern oder direkt von Dateien in die Steuerung schreiben.

Rezepturen können über die Programmieroberfläche von TwinCAT, über Visualisierungselemente oder über das Anwendungsprogramm geladen werden.

Für die Verwendung von Rezepturen in Ihrem Anwendungsprogramm wird empfohlen, diesen Programmabschnitt über einen eigenen Task mit niedriger Priorität aufzurufen.

10.2.1 Objekt Rezepturverwalter

Symbol: 

Der Rezepturverwalter bietet Funktionen zur Verwaltung von benutzerdefinierten Variablenlisten, so genannten Rezepturdefinitionen. Die Rezepturdefinitionen können in „Rezepturdateien“ abgelegt werden.

Wenn Sie dem SPS-Projekt einen Rezepturverwalter hinzufügen, wird im Bibliotheksverwalter automatisch die Bibliothek RecipeManagement ergänzt.

Objekt Rezepturverwalter anlegen

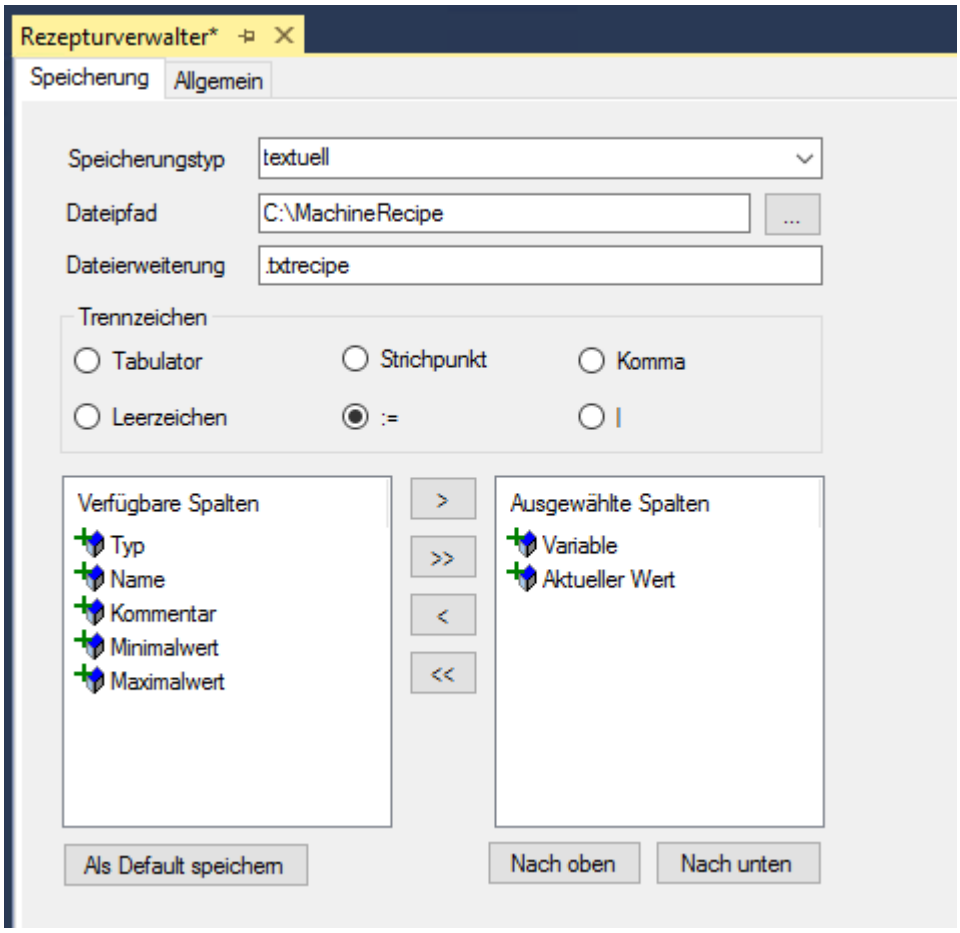
1. Selektieren Sie im **Projektmappen-Explorer** das SPS-Projekt.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Rezepturverwalter...**
   ⇨ Der Dialog **Rezepturverwalter hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein.
4. Klicken Sie auf **Öffnen**.

⇨ Der Rezepturverwalter wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet. Dem Objekt **Rezepturverwalter** können Sie anschließend die Objekte **Rezepturdefinition** hinzufügen.

Struktur des Rezepturverwalters

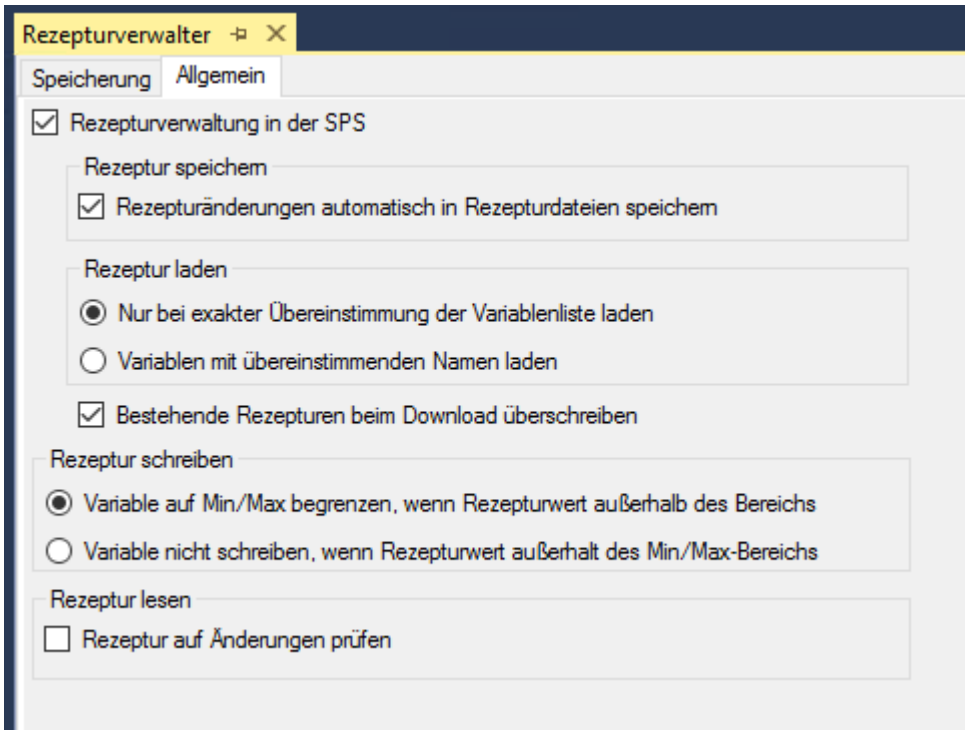
Der Editor des Rezepturverwalters setzt sich aus den beiden Registerkarten **Speicherung** und **Allgemein** zusammen, in denen die Einstellungen für den Rezepturverwalter festgelegt werden.

Registerkarte Speicherung



Speicherungstyp	**textuell**: TwinCAT speichert die Rezeptur in einem lesbaren Format mit den konfigurierten Spalten und Trennzeichen. **binär**: TwinCAT speichert die Rezeptur in einem nicht lesbaren binären Format. Dieses Format benötigt weniger Speicherplatz. Sie können binär gespeicherte Rezepturen nur dann wieder einlesen, wenn Sie die Variablenlisten nicht verändert haben.
Dateipfad	Relativer Pfad auf dem Laufzeitsystem. Dieser Pfad wird auf dem Zielsystem im Verzeichnis der Laufzeitdateien angelegt. TwinCAT legt beim Download auf die Steuerung für jede Rezeptur eine Datei in diesem Verzeichnis ab. Voraussetzung ist, dass die Option **Rezepturverwaltung in der SPS** aktiviert ist. Die Dateien werden bei jedem Neustart des SPS-Projekts in den Rezepturverwalter geladen.
Dateierweiterung	Dateierweiterung für die Rezepturdatei. Daraus ergibt sich der Standardname für Rezepturdateien in der Form <Rezeptur>.<Rezepturdefinition>.<Dateierweiterung>.
Trennzeichen	Trennzeichen zwischen den einzelnen Werten in der gespeicherten Datei.
Verfügbare Spalten Ausgewählte Spalten	Festlegung, welche Informationen in welcher Reihenfolge in die Rezepturdatei gespeichert werden.
Als Default speichern	TwinCAT verwendet die im Dialog vorliegenden Einstellungen ab sofort als Standardeinstellungen.

Registerkarte Allgemein



Rezepturverwaltung in der SPS	☑: Muss aktiviert sein, wenn zur Laufzeit Rezepturen durch Visualisierungselemente oder durch das Anwenderprogramm geladen werden. Falls Rezepturen ausschließlich über die TwinCAT-Programmieroberfläche zur Steuerung übertragen werden, kann die Option deaktiviert werden.

Rezeptur speichern

Rezepturänderungen automatisch in Rezepturdateien speichern	☑: Empfohlene Option, weil sie das „übliche“ Verhalten einer Rezepturverwaltung bewirkt. Der Rezepturverwalter speichert zur Laufzeit die Rezepturen bei einer Änderung automatisch in einer Datei, d. h. die Speicherdateien werden automatisch bei jeder Änderung einer Rezeptur zur Laufzeit aktualisiert. Die Option ist nur wirksam, wenn die Option Rezepturverwaltung in der SPS aktiviert ist.

Rezeptur laden

Nur bei exakter Übereinstimmung der Variablenliste laden	Das Laden der Rezeptur erfolgt nur dann, wenn die Datei alle Variablen aus der Variablenliste der Rezepturdefinition des SPS-Projekts enthält und diese in derselben Reihenfolge sortiert sind. Zusätzliche Einträge am Ende werden ignoriert. Wenn die nötige Übereinstimmung nicht vorliegt, wird der Fehlerstatus ERR_RECIPE_MISMATCH gesetzt (RecipeManCommands.GetLastError).
Variablen mit übereinstimmenden Namen laden	Die Rezepturwerte werden nur für diejenigen Variablen geladen, die in der Rezepturdefinition des SPS-Projekts den gleichen Namen haben wie in der Rezepturdatei. Wenn sich die Variablenlisten in Zusammensetzung und Sortierung unterscheiden, wird kein Fehlerstatus gesetzt. Damit können Rezepturdateien auch dann noch geladen werden, wenn Variablen in der Datei oder in der Rezepturdefinition gelöscht wurden.

Bestehende Rezepturen beim Download überschreiben	☑ : Wenn Rezepturdateien mit dem gleichen Namen auf der Steuerung vorhanden sind, werden diese beim Start des SPS-Projekts mit den konfigurierten Werten aus dem Projekt überschrieben. Wenn stattdessen die Werte aus den bereits vorhandenen Rezepturdateien geladen werden sollen, muss diese Option deaktiviert werden. Voraussetzung: Speicherungstyp ist **textuell** und die Option **Rezepturänderungen automatisch in Rezepturdateien speichern** ist aktiviert.

Rezeptur schreiben

Variable auf Min/Max begrenzen, wenn Rezepturwert außerhalb des Bereichs	Wenn die Rezeptur einen Wert enthält, der außerhalb des in der Definition eingetragenen Wertebereichs liegt, wird statt dieses Werts der definierte minimale oder maximale Wert in die SPS-Variable geschrieben.
Variable nicht schreiben, wenn Rezepturwert außerhalb des Min/Max-Bereichs	Wenn die Rezeptur einen Wert enthält, der außerhalb des in der Definition eingetragenen Wertebereichs liegt, wird kein Wert in die SPS-Variable geschrieben. Sie behält ihren aktuellen Wert.

Rezeptur lesen

Nur bei exakter Übereinstimmung der Variablenliste laden	☑ : Bei jedem Methodenaufruf werden zunächst die aktuellen SPS-Variablenwerte in die Rezeptur eingelesen. Dann wird überprüft, ob sich die Werte geändert haben. Nur, wenn sich die Werte geändert haben, wird die Rezeptur gespeichert, also die Rezepturdatei mit den aktuellen Rezepturen überschrieben. Die Option kann genutzt werden, um die im lokalen Dateisystem liegende Rezepturdatei nur dann zu aktualisieren, wenn sich auf der SPS Rezepturwerte geändert haben. Allerdings wirkt sich das auf die Performance aus, weil für die Prüfung zusätzlicher Code erzeugt wird. ☐ : Bei jedem Methodenaufruf werden zunächst die aktuellen SPS-Variablenwerte in die Rezeptur eingelesen. Dann wird die Rezeptur in der Rezepturdatei im lokalen Dateisystem geschrieben. Da bei jedem Aufruf auf das Dateisystem geschrieben wird, kann die Steuerung belastet werden.

Rezepturen im Onlinebetrieb, wenn die Option **Änderungen in Rezeptur automatisch speichern** aktiviert ist:

Aktionen	Im Projekt definierte Rezepturen	Zur Laufzeit definierte Rezepturen
Online Reset Warm Online Reset Kalt Download	Die Rezepturen aller Rezepturdefinitionen werden mit den Werten aus dem aktuellen Projekt belegt.	Dynamisch erzeugte Rezepturen bleiben unverändert.
Online Reset Ursprung	Die Anwendung wird von der SPS entfernt. Wenn danach ein neuer Download erfolgt, werden die Rezepturen so wiederhergestellt wie bei einem Online Reset Warm.	
Shutdown und Neustart der SPS	Nach dem Neustart werden die Rezepturen erneut aus den automatisch angelegten Dateien geladen. Damit wird der gleiche Zustand wie vor dem Herunterfahren wiederhergestellt.	
Online-Change	Die Rezepturwerte bleiben unverändert. Während der Laufzeit kann eine Rezeptur nur über die Befehle des Funktionsbausteins RecipeManCommands verändert werden.	
Stopp/Start	Bei einem Stopp/Start der SPS bleiben die Rezepturen unverändert.	

Rezepturen im Onlinebetrieb, wenn **Rezepturänderungen automatisch in Rezepturdateien speichern** NICHT aktiviert ist:

Aktionen	Im Projekt definierte Rezepturen	Zur Laufzeit definierte Rezepturen
Online Reset Warm Online Reset Kalt Download	Die Rezepturen aller Rezepturdefinitionen werden mit den Werten aus dem aktuellen Projekt belegt. Allerdings werden diese nur im Speicher gesetzt. Um die Rezepturen in einer Datei zu speichern, muss explizit der Befehl Rezeptur speichern verwendet werden.	Dynamisch erzeugte Rezepturen gehen verloren.
Online Reset Ursprung	Die Anwendung wird von der SPS entfernt. Wenn danach ein neuer Download erfolgt, werden die Rezepturen wiederhergestellt.	Dynamisch erzeugte Rezepturen gehen verloren.
Shutdown und Neustart der SPS	Nach dem Neustart werden die Rezepturen erneut aus den automatisch angelegten Dateien geladen. Damit wird der gleiche Zustand wie vor dem Herunterfahren wiederhergestellt.	
Online-Change	Die Rezepturwerte bleiben unverändert. Während der Laufzeit kann eine Rezeptur nur über die Befehle des Funktionsbausteins RecipeManCommands verändert werden.	
Stopp/Start	Bei einem Stopp/Start der SPS bleiben die Rezepturen unverändert.	

10.2.2 Objekt Rezepturdefinition

Symbol: 

Innerhalb einer Rezepturdefinition definieren Sie verschiedene Werte-Sets für die Variablen, welche Rezepturen genannt werden.

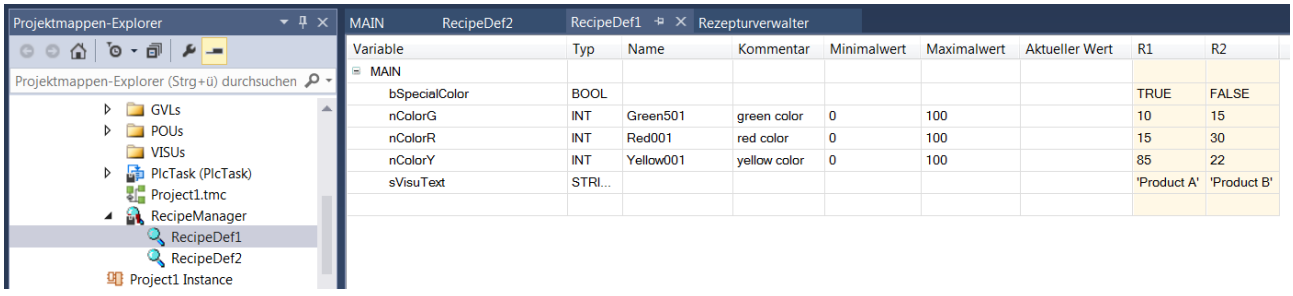
Objekt Rezepturdefinition anlegen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum das Objekt **Rezepturverwalter**.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Rezepturdefinition...**
   ⇨ Der Dialog **Rezepturdefinition hinzufügen** öffnet sich.
3. Geben Sie einen Namen ein.
4. Klicken Sie auf **Öffnen**.

⇨ Die Rezepturdefinition wird unterhalb des Rezepturverwalters hinzugefügt und im Editor geöffnet.

Struktur einer Rezepturdefinition

Im Editor der Rezepturdefinition werden die Werte-Sets tabellarisch dargestellt. Sie können die Ansicht der Rezepturdefinition zwischen der einfachen Ansicht und der strukturierten Ansicht umschalten. In der strukturierten Ansicht stellt TwinCAT die zu einer Struktur zugehörigen Variablen gruppiert dar.



Variable	Name der Variablen
Typ	Wird automatisch eingetragen
Name	Optional
Minimalwert Maximalwert	Wenn der Variablenwert kleiner als der Minimalwert oder größer als der Maximalwert, dann setzt TwinCAT den Wert auf den Minimalwert oder Maximalwert.
Kommentar	Zusätzliche Information, wie beispielsweise die Einheit des Werts.
Aktueller Wert	Aktueller Variablenwert, wird nur im Onlinebetrieb angezeigt.
Rezeptur (z. B. R1)	Variablenwerte der Rezeptur

10.2.3 Rezepturen verwenden

Handhaben von Rezepturen in der TwinCAT-Benutzeroberfläche

Die Programmieroberfläche von TwinCAT bietet Ihnen Befehle zum Erzeugen von Rezepturen sowie zum Lesen/Schreiben im Onlinebetrieb.

Siehe auch:

- Dokumentation TC3 User Interface: Rezepturen [▶ 1098]

Verwenden von Rezepturen in der Anwendung

Sie können Rezepturen zur Laufzeit im Anwenderprogramm oder über Visualisierungselemente nutzen.

Im Anwenderprogramm verwenden Sie die Methoden des Funktionsbausteins RecipeManCommands aus der Bibliothek RecipeManagement. In der Visualisierung erfolgt die Verwendung von Rezepturen über die Eingabekonfiguration (Internes Kommando) von Visualisierungselementen.

Die Rezepturverwaltung liest beim Initialisierungsvorgang die Werte der Variablen, die in der Rezepturdefinition festgelegt sind. Dieser Vorgang findet am Ende der Initialisierungsphase der Applikation statt. Zu diesem Zeitpunkt sind alle Initialwerte der Applikationsvariablen gesetzt. Dies wird durchgeführt, um fehlende Werte aus Rezepturdateien richtig initialisieren zu können.

Siehe auch:

- Bibliothek Recipe Management - RecipeManCommands [▶ 247]

Erstellen einer Rezeptur

1. Selektieren Sie im **Projektmappen-Explorer** das SPS-Projekt.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Rezepturverwalter..**
   ⇨ TwinCAT fügt den Rezepturverwalter zum SPS-Projekt hinzu.
3. Selektieren Sie im SPS-Projektbaum das Objekt **Rezepturverwalter**.
4. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Rezepturdefinition**.
   ⇨ TwinCAT fügt die Rezepturdefinition unterhalb des Rezepturverwalters hinzu.
5. Öffnen Sie den Editor der Rezepturdefinition durch einen Doppelklick auf das Objekt.
6. Führen Sie einen Doppelklick im Editor auf das leere Feld in der Spalte **Variable** aus. Geben Sie den Namen einer Variablen an, für die Sie eine Rezeptur definieren wollen. Die Eingabehilfe steht dazu bereit (Schaltfläche ).
7. Wählen Sie im Menü **Rezeptur** oder im Kontextmenü des Editors den Befehl **Rezeptur einfügen** und geben Sie einen Namen für die neue Rezeptur an (z. B. „MyRec“).
   ⇨ Im Editor erscheint eine Spalte mit dem Rezepturnamen.
8. Geben Sie den Wert der Variable für diese Rezeptur ein.
9. Fügen Sie bei Bedarf weitere Variablen ein.

10. Selektieren Sie einen Variablenwert der Rezeptur und führen Sie über das Menü **Rezeptur** oder das Kontextmenü den Befehl **Rezeptur speichern** aus. Wählen Sie einen Speicherort und einen Dateinamen aus.

⇨ TwinCAT speichert die Rezeptur in dem Format, das im Rezepturverwalter definiert ist.

Überschreiben der impliziten Rezepturdatei

Die implizit verwendeten Rezepturdateien, die als Zwischenablage beim Lesen und Schreiben von Rezepturen nötig sin, dürfen nicht überschrieben werden. D. h. der Name für die Datei muss anders lauten als *<Rezepturname>.<Rezepturdefinitionsname>.txtrecipe*

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Rezeptur einfügen [▶ 1098]
- Dokumentation TC3 User Interface: Befehl Rezeptur speichern [▶ 1099]

Laden einer Rezeptur aus einer Datei

✓ In dem SPS-Projekt gibt es eine Rezepturverwaltung. In einer Rezepturdefinition gibt es eine Rezeptur „MyRec“ mit Variablenwerten. Eine Rezepturdatei *MyRec.txt*, die Einträge für diese Rezeptur enthält, liegt im Dateisystem vor.

1. Öffnen Sie mit einem Doppelklick auf das Objekt **Rezepturdefinition** im SPS-Projektbaum den Tabelleneditor für die Definition der einzelnen Rezepturen.
   ⇨ Sie sehen eine Spalte **MyRec** mit den aktuellen Werten für diese Rezeptur.
2. Bearbeiten Sie die Datei *MyRec.txt* in einem externen Texteditor und ersetzen die Variablenwerte durch andere, die sie in die Rezepturdefinition in TwinCAT laden möchten. Speichern Sie die Datei.
3. Klicken Sie in der Rezepturdefinition in die Spalte **MyRec** und wählen im Menü **Recipe** oder im Kontextmenü den Befehl **Rezeptur laden**.
   ⇨ Der Dialog **Rezeptur laden** öffnet sich.
4. Wählen Sie die Datei *MyRec.txt* aus dem Datei-Explorer zum Laden aus.

⇨ Die Rezepturwerte in der Rezepturdefinition werden entsprechend den in der Datei gelesenen Werten aktualisiert. Wenn Sie durch das Laden der Rezeptur die aktuellen Werte der Rezepturvariablen verändern, erscheint beim nächsten Einloggen eine Abfrage, ob Sie sich mit Online-Change, Download oder ohne Änderungen einloggen möchten.

Beispiel einer Rezepturdatei:

```
MAIN.nVar1:=0
MAIN.nVar2:=2
MAIN.nVar3:=35232
MAIN.sVar4:='first'
MAIN.wsVar5:='123443245'
```

Wenn Sie nur einzelne Variablen der Rezeptur mit neuen Werten überschreiben möchten, entfernen Sie vor dem Laden in der Rezepturdatei die Werte für die restlichen Variablen. Einträge ohne Wertangabe werden nicht eingelesen und somit bleiben diese Variablen auf der Steuerung und im Projekt durch die Aktualisierung unberührt.

Bei Werten vom Datentyp REAL/LREAL wird in manchen Fällen auch der Hexadezimalwert in die Rezepturdatei geschrieben. Dies ist notwendig, damit bei der Rückkonvertierung der exakt identische Wert wiederhergestellt wird. In diesem Fall ändern Sie den Dezimalwert und löschen Sie den Hexadezimalwert.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Rezeptur laden [▶ 1098]

Lesen einer Rezeptur

✓ TwinCAT ist im Onlinebetrieb.

1. Klicken Sie in der Rezepturdefinition in die Rezepturspalte und wählen Sie im Menü **Rezeptur** oder im Kontextmenü den Befehl **Rezeptur lesen** aus.

⇨ TwinCAT überschreibt die Werte der selektierten Rezeptur mit den gelesenen Werten aus der Steuerung. Dabei werden die Werte implizit gespeichert (in einer Datei auf der Steuerung) und gleichzeitig in der Tabelle der Rezepturdefinition angezeigt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Rezeptur lesen [▶ 1099]
- Dokumentation TC3 User Interface: Befehl Rezeptur lesen und speichern [▶ 1100]

Schreiben einer Rezeptur

✓ TwinCAT ist im Onlinebetrieb.

1. Klicken Sie in der Rezepturdefinition in die Rezepturspalte und wählen Sie im Menü **Rezeptur** oder im Kontextmenü den Befehl **Rezeptur schreiben** aus.

⇨ TwinCAT überschreibt die Werte in der Steuerung mit den Werten der selektierten Rezeptur.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Rezeptur schreiben [▶ 1100]
- Dokumentation TC3 User Interface: Befehl Rezeptur laden und schreiben [▶ 1100]

10.2.4 Bibliothek Recipe Management - RecipeManCommands

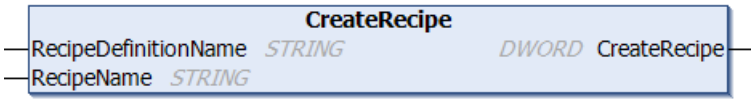
Die Methoden des Funktionsbausteins RecipeManCommands der Bibliothek „Recipe Management“ ermöglichen, die Rezepturen programmatisch zu verwalten.

Die Anwendung erstellt automatisch Rezepturdateien mit Namen *<Rezeptur>.<Rezepturdefinition>.txtrecipe* auf der Steuerung. Sie dienen als Zwischenablage beim Lesen und Schreiben der Rezepturvariablen. Die Option **Rezepturänderungen automatisch in Rezepturdateien speichern** in der Registerkarte **Rezepturverwalter > Allgemein** beeinflusst den Dateizugriff auf diese Dateien.

Wenn die Option **Änderungen in Rezepturen automatisch speichern** aktiviert ist, werden die Rezepturen der Definition in TwinCAT und die impliziten Rezepturdateien auf der Steuerung automatisch gleich gehalten. Die Änderung von Rezepturen führt dann auch zu Dateizugriffen.

Methode CreateRecipe

Die Methode erzeugt eine neue Rezeptur in der angegebenen Rezepturdefinition. Anschließend liest sie die aktuellen SPS-Werte in die neue Rezeptur und speichert sie als Rezepturdatei mit Standardname. Der Standardname ist <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung>.



Eingänge (VAR_INPUT)

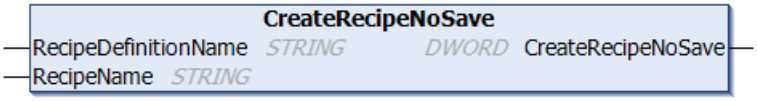
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
CreateRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_ALREADY_EXIST ERR_RECIPE_NOMEMORY ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode CreateRecipeNoSave

Die Methode erzeugt eine neue Rezeptur in der angegebenen Rezepturdefinition. Anschließend liest sie die Istwerte in die neue Rezeptur.



Eingänge (VAR_INPUT)

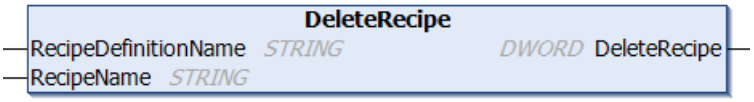
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
CreateRecipeNoSave	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_ALREADY_EXIST ERR_RECIPE_NOMEMORY ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode DeleteRecipe

Die Methode löscht eine Rezeptur aus der Rezepturdefinition.



Eingänge (VAR_INPUT)

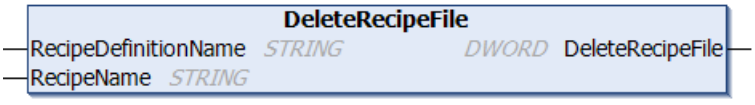
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
DeleteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode DeleteRecipeFile

Die Methode löscht die angegebene Rezepturdatei einer Rezeptur. Die Rezepturdatei muss im Standardnamen <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung> gespeichert sein.



Eingänge (VAR_INPUT)

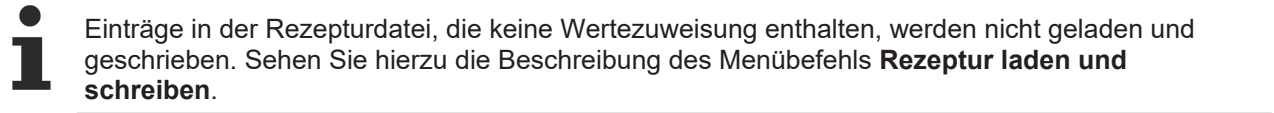
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

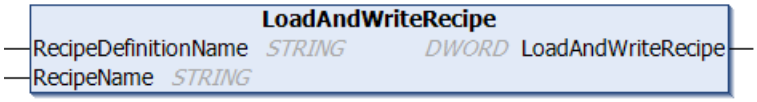
Name	Datentyp	Beschreibung
DeleteRecipeFile	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode LoadAndWriteRecipe

Die Methode lädt eine Rezeptur aus der angegebenen Rezepturdatei. Die Rezepturdatei muss im Standardnamen <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung> gespeichert sein. Anschließend schreibt sie die Rezeptur in die SPS-Variablen.



Einträge in der Rezepturdatei, die keine Wertezuweisung enthalten, werden nicht geladen und geschrieben. Sehen Sie hierzu die Beschreibung des Menübefehls **Rezeptur laden und schreiben**.



Eingänge (VAR_INPUT)

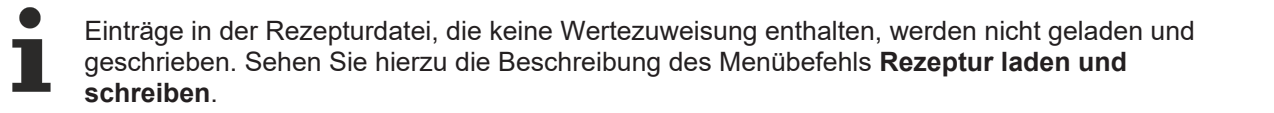
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

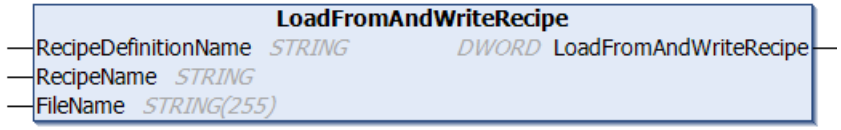
Name	Datentyp	Beschreibung
LoadAndWriteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_RECIPE_MISMATCH ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode LoadFromAndWriteRecipe

Die Methode lädt die angegebene Rezepturdatei in eine Rezeptur. Anschließend schreibt sie die Rezeptur in die SPS-Variablen.



Einträge in der Rezepturdatei, die keine Wertezuweisung enthalten, werden nicht geladen und geschrieben. Sehen Sie hierzu die Beschreibung des Menübefehls **Rezeptur laden und schreiben**.



Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur
FileName	STRING (255)	Name der Datei

Rückgabewert

Name	Datentyp	Beschreibung
LoadFromAndWriteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_RECIPE_MISMATCH ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED ERR_NO_RECIPE_MANAGER_SET ERR_OK

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Rezeptur laden und schreiben [▶ 1100]

Methode LoadRecipe

Die Methode lädt eine Rezeptur aus einer Rezepturdatei. Die Rezepturdatei muss im Standardnamen <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung> gespeichert sein.

> **Einträge in der Rezepturdatei, die keine Wertezuweisung enthalten, werden nicht geladen und geschrieben. Sehen Sie hierzu die Beschreibung des Menübefehls Rezeptur laden und schreiben.**

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
LoadRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_FILE_NOT_FOUND ERR_RECIPE_MISMATCH ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED ERR_NO_RECIPE_MANAGER_SET ERR_OK

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Rezeptur laden und schreiben [▶ 1100]

Methode ReadAndSaveAs

Die Methode liest die aktuellen SPS-Werte aus den Variablen der Rezepturdefinition, und speichert diesen Datensatz in einer Rezepturdatei ohne die existierende Standardrezepturdatei <recipe.recipedefinition.extension> zu ändern.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition. Die in der Rezepturdefinition angegebenen Variablen werden ausgelesen.
FileName	STRING(255)	Name der Datei. In der Datei wird der aktuell ausgelesenen Datensatz als Rezeptur gespeichert.

Rückgabewert

Name	Datentyp	Beschreibung
ReadAndSaveAs	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode ReadAndSaveRecipe

Die Methode liest die aktuellen SPS-Werte in die Rezeptur. Anschließend speichert sie die Rezeptur in eine Rezepturdatei mit Standardname. Der Standardname ist <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung>. Der Inhalt einer eventuell existierenden Datei wird überschrieben.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
ReadAndSaveRecipe	DWORD	Rückgabewert, mögliche Werte: ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode ReadAndSaveRecipeAs

Die Methode liest die aktuellen SPS-Werte in die Rezeptur. Anschließend speichert sie die Rezeptur in eine angegebene Rezepturdatei. Der Inhalt einer eventuell existierenden Datei wird überschrieben.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur
FileName	STRING	Name der Datei

Rückgabewert

Name	Datentyp	Beschreibung
ReadAndSaveRecipeAs	DWORD	Rückgabewert, mögliche Werte: ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode SaveRecipe

Die Methode speichert die Rezeptur in eine Rezepturdatei mit Standardname. Der Standardname ist <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung>. Der Inhalt einer eventuell existierenden Datei wird überschrieben.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
SaveRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_RECIPE_SAVE_ERR ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode ReadRecipe

Die Methode liest die aktuellen SPS-Werte in die Rezeptur ein.

Eingänge (VAR_INPUT)

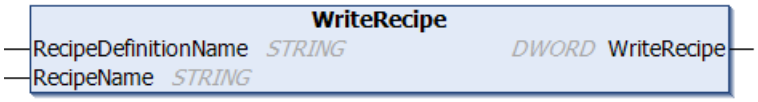
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
ReadRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK Dataserver-Fehler von 16#2000 bis 16#20FF Datasourcedriver-Fehler von 16#2100 bis 16#21FF

Methode WriteRecipe

Die Methode schreibt die Rezeptur in die SPS-Variablen.



Eingänge (VAR_INPUT)

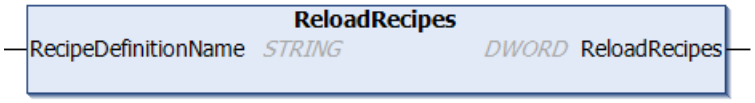
Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName	STRING	Name der Rezeptur

Rückgabewert

Name	Datentyp	Beschreibung
WriteRecipe	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode ReloadRecipes

Die Methode lädt die Liste der Rezepturen von einem Dateisystem. Dabei werden Rezepturen mit dem Standardnamen <Rezeptur>.<Rezepturdefinition>.<Rezepturerweiterung> berücksichtigt, die in dem im Rezepturverwalter definierten Pfad liegen.



Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition

Rückgabewert

Name	Datentyp	Beschreibung
ReloadRecipes	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode GetRecipeCount

Die Methode gibt die Anzahl der Rezepturen einer Rezepturdefinition zurück.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition

Rückgabewert

Name	Datentyp	Beschreibung
GetRecipeCount	INT	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode GetRecipeNames

Die Methode gibt die Rezepturnamen einer Rezepturdefinition zurück.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
pStrings	POINTER TO ARRAY [] OF STRING	Zeiger auf das Array, das die Rezepturnamen enthält.
iSize	INT	Anzahl der Elemente des STRING-Arrays
iStartIndex	INT	Startindex Beispiel: 1

Rückgabewert

Name	Datentyp	Beschreibung
GetRecipeNames	DWORD	Rückgabewert, mögliche Werte: ERR_RECIPE_DEFINITION_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Beispiel:

Es gibt zum Beispiel 50 Rezepturen. Wenn Sie eine Tabelle erzeugen wollen, die 10 Rezepturnamen gleichzeitig anzeigt, müssen Sie ein STRING-Array definieren:

```
strArr: ARRAY[0..9] OF STRING;
```

Korrespondierend zum iStartIndex erhalten Sie dann die Rezepturnamen für den spezifizierten Bereich.

```
iStartIndex := 0; die Namen 0..9 werden zurückgegeben
iStartIndex := 20; die Namen 20..29 werden zurückgegeben
```

In diesem Beispiel gilt:

```
iSize := 10;
```

Methode GetRecipeValues

Die Methode gibt die Werte einer Rezeptur zurück.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName		Name der Rezeptur
pStrings	POINTER TO ARRAY [] OF STRINGS	Zeiger auf Zeichenketten, in denen die Rezepturwerte gespeichert werden.
iSize	INT	Größe des Zeichenfolgen-Arrays.
iStartIndex	INT	Der Startindex.
iStringLength	INT	Die Länge der Zeichenkette im Array.

Rückgabewert

Name	Datentyp	Beschreibung
GetRecipeValues	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Beispiel:

Es gibt zum Beispiel 50 Rezepturen. Wenn Sie eine Tabelle erzeugen wollen, die 10 Rezepturwerte gleichzeitig anzeigt, müssen Sie dazu ein STRING-Array definieren:

```
strArr: ARRAY[0..9] OF STRING;
```

Korrespondierend zum iStartIndex erhalten Sie dann die Rezepturwerte für einen spezifizierten Bereich.

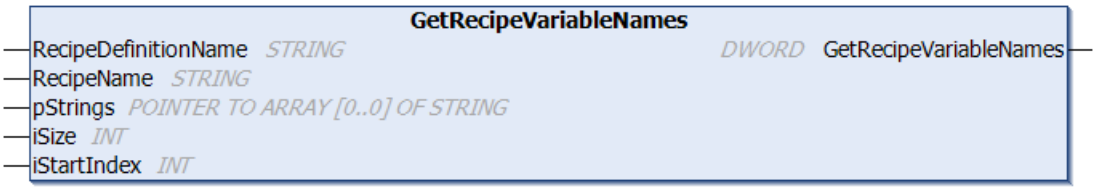
```
iStartIndex := 0; die Werte 0..9 werden zurückgegeben
iStartIndex := 20; die Namen 20..29 werden zurückgegeben
```

In diesem Beispiel gilt:

```
iStringLength := 80;
iSize := 10;
```

Methode GetRecipeVariableNames

Die Methode gibt die Namen der Rezepturvariablen einer Rezeptur wieder.



Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName		Name der Rezeptur
pStrings	POINTER TO ARRAY [] OF STRINGS	Zeiger auf STRING-Datentypen, in denen die Namen der Rezepturvariablen gespeichert werden.
iSize	INT	Größe des STRING-Arrays.
iStartIndex	INT	Der Startindex.

Rückgabewert

Name	Datentyp	Beschreibung
GetRecipeVariableNames	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Beispiel:

Es gibt zum Beispiel 50 Rezepturen. Wenn Sie eine Tabelle erzeugen wollen, die 10 Variablennamen einer Rezeptur gleichzeitig anzeigt, müssen Sie ein STRING-Array definieren:

```
strArr: ARRAY[0..9] OF STRING;
```

Korrespondierend zum iStartIndex erhalten Sie dann die Variablennamen der Rezeptur für den spezifizierten Bereich.

```
iStartIndex := 0; die Namen 0..9 werden zurückgegeben
iStartIndex := 20; die Namen 20..29 werden zurückgegeben
```

In diesem Beispiel gilt:

```
iSize := 10;
```

Methode SetRecipeValues

Die Methode setzt die Rezepturwerte einer Rezeptur.

Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
RecipeDefinitionName	STRING	Name der Rezepturdefinition
RecipeName		Name der Rezeptur
pStrings	POINTER TO ARRAY [] OF STRINGS	Zeiger auf STRING-Datentypen, in denen die Rezepturwerte gespeichert werden.
iSize	INT	Größe des STRING-Arrays.
iStartIndex	INT	Der Startindex.

Rückgabewert

Name	Datentyp	Beschreibung
SetRecipeValues	DWORD	ERR_RECIPE_DEFINITION_NOT_FOUND ERR_RECIPE_NOT_FOUND ERR_NO_RECIPE_MANAGER_SET ERR_OK

Beispiel:

Es gibt zum Beispiel 50 Rezepturen. Wenn Sie eine Tabelle erzeugen wollen, die 10 Rezepturwerte gleichzeitig setzt, müssen Sie ein STRING-Array definieren:

```
strArr: ARRAY[0..9] OF STRING;
```

Korrespondierend zum iStartIndex können Sie dann die Rezepturwerte für einen spezifizierten Bereich festlegen.

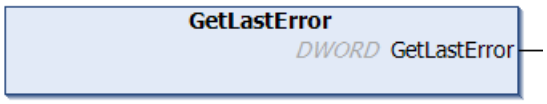
```
iStartIndex := 0; Die Werte 0..9 werden gesetzt
iStartIndex := 20; Die Werte 20..29 werden gesetzt
```

In diesem Beispiel gilt:

```
iStringLength := 80;
iSize := 10;
```

Methode GetLastError

Die Methode gibt den letzten Fehler der vorherigen Operation zurück.



Rückgabewert

Name	Datentyp	Beschreibung
GetLastError	DWORD	Rückgabewert, mögliche Werte: ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode ResetLastError

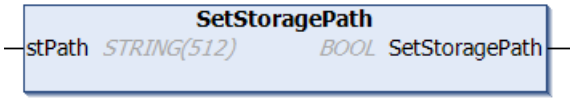
Die Methode setzt den letzten Fehler zurück.

Rückgabewert

Name	Datentyp	Beschreibung
ResetLastError	DWORD	ERR_NO_RECIPE_MANAGER_SET ERR_OK

Methode SetStoragePath

Die Methode dient dem Einstellen des Speicherpfads für die Rezepturdatei. Sie überschreibt die Pfadangabe im Dialog Speicherung des Rezepturverwalters.



Eingänge (VAR_INPUT)

Name	Datentyp	Beschreibung
stPath	STRING	Dateipfad, Beispiel: • D:/recipefiles/

Rückgabewert

Name	Datentyp	Beschreibung
stPath	BOOL	TRUE (Pfad wurde gesetzt) FALSE Mögliche Fehler: ERR_OK ERR_NO_RECIPE_MANAGER_SET

Rückgabewerte

Die Rückgabewerte der oben beschriebenen Funktionen sind in der GVL ReturnValues enthalten.

Es handelt sich um InOut-Konstanten vom Datentyp UDINT.

Name	Initialisierungswert	Kommentar
ERR_OK	16#0	Die Operation wurde erfolgreich durchgeführt.
ERR_FAILED	16#1	Die Operation schlug fehl.
ERR_PARAMETER	16#2	Falscher Parameter
ERR_NOTINITIALIZED	16#3	Das Dataserver-Objekt ist nicht initialisiert. Der Dataserver wird benötigt, wenn die Rezepturverwaltung in Kombination mit TwinCAT HMI verwendet wird.
ERR_NOTIMPLEMENTED	16#C	Der Dataserver implementiert nicht die Schnittstelle IDataServer4, die benötigt wird, wenn die Rezepturverwaltung in Kombination mit TwinCAT HMI verwendet wird.
ERR_NO_OBJECT	16#10	Nicht alle Variablen einer Rezepturdefinition können über den Dataserver geschrieben werden. Nur die gültigen Variablen werden geschrieben.
ERR_NOMEMORY	16#11	Der Dataserver hat nicht genügend Speicher.
ERR_RECIPE_FILE_NOT_FOUND	16#4000	Die Rezepturdatei wurde nicht gefunden.
ERR_RECIPE_MISMATCH	16#4001	Der Inhalt der Rezepturdatei passt nicht zur aktuellen Rezeptur. Dieser Fehler wird nur ausgegeben, wenn der Speicherungstyp **textuell** ist (siehe Editor **Rezepturverwalter**, Registerkarte **Speicherung**, **Speicherungstyp**) und wenn ein Variablenname in der Datei nicht mit dem Variablenname in der Rezepturdefinition übereinstimmt. Die Rezepturdatei wird nicht geladen, wenn dieser Fehler auftritt. Mögliche Ursachen: Eine Variable wurde aus der Rezepturdefinition im Projekt entfernt.
ERR_RECIPE_SAVE_ERR	16#4002	Der Speichervorgang schlug fehl. Mögliche Ursachen • Die Datei konnte nicht angelegt oder geöffnet werden, weil die Festplatte voll ist. • Der konfigurierte Dateipfad existiert nicht (siehe Editor **Rezepturverwalter**, Registerkarte **Speicherung**, **Dateipfad**). • Die konfigurierte Dateierweiterung ist vom Laufzeitsystem nicht zugelassen (siehe Editor Rezepturverwalter, Registerkarte **Speicherung**, **Dateierweiterung**)
ERR_RECIPE_NOT_FOUND	16#4003	Die Rezeptur existiert nicht.
ERR_RECIPE_DEFINITION_NOT_FOUND	16#4004	Die Rezepturdefinition existiert nicht.
ERR_RECIPE_ALREADY_EXIST	16#4005	Die Rezeptur existiert bereits in der Rezepturdefinition. Legen Sie eine neue Rezeptur unter einem anderen Namen an.
ERR_NO_RECIPE_MANAGER_SET	16#4006	Der globale Rezepturverwalter ist nicht angelegt. Mögliche Ursache • Die Option **Rezepturverwaltung in der SPS** im Editor des Rezepturverwalters, Registerkarte **Allgemein** des aktuellen SPS-Projekts ist nicht gesetzt.
ERR_RECIPE_NOT_ALL_VARIABLES_WERE_LOADED	16#4007	Die Rezepturdefinition enthält mehr Variablen als die Rezepturdatei. In diesem Fall werden die Variablenwerte der Rezepturdatei auf jeden Fall geschrieben. Dies ist nur eine Information, kein tatsächlicher Fehler.
ERR_RECIPE_NOMEMORY	16#4008	Die Rezepturdefinition hat keinen freien Speicher, um eine neue Rezeptur anzulegen. Mögliche Ursachen • Die Option **Rezepturänderungen automatisch in Rezepturdateien speichern** im Editor des Rezepturverwalters, Registerkarte **Allgemein** des aktuellen SPS-Projekts ist nicht gesetzt. • In diesem Fall sind nur 50 Rezepturen pro Rezepturdefinition möglich. Wenn Die Option **Rezepturänderungen automatisch in Rezepturdateien speichern** gesetzt ist, kann der Fehler nicht auftreten. Wenn die Festplatte voll ist, wird der Fehler ERR_RECIPE_SAVE_ERR ausgegeben.

Name	Initialisierungswert	Kommentar
ERR_RECIPE_MANAGER_LOCKED_DURING_ONLINE_CHANGE	16#4009	Der Rezepturverwalter war während des Online-Change blockiert. Mögliche Ursachen: Einige der RecipeMan-Befehle, die während eines Online-Change ausgeführt werden sollten, wurden nicht ausgeführt.
ERR_SOURCE_EXHAUSTED	16#40A0	Verwendet für UTF8 Helper
ERR_TARGET_EXHAUSTED	16#40A1	Verwendet für UTF8 Helper
ERR_SOURCE_ILLEGAL	16#40A2	Verwendet für UTF8 Helper

10.3 Fehleranalyse mit Core Dump

Ein Core Dump ist ein Speicherauszug der SPS-Projektdaten.

Verfügbar ab TC3.1 Build 4024.11

Core Dump nur nutzbar mit zugehöriger Compile-Info-Datei

Wenn Sie eine Core Dump-Datei archivieren oder abspeichern, beachten Sie bitte, dass zum Laden eines Core Dumps das zugehörige Projekt und die zugehörige Compile-Info-Datei (*.compileinfo-Datei, die z.B. beim Erstellen des Projekts im „_CompileInfo“-Ordner abgelegt wird) vorliegen müssen. Falls dies nicht der Fall ist, kann TwinCAT den Dump später nicht mehr verwenden.

Bitte beachten Sie hierzu auch die Einstellungsmöglichkeiten auf der Registerkarte Settings [▶ 969]. Mit Hilfe der Einstellung **Core Dump** können Sie konfigurieren, ob die Core Dump-Datei, die sich möglicherweise im Projektverzeichnis befindet, zusammen mit den verfügbaren Compile-Info-Dateien in einem TwinCAT-Dateiarchiv gespeichert werden soll.

Erzeugen eines Core Dumps

1) Automatische Erzeugung

Laufzeitsysteme legen im Fall eines Ausnahmefehlers automatisch einen Core Dump auf dem Zielsystem ab, wenn das zugehörige SPS-Projekt gerade nicht in einer Entwicklungsumgebung eingeloggt ist. Dieser Core Dump wird als *.core-Datei im Boot-Ordner des Zielsystems abgelegt (standardmäßig unter C:\TwinCAT\3.1\Boot\Plc). Die automatische Erzeugung eines Core Dumps erfolgt auch dann, wenn der Ausnahmefehler innerhalb des Codes einer FB_init-/FB_reinit-/FB_Exit-Methode auftritt (ab TC3.1 Build 4024.25).

Warnungsmeldung, wenn Core Dump auf Zielsystem verfügbar ist

Wenn Sie ein SPS-Projekt einloggen und auf dem ausgewählten Zielsystem ein Core Dump vorliegt, wird im Meldungsfenster eine Warnung ausgegeben, dass auf dem Gerät ein Core Dump verfügbar ist.

Wenn die Warnung beim Einloggen des Projekts nicht mehr ausgegeben werden soll (z. B. weil Sie die Analyse dieser Core Dump-Datei abgeschlossen haben oder weil Sie die Datei anderweitig archiviert haben), können Sie die entsprechende Core Dump-Datei vom Zielsystem löschen (standardmäßig unter < TC3.1.4026.0: *C:\TwinCAT\3.1\Boot\Plc*; >=TC3.1.4026.0: *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc*).

2) Manuelle Erzeugung

Im Onlinebetrieb können Sie auch explizit einen Core Dump erzeugen lassen, wenn das SPS-Projekt gerade an einem Haltepunkt steht oder ein Ausnahmefehler aufgetreten ist. In diesem Fall legt TwinCAT die Core Dump-Datei nur im Projektverzeichnis und nicht auf dem Zielsystem ab. Die erzeugte Core Dump-Datei wird direkt im SPS-Projektverzeichnis abgelegt (<SPS-Projektname>.<SPS-Projekt-GUID>.core).

Laden eines Core Dumps

Im Onlinebetrieb können Sie den Core Dump vom Zielsystem ins Projektverzeichnis laden (mit dem Befehl Core Dump erzeugen [▶ 995]). Außerdem können Sie im Offlinebetrieb einen Core Dump vom Projektverzeichnis ins Projekt laden (mit dem Befehl Core Dump laden [▶ 996]). Sie erhalten dann eine Onlineansicht des SPS-Projekts mit den Daten und Werten zum Zeitpunkt des Ausnahmefehlers bzw. zum Zeitpunkt des Erzeugens der Core Dump-Datei.

In der Onlineansicht, die TwinCAT beim Laden des Core Dumps ins Projekt erzeugt, erscheinen Menübefehle als verfügbar, die in diesem Status nicht wirksam sind. Bei Anwählen eines solchen Befehls erhalten Sie eine entsprechende Meldung.

Außerdem können Sie die Core Dump-Ansicht nur über den Befehl Core Dump schließen [▶ 997] wieder schließen. Der Befehl Ausloggen ist in dieser Ansicht nicht wirksam.

Archivierung

Wenn die Core Dump-Datei, die sich im Projektverzeichnis befindet, im Projektarchiv enthalten sein soll, aktivieren Sie die Option „Core Dump“ bei der Konfiguration des File-Archiv-Inhalts (siehe Dokumentation Referenz Benutzeroberfläche > Projekt > SPS-Projekteinstellungen > Registerkarte Settings [▶ 969]).

Nachfolgend sind zwei beispielhafte Arbeitsabläufe beschrieben.

Core Dump zur Analyse vom Zielsystem ins Projekt laden

Voraussetzung:

✓ Mit der Entwicklungsumgebung haben Sie das Projekt geöffnet, das auf dem Zielsystem einen Ausnahmefehler produziert hat. Das SPS-Projekt befindet sich in der Entwicklungsumgebung im Onlinebetrieb.

1. Laden Sie mit dem Befehl Core Dump erzeugen [▶ 995] den Core Dump vom Zielsystem.
   ⇒ TwinCAT kopiert die Core Dump-Datei mit der Benennung <SPS-Projektname>.<SPS-Projekt-GUID>.core ins lokale SPS-Projektverzeichnis.
2. Loggen Sie das SPS-Projekt aus.
3. Laden Sie mit dem Befehl Core Dump laden [▶ 996] den gewünschten Core Dump ins Projekt.
   ⇒ TwinCAT stellt eine Onlineansicht des SPS-Projekts dar. Sie sehen die Variablenwerte und Aufrufliste zum Fehlerzeitpunkt.
4. Nach Abschluss der Core Dump-Analyse wählen Sie den Befehl Core Dump schließen [▶ 997].

⇨ TwinCAT schließt die Core Dump-Ansicht des SPS-Projekts. Die Entwicklungsumgebung erscheint wieder mit den Ansichten des normalen Offlinebetriebs.

Manuelles Erzeugen eines Core Dumps des eingeloggten SPS-Projekts

Voraussetzung:

✓ Ein SPS-Projekt befindet sich in der Entwicklungsumgebung im Onlinebetrieb. Das SPS-Projekt steht gerade an einem Haltepunkt oder es ist ein Ausnahmefehler aufgetreten.

1. Wählen Sie den Befehl Core Dump erzeugen [▶ 995].

⇨ TwinCAT erzeugt einen neuen Core Dump und legt die Datei mit dem Namen <SPS-Projektname>.<SPS-Projekt-GUID>.core im lokalen SPS-Projektverzeichnis an.

Siehe auch:

- Dokumentation TC3 User Interface: Referenz Benutzeroberfläche > PLC > Core Dump >
  - Befehl Core Dump erzeugen [▶ 995]
  - Befehl Core Dump laden [▶ 996]
  - Befehl Core Dump schließen [▶ 997]

10.4 SPS-Operationssteuerung

Verfügbar ab TC3.1 Build 4026

Es liegt in Ihrer Verantwortung, dass Laufzeitsystemdienste in sicheren Applikationszuständen aktiviert und nur in kritischen deaktiviert sind.

Eine Anlage oder ein Projekt kann zur Laufzeit in einen sensiblen Zustand kommen, in dem störende Aktionen die gesamte Maschine oder Anlage gefährden können. Sie können aber in diesem Zustand bestimmte Befehle unterdrücken und gefährliche Aktionen verhindern. Hierfür steht Ihnen der globale Datentyp PlcAppSystemInfo zur Verfügung.

Beispiele von TwinCAT-Befehlen, deren Ausführung unterdrückt werden kann:

- Werte schreiben, Werte forcen
- Haltepunkt setzen
- Download, Online-Change

Wenn zur Laufzeit des Projekts ein Laufzeitsystemdienst angefordert wird, der aber gerade deaktiviert ist, erhalten Sie in TwinCAT eine Meldung darüber.

Verwendung von PlcAppSystemInfo zur Operationssteuerung

Sie können Operationen über die Variable _AppInfo.Flags vom Typ DWORD aktivieren oder deaktivieren. _AppInfo ist eine Instanz des Typs PlcAppSystemInfo.

Operation	Bit von _AppInfo.Flags	Beispielzugriff
Werte schreiben deaktivieren	Bit 0	`_AppInfo.Flags.0 := TRUE;`
Werte forcen deaktivieren	Bit 1	`_AppInfo.Flags.1 := TRUE;`
Haltepunkt setzen deaktivieren	Bit 2	`_AppInfo.Flags.2 := TRUE;`
Download deaktivieren	Bit 3	`_AppInfo.Flags.3 := TRUE;`
Online Change Deaktivieren	Bit 4	`_AppInfo.Flags.4 := TRUE;`

Siehe auch:

- Bitzugriff auf Variablen [▶ 788]

Sehen Sie dazu auch

Bitzugriff auf Variablen [▶ 788]

11 SPS-Projekt auf der SPS aktualisieren

Situation: Sie sind dabei, ein SPS-Projekt auf die Steuerung zu laden, das sich von dem bereits dort liegenden unterscheidet. In diesem Fall erscheint ein Meldungsfenster, in dem Sie auswählen können, wie Sie das geänderte SPS-Projekt auf die Steuerung übertragen möchten: Download und Online-Change.

- Ein Download führt zu einem erneuten Übersetzen des SPS-Programms. Dabei wird neben einer Syntax-Prüfung auch Programmcode erzeugt und auf die Steuerung geladen. Dies führt zu einem Stopp des laufenden Programms. Ein Download ist die empfohlene Art der Datenübertragung, da aufgrund des Programmstopps und der Neuinitialisierung immer ein definierter Ausgangzustand geschaffen wird.
- Bei einem Online-Change werden nur die geänderten Teile neu in die Steuerung geladen. Dabei wird ein laufendes Programm nicht angehalten. Führen Sie einen Online-Change nur bei kleinen Änderungen des SPS-Projekts aus. Bei umfangreichen Änderungen kann das Verhalten eines Programms nicht sicher vorhergesagt werden.

Änderungsübersicht

Möglichkeit 1:

Das oben beschriebene Meldungsfenster enthält außerdem einen **Details**-Button. Über diesen Button können Sie das **Applikationsinformation**-Fenster öffnen, das eine Prüfung der Unterschiede zwischen dem aktuellen SPS-Projekt und dem SPS-Projekt auf der Steuerung ermöglicht. Dabei geht es um den Vergleich der Anzahl von Bausteinen, der Daten und der Speicherorte.

Das Applikationsinformation-Fenster enthält eine grobe Beschreibung der Unterschiede, beispielsweise:

- Deklaration von MAIN geändert
- Variable fbMyNewInstance in MAIN eingefügt
- Anzahl der Methoden/Aktionen von FB_Sample geändert

Wenn die Einstellung **Download Applikationsinfo** (SPS-Projekteigenschaften, Kategorie Übersetzen [▶ 951]) aktiviert ist, wird zudem eine erweiterte Prüfung der Unterschiede zwischen dem aktuellen SPS-Projekt und dem SPS-Projekt auf der Steuerung ermöglicht. Die erweiterte Prüfungsmöglichkeit besteht darin, dass das **Applikationsinformation**-Fenster den zusätzlichen Reiter **Onlinevergleich** enthält, welcher eine Baumvergleichsansicht zeigt. Anhand dieser können Sie erkennen, welche POUs geändert, gelöscht oder hinzugefügt wurden. Der zusätzliche Reiter erscheint, wenn Sie den blau unterstrichenen Befehl im unteren Bereich des Applikationsinformation-Fensters ausführen („Applikation nicht aktuell. Code jetzt generieren, um den Onlinevergleich anzuzeigen?“).

Diese Baumvergleichsansicht bietet nur eine rudimentäre Übersicht. Für einen detaillierten Vergleich des aktuellen SPS-Projekts und dem SPS-Projekt auf der Steuerung sehen Sie bitte Möglichkeit 2.

Möglichkeit 2:

Voraussetzung: Die Quellcode-Dateien des SPS-Projekts wurden mit auf das Zielsystem übertragen (konfigurierbar über die SPS-Projekteinstellungen, Registerkarte Settings [▶ 969]).

Mit Hilfe des TwinCAT Project Compare Tools kann ein detaillierter Vergleich zwischen dem aktuellen TC3-Projekt und dem TC3-Projekt auf der Steuerung durchgeführt werden (Befehl <TwinCAT-Projektname> mit dem Zielsystem vergleichen... [▶ 1114]). Dabei wird eine Vergleichs-Baumansicht geöffnet, über die die gewünschten SPS-Editoren in einem Vergleichsfenster geöffnet werden können. Unterschiede werden hierbei farblich hervorgehoben. Auch die (partielle) Übernahme vom Quellcode des Zielsystem-Projekts ist möglich (Merge).

Siehe auch:

- Dokumentation Referenz Benutzeroberfläche: Befehl Laden [▶ 999]
- Dokumentation Referenz Benutzeroberfläche: Befehl Einloggen [▶ 1001]
- Dokumentation Referenz Benutzeroberfläche: Befehl Online-Change [▶ 1000]

11.1 Ausführen eines Online-Change

TwinCAT bietet Ihnen automatisch einen Online-Change an, wenn Sie mit einem SPS-Projekt einloggen, das bereits auf der Steuerung vorhanden ist, aber seit dem letzten Download im Programmiersystem verändert wurde. Bei diesem Vorgang werden nur die geänderten Teile neu in die Steuerung geladen. Ein laufendes Programm auf der Steuerung wird beim Online-Change nicht angehalten.

⚠ WARNUNG

Sach- und Personenschäden durch unerwartetes Verhalten der Maschine oder Anlage

Ein Online-Change verändert das laufende Anwendungsprogramm und bewirkt keinen Neustart. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

- Stellen Sie sicher, dass der neue Programmcode das gewünschte Verhalten des gesteuerten Systems bewirkt.

Projektspezifische Initialisierungen

Wenn ein Online-Change durchgeführt wird, werden die projektspezifischen Initialisierungen (Referenzfahrt etc.) nicht ausgeführt, weil die Maschine ihren Status beibehält. Aus diesem Grund hat der neue Programmcode möglicherweise nicht den gewünschten Effekt.

Schwerwiegende Änderungen im Downloadcode

Wenn der Online-Change schwerwiegende Änderungen im Downloadcode bewirkt (z. B. Verschieben von Variablen nötig), informiert ein Dialog über die Effekte und ermöglicht, den Online-Change abzubrechen.

Schneller Online-Change

Für kleine Änderungen (z. B. kleine Änderung im Implementierungsbereich und Verschieben von Variablen nicht nötig) wird ein „schneller Online-Change“ durchgeführt. In diesem Fall wird nur der jeweils geänderte Baustein übersetzt und nachgeladen. Insbesondere wird in dem Fall kein Initialisierungscode erzeugt. Das bedeutet, dass auch kein Code zur Initialisierung von Variablen mit dem Attribut 'init_on_onlchange' erzeugt wird. In der Regel wird das keine Auswirkungen haben, da das Attribut meist dazu verwendet wird, um Variablen mit Adressen zu initialisieren, es kann aber beim schnellen Online-Change nicht dazu kommen, dass eine Variable ihre Adresse ändert.

Um die Wirkung des Attributs init_on_onlchange auf den gesamten Applikationscode sicherzustellen, schalten Sie den schnellen Online-Change mithilfe der Compiler-Definition no_fast_online_change generell für das SPS-Projekt aus. Fügen Sie die Definition zu diesem Zweck in den Eigenschaften des SPS-Projekts in der Kategorie Übersetzen [▶ 951] ein.

Keine Wirkung des Attributs 'init_on_onlchange' bei einzelnen FB-Variablen

Das Attribut 'init_on_onlchange' [▶ 843] wirkt nur bei globalen Variablen, Programmvariablen und lokalen statischen Variablen von Funktionsbausteinen.

Um einen Funktionsbaustein bei einem Online Change neu zu initialisieren muss die Funktionsbausteininstanz mit dem Attribut deklariert werden. Für eine einzelne Variable in einem Funktionsbaustein wird das Attribut nicht ausgewertet.

Zeigervariablen

Zeigervariablen (Pointer) behalten ihren Wert aus dem letzten Zyklus. Wenn ein Zeiger auf eine Variable zeigt, die durch den Online-Change ihre Größe verändert hat, wird der Wert nicht mehr korrekt geliefert. Stellen Sie sicher, dass Zeigervariablen in jedem Zyklus erneut zugewiesen werden.

Überwachungsfunktionen

Nach dem Entfernen impliziter Überwachungsfunktionen, wie beispielsweise CheckBound, ist kein Online-Change möglich, nur ein Download. Eine entsprechende Meldung wird ausgegeben.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Online-Change [▶ 1000]

Online-Change beim Einloggen

✓ Die Verbindungseinstellungen der Steuerung sind korrekt eingestellt. Die Anwendungsprogamme im Projekt und auf der Steuerung sind identisch. Das Programm auf der Steuerung läuft. Das SPS-Projekt ist ausgeloggt.

1. Verändern Sie Ihr SPS-Projekt.
2. Wählen Sie im Menü **PLC** oder in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** den Befehl **Einloggen**.
   ⇨ Ein Dialog erscheint, mit dem Hinweis, dass die Applikation seit dem letzten Download geändert wurde.
3. Wählen Sie die Option **Mit Online-Change einloggen** und klicken Sie auf **OK**.

⇨ Die Änderung wird auf die Steuerung geladen. Das laufende Programm auf der Steuerung wird dabei nicht angehalten. Das SPS-Projekt ist eingeloggt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Einloggen [▶ 1001]

Online-Change im eingeloggten Zustand (Onlinebetrieb)

✓ Die Verbindungseinstellungen der Steuerung sind korrekt eingestellt. Die Anwendungsprogramme im Projekt und auf der Steuerung sind identisch. Das Programm auf der Steuerung läuft. Das SPS-Projekt ist eingeloggt.

1. Selektieren Sie ein Objekt im SPS-Projektbaum. Hier wählen Sie am besten eine POU oder GVL.
2. Wählen Sie im Kontextmenü den Befehl **Objekt (offline) bearbeiten**.
   ⇨ Das Objekt öffnet sich im Editor.
3. Verändern Sie das Objekt. Hier können Sie beispielsweise eine neue Variable deklarieren oder eine Wertzuweisung ändern.
4. Wählen Sie im Menü **PLC** den Befehl **Online-Change**.
   ⇨ Sie erhalten eine Abfrage, ob Sie den Online-Change wirklich ausführen wollen.
5. Bestätigen Sie den Dialog mit **Ja**.

⇨ Die Änderung wird auf die Steuerung geladen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Objekt (offline) bearbeiten [▶ 926]
- Dokumentation TC3 User Interface: Befehl Online-Change [▶ 1000]

Was verhindert einen Online-Change?

Es gibt Aktionen in TwinCAT, nach denen ein Online-Change auf einer Steuerung nicht mehr möglich ist. Danach ist immer ein Download [▶ 265] des Projekts erforderlich. Ein typischer Fall sind die Aktionen **Bereinigen** und **Alles bereinigen**, die die beim letzten Download abgelegte Übersetzungsinformation löschen. Aber es gibt auch „normale“ Editieraktionen, die dazu führen, dass beim nächsten Einloggen ein Online-Change nicht mehr möglich ist. Folgende Aktionen können einen Online-Change verhindern:

Checkfunktionen	Aktivieren oder Entfernen einer Checkfunktion [▶ 172] (CheckBounds, CheckRange, CheckDiv etc.). Änderung in der Schnittstelle einer Checkfunktion (auch das Einfügen und Löschen von lokalen Variablen).
Taskkonfiguration	Ändern in den Konfigurationseinstellungen.
Projekteinstellungen	Kategorie Übersetzen [▶ 951]: In der Sektion **Einstellungen** (Konstanten ersetzen), Änderung in den Compiler-Defines Kategorie Common [▶ 948]: ID-Änderungen in TwinCAT-Dateien minimieren.
Bausteineigenschaften	Änderung der Option Externe Implementierung
Funktionsbaustein	Ändern des Basisbausteins eines Funktionsbausteins (EXTENDS [▶ 201] FB_Base), auch das Einfügen oder Löschen eines solchen Basisbausteins. Änderung in der Schnittstellenliste (IMPLEMENTS [▶ 208] I_Sample). Ausnahme: Hinzufügen einer neuen Schnittstelle am Ende der Liste.
Datentyp	Ändern des Datentyps einer Variable von einem benutzerdefinierten Datentyp zu einem anderen benutzerdefinierten Datentyp (beispielsweise von TON zu TOF). Ändern des Datentyps von einem benutzerdefinierten Datentyp zu einem Basisdatentyp (beispielsweise von TON zu TIME). Hinweis: Als Workaround ändern Sie gleichzeitig mit dem Datentyp immer auch den Namen der Variablen. Dann wird die Variable als neue Variable initialisiert und die alte entfernt. Ein Online-Change ist danach möglich.

11.2 Ausführen eines Downloads

Ein Download des SPS-Projekts bewirkt ein Übersetzen des aktiven Projekts. Dabei wird neben einer Syntax-Prüfung auch Programmcode erzeugt und auf die Steuerung geladen. Ein laufendes Programm auf der Steuerung wird beim Download angehalten.

Beim Download werden alle Variablen mit Ausnahme von persistenten Variablen neu initialisiert.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Laden [▶ 999]

Download beim Einloggen

✓ Die Verbindungseinstellungen der Steuerung sind korrekt eingestellt. Die Applikationen im Projekt und auf der Steuerung sind identisch. Das Programm auf der Steuerung läuft. Das SPS-Projekt ist ausgeloggt.

1. Verändern Sie Ihr SPS-Projekt.
2. Wählen Sie im Menü **PLC** oder in der Symbolleiste **TwinCAT SPS Symbolleistenoptionen** den Befehl **Einloggen**.
   ⇨ Ein Dialog erscheint mit dem Hinweis, dass die Applikation seit dem letzten Download geändert wurde.
3. Wählen Sie die Option **Mit Download einloggen** und klicken Sie auf **OK**.

⇨ Das laufende Programm auf der Steuerung wird angehalten und die Änderung auf die Steuerung geladen. Das SPS-Projekt ist eingeloggt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Einloggen [▶ 1001]

Download im eingeloggten Zustand (Onlinebetrieb)

✓ Die Verbindungseinstellungen der Steuerung sind korrekt eingestellt. Die Applikationen im Projekt und auf der Steuerung sind identisch. Das Programm auf der Steuerung läuft. Das SPS-Projekt ist eingeloggt.

1. Selektieren Sie ein Objekt im SPS-Projektbaum. Hier wählen Sie am besten eine POU oder GVL.
2. Wählen Sie im Kontextmenü den Befehl **Objekt (offline) bearbeiten**.
   ⇨ Das Objekt öffnet im Editor.
3. Verändern Sie das Objekt. Hier können Sie beispielsweise eine neue Variable deklarieren oder eine Wertzuweisung ändern.
4. Wählen Sie im Menü **PLC** den Befehl **Laden**.
   ⇨ Sie erhalten eine Abfrage, ob Sie die Operation **Laden** wirklich ausführen wollen.
5. Bestätigen Sie den Dialog mit **Ja**.

⇨ Das laufende Programm auf der Steuerung wird angehalten und die Änderung auf die Steuerung geladen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Objekt (offline) bearbeiten [▶ 926]
- Dokumentation TC3 User Interface: Befehl Laden [▶ 999]

12 Stand-alone SPS-Projekt verwenden

Ein stand-alone SPS-Projekt verwaltet ein herkömmliches SPS-Projekt (im TwinCAT-Projekt eingebettetes SPS-Projekt) in einem separaten Projekttyp. Die Programmierung erfolgt wie bei einem eingebetteten SPS-Projekt. Im Unterschied zu diesem steht die SPS-Instanz nicht direkt im Projekt zur Verfügung, sondern kann erst nach Einbindung innerhalb des TwinCAT-Projekts konfiguriert werden. Zur Verwaltung projektbasierter Datentypen oder Konfiguration des TwinCAT 3 Eventloggers steht im stand-alone SPS-Projekt zusätzlich das TwinCAT-3-Typsystem zur Verfügung.

Stand-alone SPS-Projekte ermöglichen es somit, die System-, Motion- und I/O-Konfiguration von der SPS-Entwicklung auf Projektebene zu trennen. So entstehen zwei separate Projekte: das stand-alone SPS-Projekt und das TwinCAT-Projekt.

Beide Projekte können sowohl in einer gemeinsamen Projektmappe als auch in separaten Projektmappen verwaltet werden. Um die SPS-Modulinstanz im TwinCAT-Projekt mit den zugehörigen Komponenten auf System-, Motion-, und I/O-Ebene verknüpfen zu können, wird die im stand-alone SPS-Projekt kompilierte TMC-Datei eingebunden.

Dieses Vorgehen ermöglicht z. B. die strikte Trennung von Systemkonfiguration und SPS-Programmierung, das Einbinden einer SPS-Instanz in unterschiedliche Systemkonfigurationen oder auch das Herunterladen der SPS-Projekt-Laufzeitdaten unter bestimmten Bedingungen ausnahmslos aus dem SPS-Projekt.

Voraussetzung

Stand-alone SPS-Projekte können Sie ab TwinCAT-Version 3.1.4022.0 verwenden.

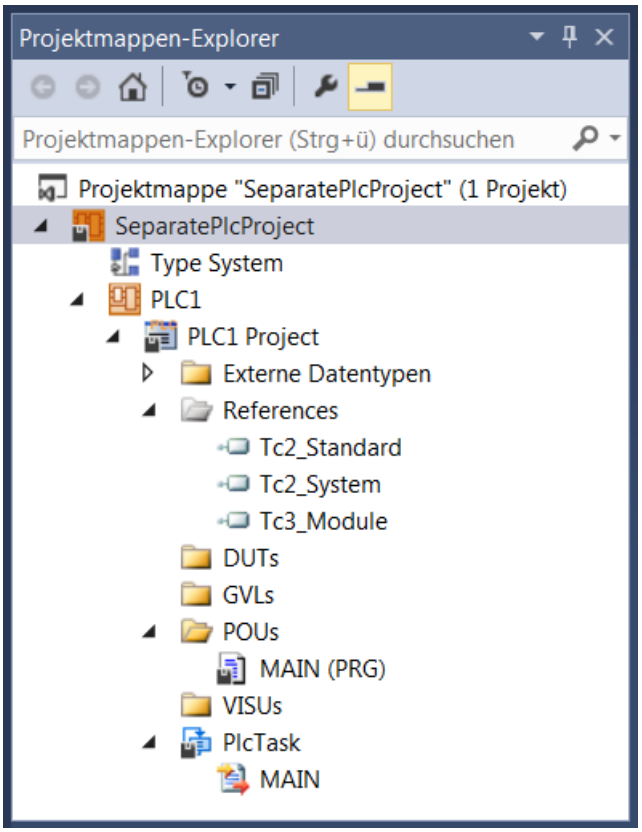
Siehe auch:

- Standardprojekt anlegen [▶ 55]
- Typsystem

12.1 Stand-alone SPS-Projekt anlegen

1. Wählen Sie im Menü **Datei** den Befehl **Neu > Projekt.**
   ⇨ Der Dialog **Neues Projekt** öffnet sich.
2. Wählen Sie die TwinCAT-SPS-Vorlage **TwinCAT SPS Projekt** aus.
3. Geben Sie dem SPS-Projekt einen Namen (z. B. „SeparatePlcProject“) und wählen Sie einen Speicherort und eine Projektmappe aus.
4. Bestätigen Sie den Dialog mit **Ok.**
   ⇨ Im **Projektmappen-Explorer** öffnet sich eine neue Projektmappe. In der Titelleiste des Hauptfensters erscheint der Projektname „SeparatePlcProject“.
5. Markieren Sie im Projektbaum das SPS-Objekt und wählen Sie im Menü **Projekt** oder im Kontextmenü den Befehl **Neues Element hinzufügen…** aus.
6. Wählen Sie in der Kategorie **Plc Templates** das **Standard PLC Projekt** aus und geben Sie einen Namen ein (z. B. „PLC1“).
7. Beenden Sie den Dialog mit **Hinzufügen**.

⇨ In der Ansicht **Projektmappen-Explorer** wird folgende Struktur angelegt:



⇨ Unterhalb des SPS-Projektobjekts (PLC1) erscheinen automatisch die Basisobjekte eines Standard-SPS-Projekts sowie das Typsystem. Im Gegensatz zum Standard-SPS-Projekt, das in ein TwinCAT-Projekt eingebettet ist, wird keine SPS-Instanz angelegt.

Siehe auch:

- Standardprojekt anlegen [▶ 55]

12.2 Stand-alone SPS-Projekt in ein TwinCAT-Projekt einbinden

Die Einbindung eines stand-alone SPS-Projekts in ein TwinCAT-Projekt erfolgt über eine TMC-Datei. Diese wird beim Erstellen des stand-alone SPS-Projekts erzeugt und dem TwinCAT-Projekt hinzugefügt.

12.2.1 TMC-Datei erzeugen und dem TwinCAT-Projekt hinzufügen

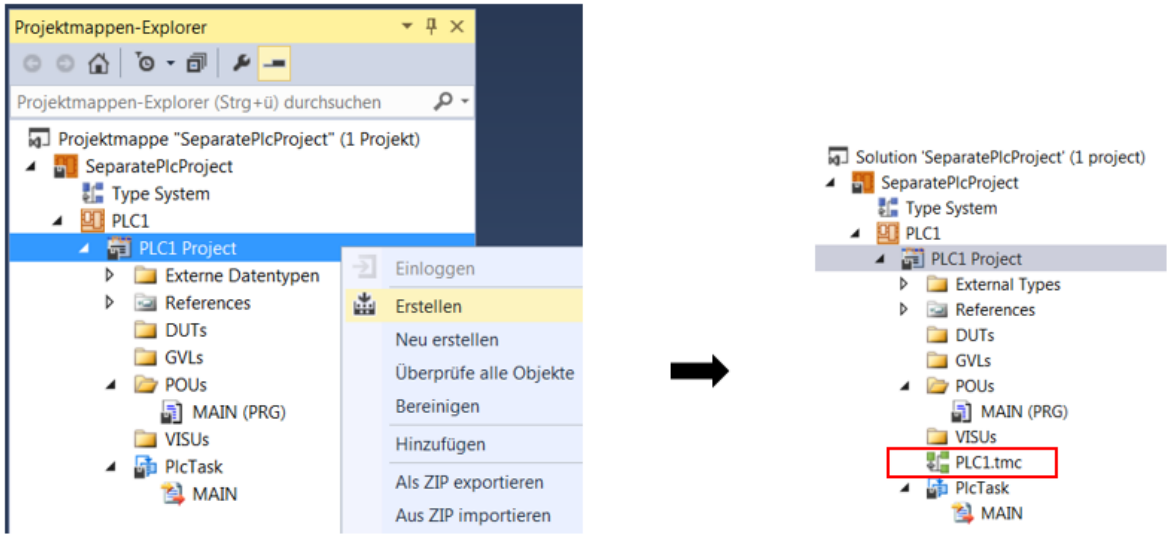
Nachfolgend wird beschrieben, wie Sie eine TMC-Datei erzeugen und diese dem TwinCAT-Projekt hinzufügen. Ein mehrfaches Hinzufügen der TMC-Datei in demselben TwinCAT-Projekt ist nicht erlaubt.

✓ Ein stand-alone SPS-Projekt ist angelegt (z. B. „SeparatePlcProject“), das ein SPS-Projekt enthält (z. B. „PLC1“).

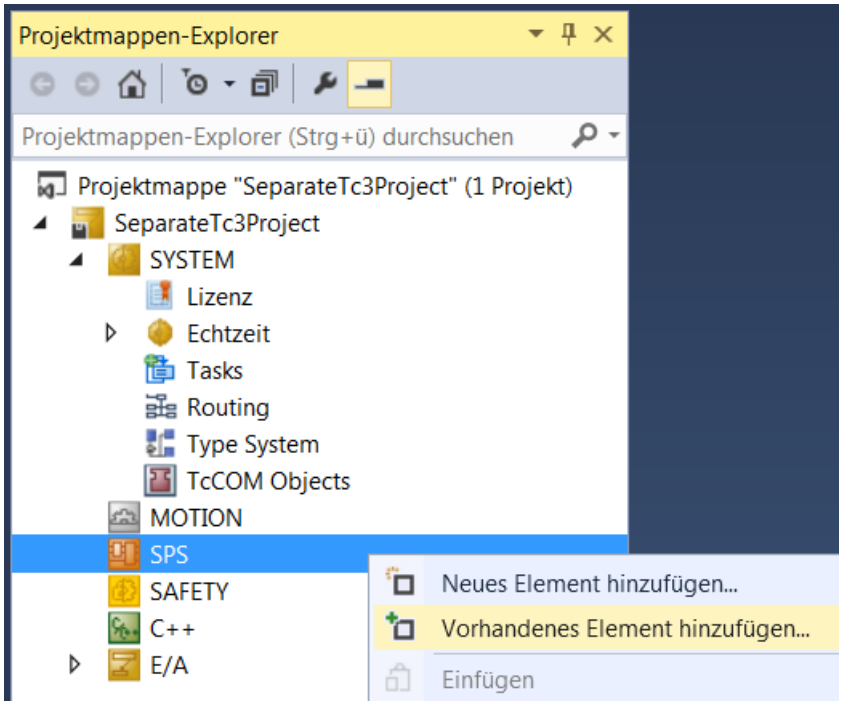
✓ Ein TwinCAT-Projekt ist angelegt, in das das stand-alone SPS-Projekt eingebunden werden soll (z. B. „SeparateTc3Project“).

1. Öffnen Sie das stand-alone SPS-Projekt und markieren Sie im SPS-Projektbaum das SPS-Projektobjekt („PLC1 Project“).
2. Wählen Sie im Kontextmenü oder im Menü **Erstellen** den Befehl **Erstellen** (**<SPS-Projektname> erstellen**).

⇨ Das SPS-Projekt wird übersetzt und dabei auf Fehler überprüft. Bei erfolgreicher Übersetzung wird die TMC-Datei erzeugt.

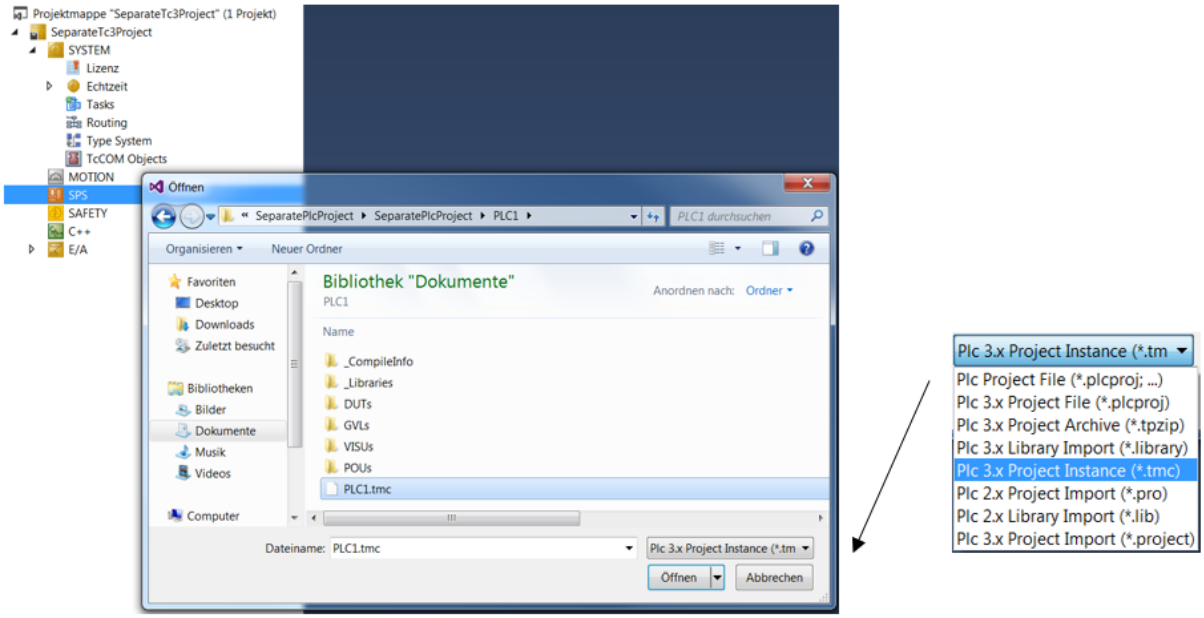


3. Öffnen Sie das TwinCAT-Projekt und markieren Sie im TwinCAT-Projektbaum das SPS-Objekt.
4. Wählen Sie im Kontextmenü oder im Menü **Projekt** den Befehl **Vorhandenes Element hinzufügen.**

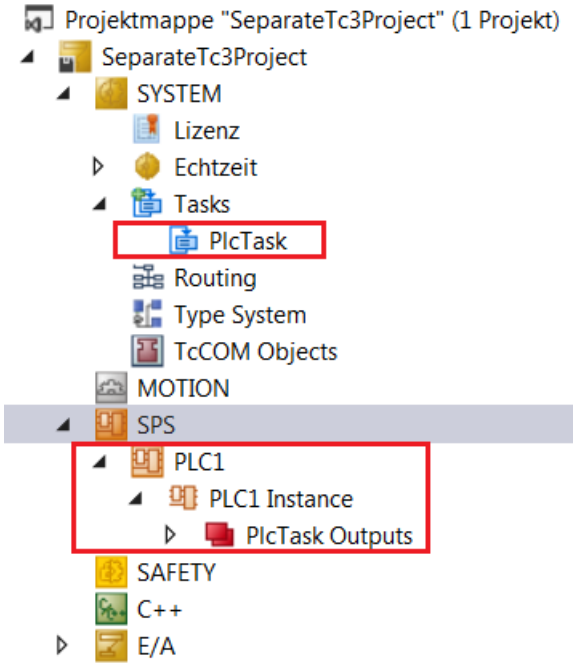


⇨ Der Standarddialog zum Öffnen einer Datei erscheint.

5. Wählen Sie die TMC-Datei aus und klicken Sie auf **Öffnen**, um sie dem Projekt hinzuzufügen.

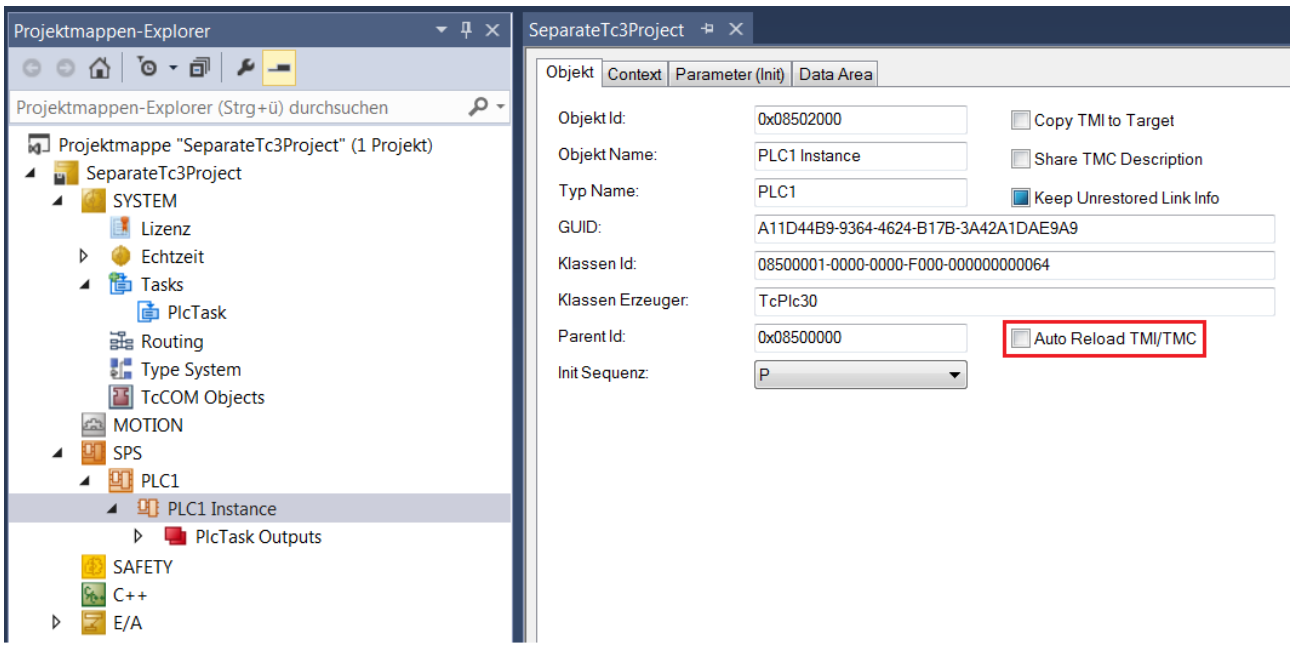


⇨ Die SPS-Instanz des stand-alone SPS-Projekts und die referenzierten Tasks werden dem TwinCAT-Projekt hinzugefügt. Wenn im TwinCAT-Projekt bereits eine Systemtask mit dem Namen der referenzierten Task des SPS-Projekts existiert, wird keine neue Task hinzugefügt, sondern die vorhandene Systemtask mit der hinzugefügten SPS-Instanz verknüpft.



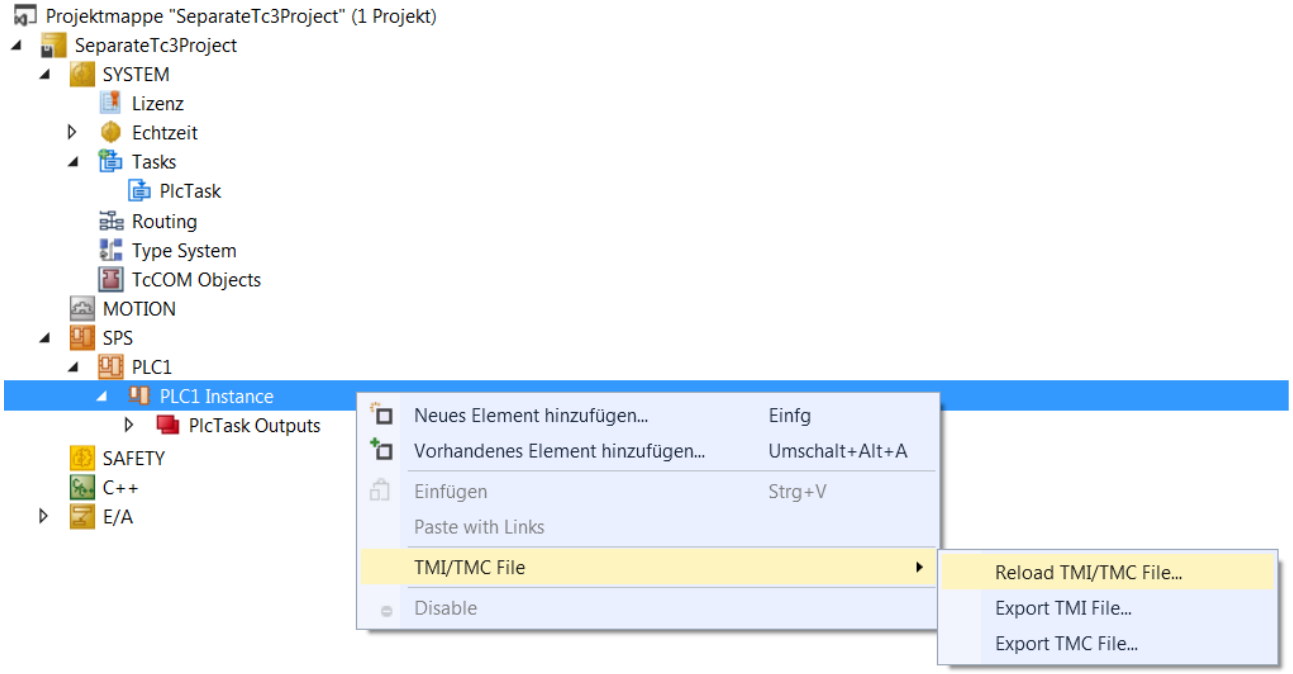
TMC-Datei automatisch neu laden

In den Einstellungen der SPS-Instanz können Sie festlegen, dass die TMC-Datei bei Änderung des stand-alone SPS-Projekts automatisch neu geladen wird. Klicken Sie dazu im Projektbaum doppelt auf die SPS-Instanz des stand-alone SPS-Projekts, um die SPS-Instanzeinstellungen im Editor zu öffnen. Aktivieren Sie in der Registerkarte **Objekt** das Auswahlkästchen **Auto Reload TMI/TMC**.



TMC-Datei manuell neu laden

Um die TMC-Datei im TwinCAT-Projekt bei Änderung des stand-alone SPS-Projekts manuell neu zu laden, markieren Sie im Projektbaum die eingebundene SPS-Instanz des stand-alone SPS-Projekts und wählen Sie im Kontextmenü den Befehl **TMI/TMC File > Reload TM/TMC File**.

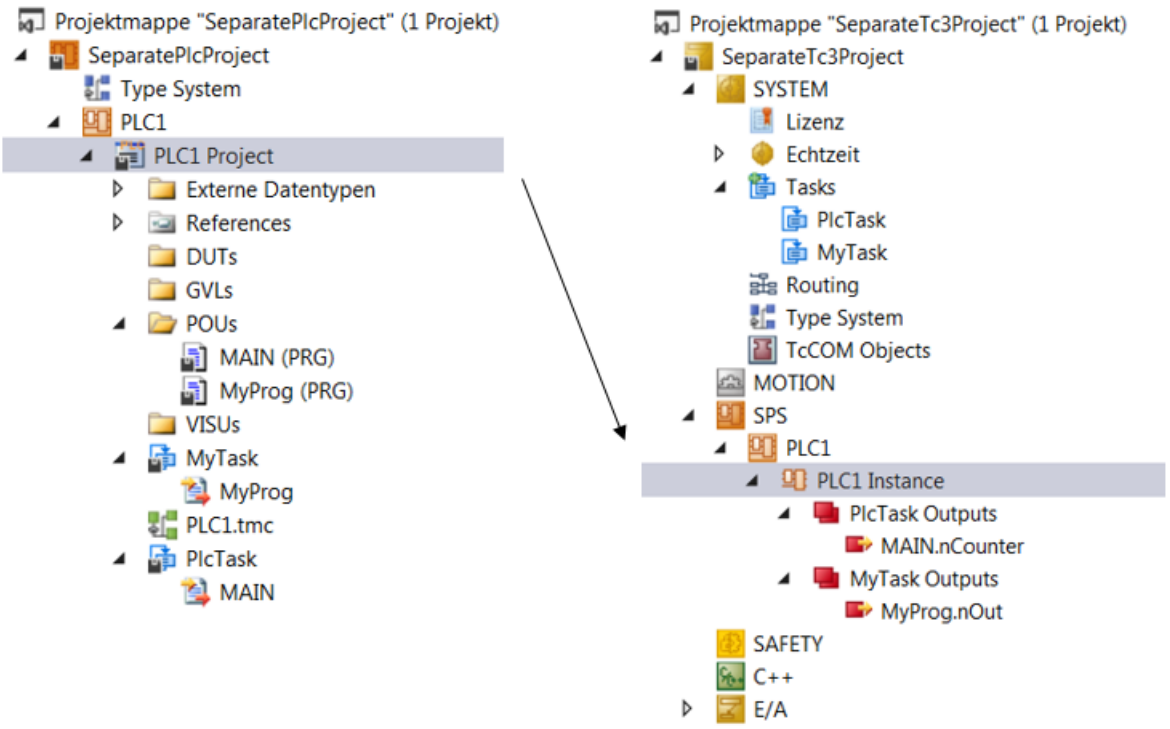


12.2.2 Task zuweisen

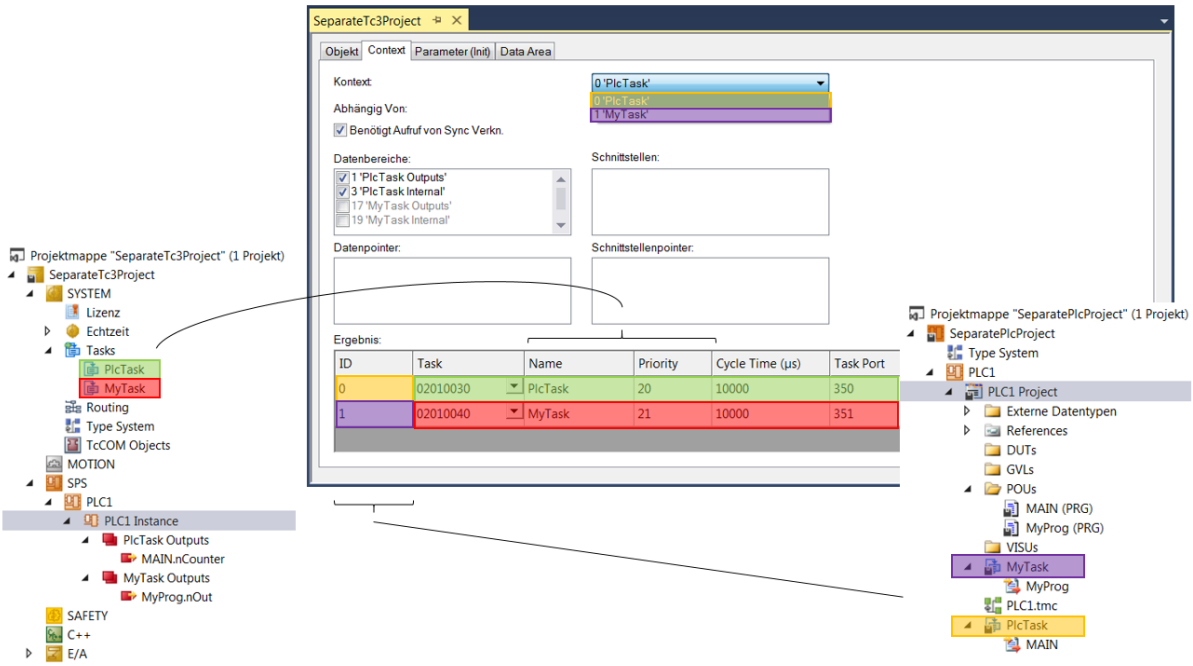
Nachfolgend wird beschrieben, wie Sie die für das stand-alone SPS-Projekt erzeugten Taskreferenzen den Systemtasks zuordnen.

✓ Für das stand-alone SPS-Projekt sind verschiedene Taskreferenzen erzeugt (z. B. „PlcTask“ und „MyTask“, welche die Abarbeitung der Programmbausteine MAIN und MyProg definieren).

✓ Die SPS-Instanz des stand-alone SPS-Projekts ist in das TwinCAT-Projekt eingebunden.



1. Klicken Sie im TwinCAT-Projektbaum doppelt auf die eingebundene SPS-Instanz, um die SPS-Instanzeinstellungen im Editor zu öffnen.
2. Wählen Sie die Registerkarte **Context**.
3. Ordnen Sie die Taskreferenzen den vorhandenen Systemtasks zu. (In diesem Beispiel sind die Taskreferenzen den gleichnamigen Systemtasks zugeordnet.)

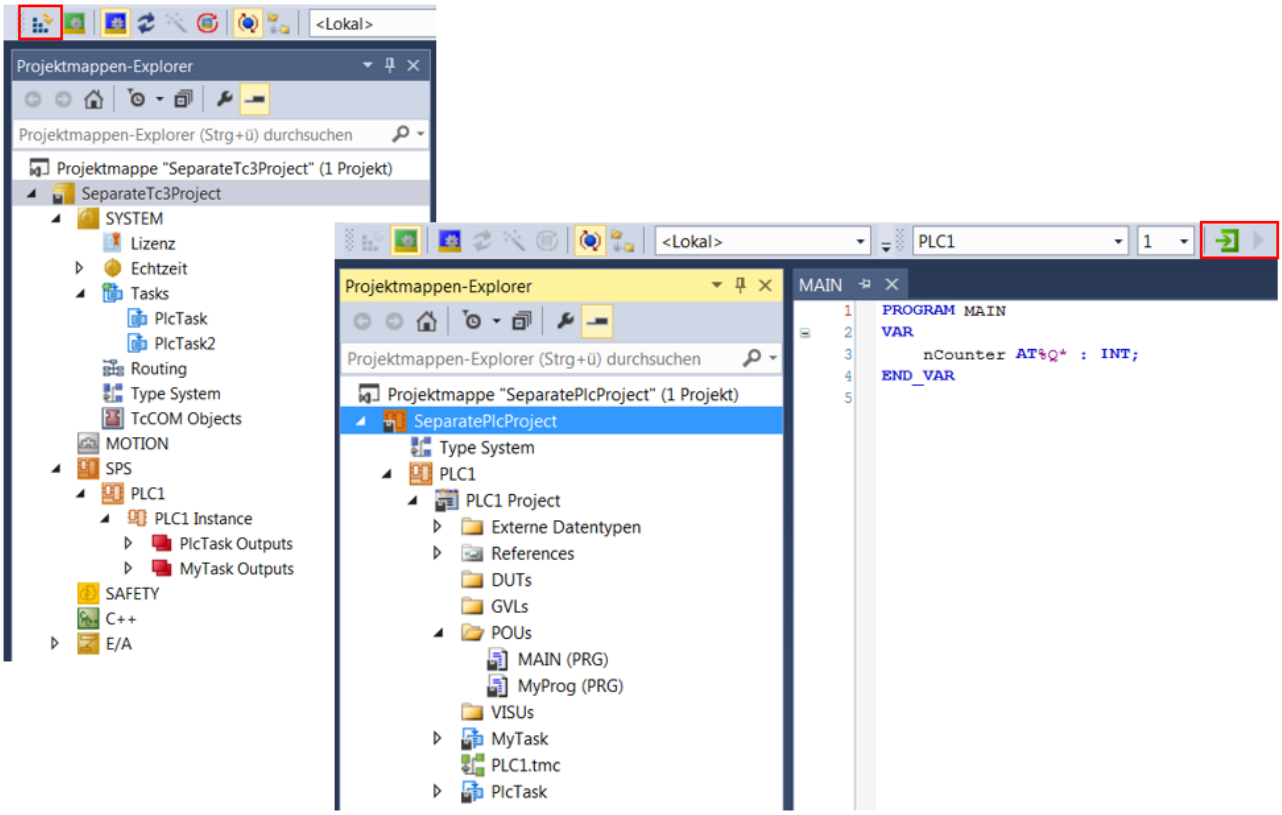


12.2.3 Programmcode laden, einloggen und SPS starten

✓ Ein stand-alone SPS-Projekt ist in ein TwinCAT-Projekt eingebunden.

1. Aktivieren Sie die Konfiguration des TwinCAT-Projekts. Wählen Sie dazu in der Symbolleiste den Befehl **Activate Configuration**.

2. Loggen Sie sich im stand-alone SPS-Projekt auf der SPS ein und starten Sie die SPS.



12.3 Best Practice

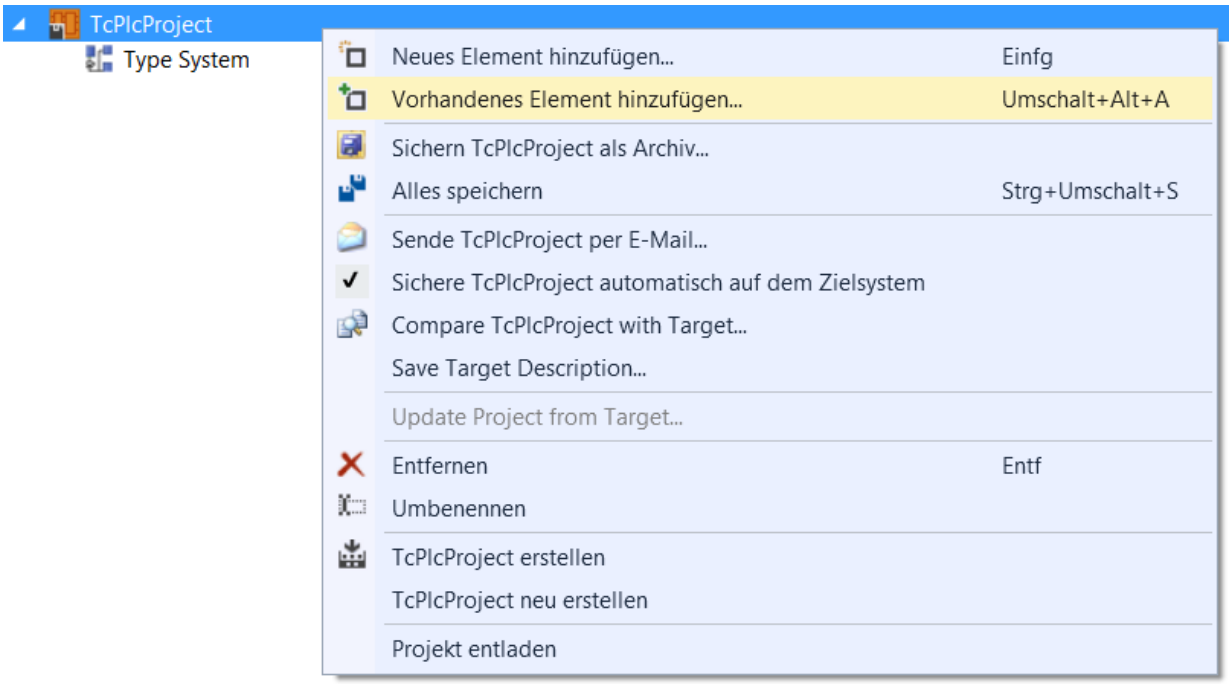
Übersicht

- Das eingebettete SPS-Projekt eines existierenden TwinCAT-Projekts in ein stand-alone SPS-Projekt umwandeln [▶ 273]
- Ein stand-alone SPS-Projekt in ein eingebettetes SPS-Projekt umwandeln [▶ 276]

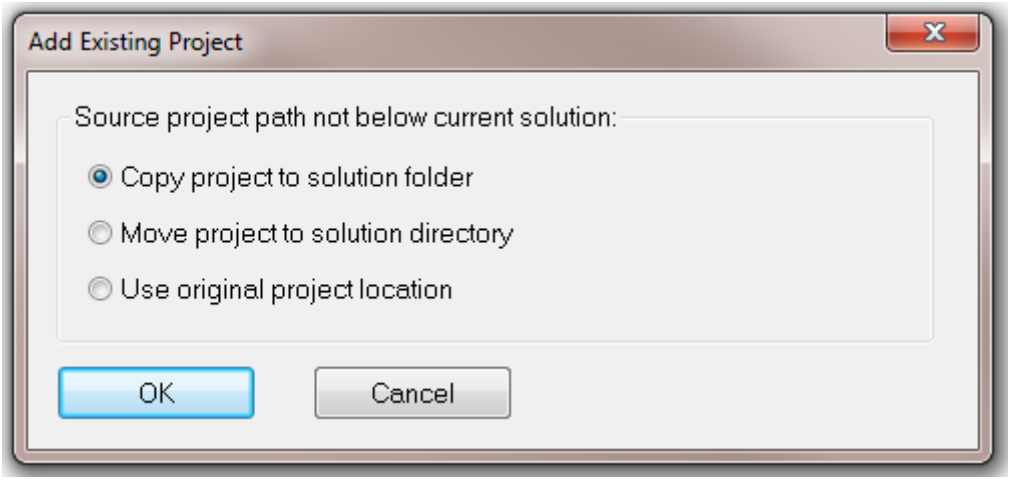
Das eingebettete SPS-Projekt eines existierenden TwinCAT-Projekts in ein stand-alone SPS-Projekt umwandeln

1. Legen Sie ein Projekt vom Typ „stand-alone SPS-Projekt“ an. Legen Sie dabei kein SPS-Projekt an.

2. Fügen Sie dem stand-alone SPS-Projekt das eingebettete SPS-Projekt des existierenden TwinCAT-Projekts hinzu. Klicken Sie dazu im Kontextmenü des SPS-Projekts auf **Vorhandenes Element hinzufügen…** und wählen Sie in dem sich öffnenden Standarddialog die .plcproj-Datei des eingebetteten SPS-Projekts aus. Bestätigen Sie den Dialog.

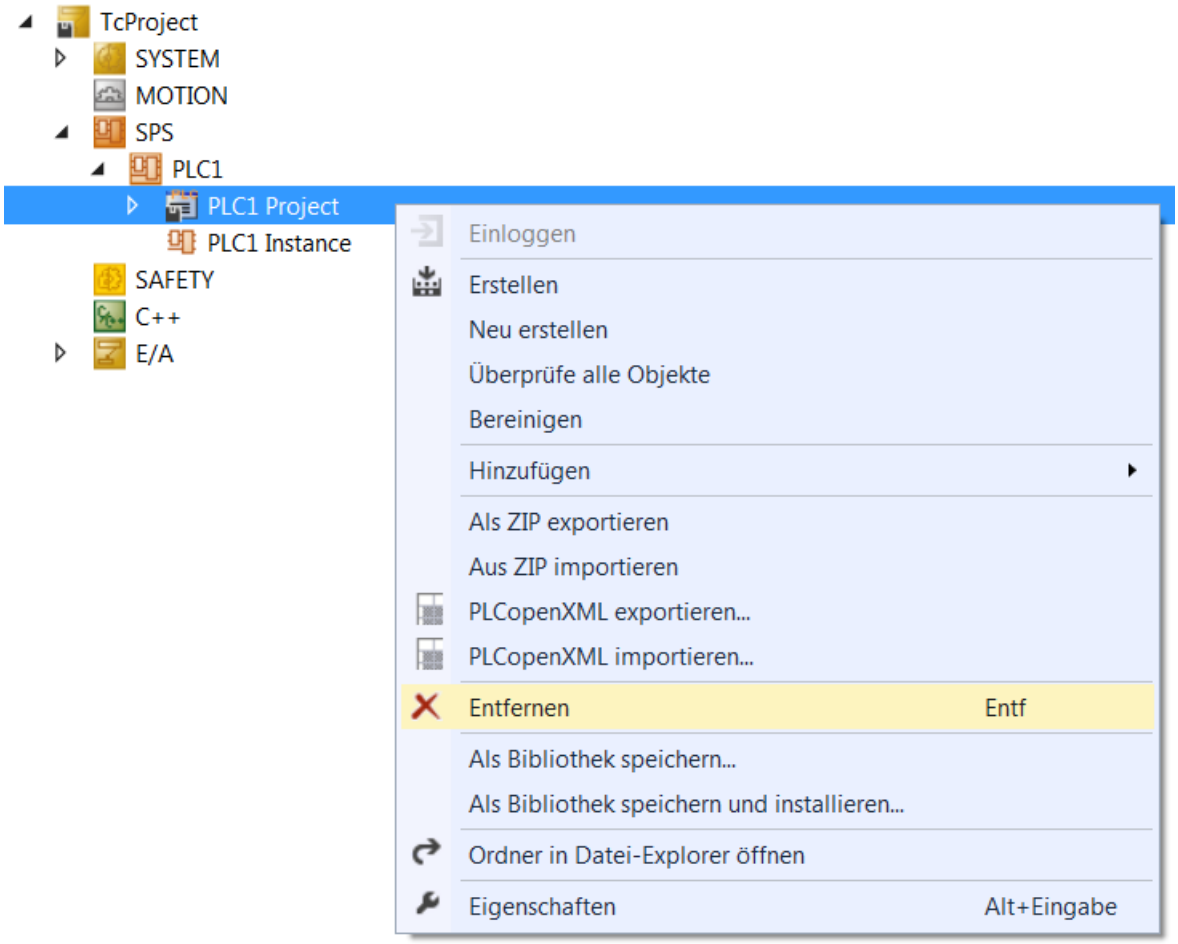


⇨ Der Dialog **Add Existing Project** öffnet sich.

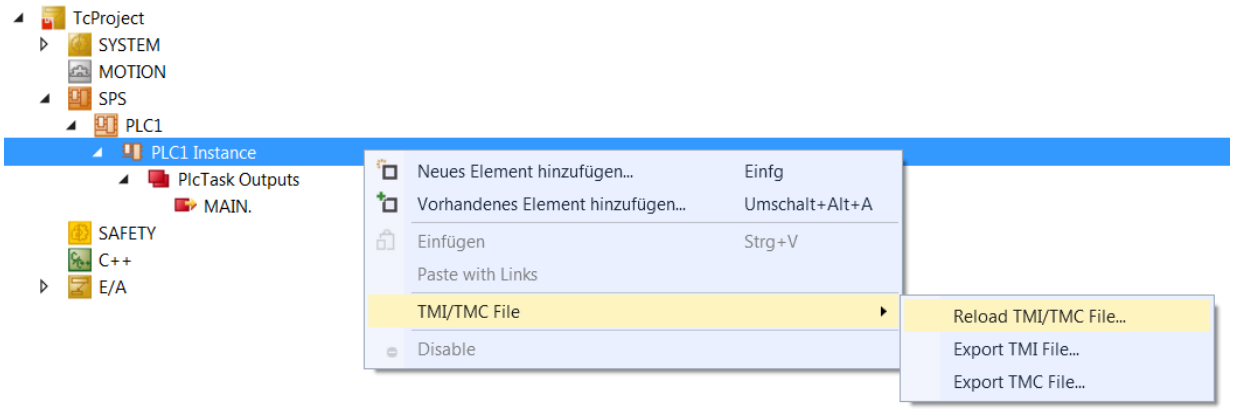


3. Wählen Sie die Option **Copy project to solution folder,** um das SPS-Projekt unabhängig vom ursprünglichen SPS-Projekt verwalten zu können.
   ⇨ Das SPS-Projekt wird dem stand-alone SPS-Projekt hinzugefügt.
4. Erstellen Sie das SPS-Projekt, um eine TMC-Datei zu erzeugen.

5. Löschen Sie das eingebettete SPS-Projekt im existierenden TwinCAT-Projekt, indem Sie nur das eigentliche SPS-Projekt, aber nicht die zugehörige Instanz löschen, sodass die vorhandenen Mappings erhalten bleiben.



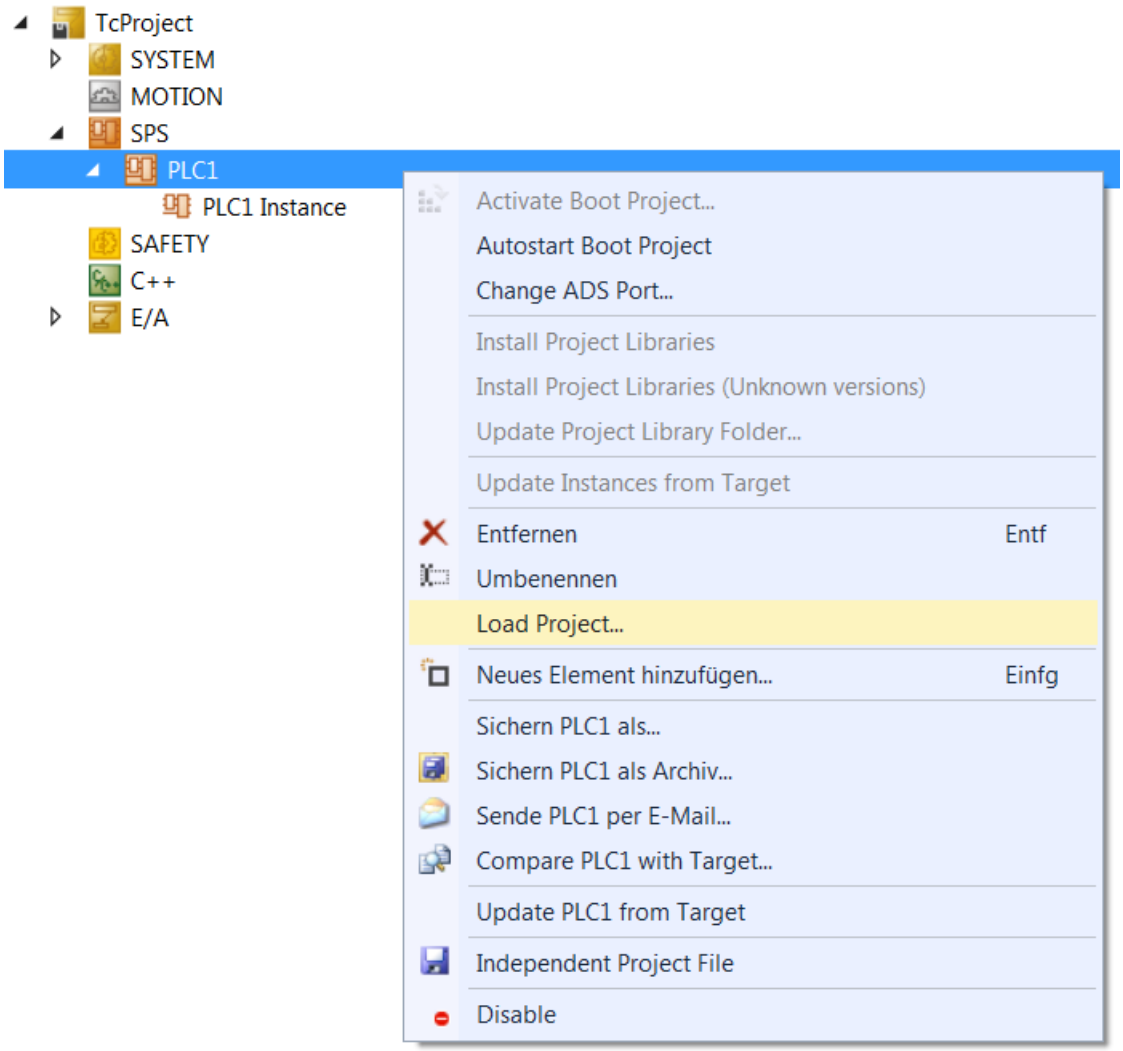
6. Verändern Sie den Pfad der TMC-Datei. Klicken Sie dazu im Kontextmenü der SPS-Projektinstanz auf **TMI/TMC File > Reload TMI/TMC File…** und wählen Sie in dem sich öffnenden Standarddialog die TMC-Datei des stand-alone SPS-Projekts aus. Bestätigen Sie den Dialog.



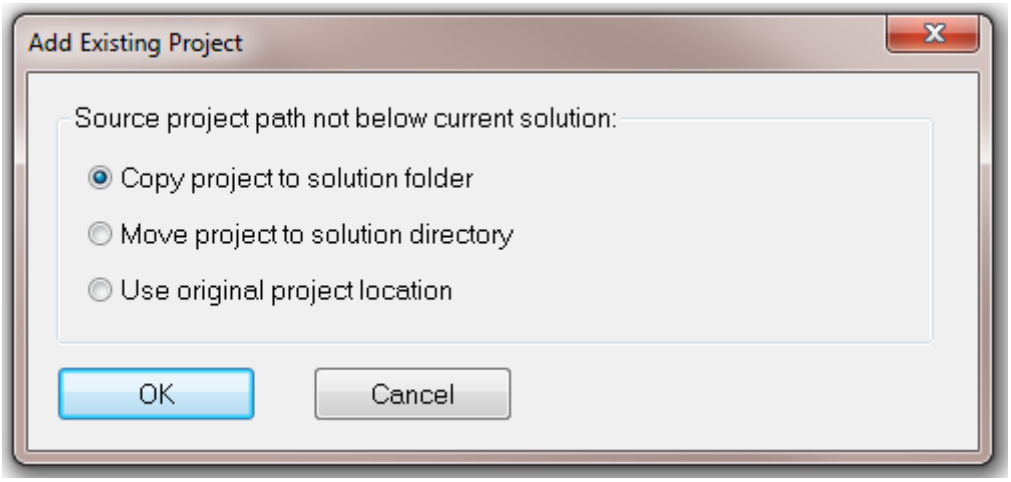
⇨ Das zuvor in das TwinCAT-Projekt eingebettete SPS-Projekt bildet nun ein eigenständiges SPS-Projekt (stand-alone SPS-Projekt).

Ein stand-alone SPS-Projekt in ein eingebettetes SPS-Projekt umwandeln

1. Fügen Sie das SPS-Projekt des stand-alone SPS-Projekts dem existierenden SPS-Projekt hinzu. Klicken Sie dazu im Kontextmenü des SPS-Projekts auf **Load Project…** und wählen Sie in dem sich öffnenden Standarddialog die .plcproj-Datei des eigenständigen SPS-Projekts aus. Bestätigen Sie den Dialog.



⇨ Der Dialog **Add Existing Project** öffnet sich.



- **Copy project to solution folder**, wenn Sie das SPS-Projekt unabhängig vom ursprünglichen stand-alone SPS-Projekt verwalten und weiterhin das stand-alone SPS-Projekt nutzen möchten.
- **Move project to solution directory**, wenn Sie das stand-alone SPS-Projekt nicht mehr nutzen möchten.

- **Use original project location,** wenn Sie mit TwinCAT-Projekt und stand-alone SPS-Projekt auf denselben Daten arbeiten möchten.

2. Wählen Sie die gewünschte Option aus und bestätigen Sie den Dialog.

⇨ Das stand-alone SPS-Projekt ist nun in das TwinCAT-Projekt eingebettet.

12.4 FAQ

Übersicht

- Warum ist das Aktivieren der Konfiguration im TwinCAT-Projekt nicht möglich? [▶ 277]
- Warum ist das Einloggen der SPS im stand-alone SPS-Projekt nicht möglich? (a) [▶ 277]
- Warum ist das Einloggen der SPS im stand-alone SPS-Projekt nicht möglich? (b) [▶ 277]

Warum ist das Aktivieren der Konfiguration im TwinCAT-Projekt nicht möglich?

Fehlermeldung: „Cannot copy file target – source file '…' not found"

Ursache: Dieser Fehler kann unterschiedliche Ursachen haben:

1. Der ADS-Port des stand-alone SPS-Projekts und der SPS-Instanz im TwinCAT-Projekt sind nicht identisch.
   Lösung: Stellen Sie auf beiden Ebenen denselben ADS-Port ein, kompilieren Sie das stand-alone SPS-Projekt neu. Wenn die Option **Auto Reload TMI/TMC** im SPS-Instanz-Konfigurator nicht aktiviert ist, laden Sie die TMC-Datei neu.
2. Das TwinCAT-Projekt und das stand-alone SPS-Projekt wurden nicht für dieselbe Plattform kompiliert.
   Lösung: Kompilieren Sie beide Projekte für dieselbe Plattform. Wenn die Option **Auto Reload TMI/TMC** im SPS-Instanz-Konfigurator nicht aktiviert ist, laden Sie die TMC-Datei neu.
3. Es wurde nur die TMC-Datei für das TwinCAT-Projekt bereitgestellt, sodass der Pfad der TMC-Datei nicht zum stand-alone SPS-Projekt führt und die SPS-Laufzeitdaten nicht erreichbar sind.
   Lösung: Deaktivieren Sie in den SPS-Einstellungen des TwinCAT-Projekt in der Registerkarte **Settings** die Option **Activate PLC configuration (copy boot files)**.
   Bei dieser Arbeitsweise können die SPS-Laufzeitdaten ausnahmslos aus dem stand-alone SPS-Projekt auf das Zielsystem übertragen werden.

Warum ist das Einloggen der SPS im stand-alone SPS-Projekt nicht möglich? (a)

Fehlermeldung: „PLC instance parameter (0x0850801a) missmatch. Download will be aborted."

Ursache: Wenn dieser Fehler auftritt, hat vor dem Einloggen keine Aktualisierung der SPS-Modulinstanz-Informationen stattgefunden.

Lösung: Bestätigen Sie den Dialog der Fehlermeldung mit **Ja** und ignorieren Sie die darauffolgenden Fehlermeldungen oder führen Sie manuell eine Aktualisierung über **Update Instances from Target** durch.

Warum ist das Einloggen der SPS im stand-alone SPS-Projekt nicht möglich? (b)

Fehlermeldung: „PLC instance parameter (0x08500005) missmatch. Download will be aborted."

Ursache: Wenn dieser Fehler auftritt, entspricht das SPS-Projekt nicht der SPS-Instanz des aktivierten TwinCAT-Projekts. D. h. die TMC-Datei und somit die SPS-Instanz wurde im TwinCAT-Projekt nach Änderungen im stand-alone SPS-Projekt nicht aktualisiert.

Lösung: Bestätigen Sie den Dialog der Fehlermeldung mit **Ja** oder **Nein** und ignorieren Sie die darauffolgenden Fehlermeldungen. Aktualisieren Sie anschließend die TMC-Datei im TwinCAT-Projekt (**Kontextmenü SPS-Projektinstanz > TMI/TMC File > Reload TMI/TMC File…**). Wenn die TMC-Datei nicht separat bereitgestellt, sondern direkt über den Pfad des stand-alone SPS-Projekts hinzugefügt wird, sollte die Option **Auto Reload TMI/TMC** im SPS-Instanz-Konfigurator aktiviert werden.

13 Bibliotheken verwenden

Bibliotheken sind Sammlungen wiederverwendbarer Objekte wie:

- POUs wie Funktionsbausteine oder Funktionen
- Interfaces und ihre Methoden und Properties
- Datentypen wie Enumerationen, Strukturen, Aliases, Unions
- Globale Variablen, Konstanten, Parameterlisten
- Textlisten, Bildersammlungen, Visualisierungen, Visualisierungselemente
- Externe Dateien (z. B. Dokumentationen)

Durch das Einbinden einer Bibliothek in ein Projekt können die durch die Bibliothek bereitgestellten Module genauso im Projekt verwendet werden wie die anderen Bausteine und Variablen, die direkt im Projekt definiert sind.

Empfehlungen und Hinweise

Beachten Sie in Ergänzung zu den Beschreibungen der Bibliotheksverwendung die Anwendungshinweise im Abschnitt Empfehlungen und Hinweise [▶ 280].

Für die Verwendung von Bibliotheken sind die nachfolgend aufgeführten Schritte relevant: Bibliothekserstellung, Bibliotheksinstallation und Bibliotheksverwaltung.

Falls eine Bibliothek bereits erstellt und installiert wurde (ist z. B. für System-Bibliotheken der Fall), wird nur der Schritt der Bibliotheksverwaltung bzw. der Einbindung von Bibliotheken benötigt. Eigene Bibliotheken müssen hingegen zuvor erstellt und installiert werden.

Bibliothekserstellung

- Bei der Erstellung einer Bibliothek kann zwischen zwei Bibliothekstypen gewählt werden: *.library (Source-Bibliothek) und *.compiled-library (übersetzte Bibliothek mit Quellcodeschutz).
- Voraussetzungen und weitere Informationen finden Sie im Abschnitt Bibliothekserstellung [▶ 282].

Bibliotheksinstallation

- Voraussetzung für die Verwendung einer Bibliothek in einem Projekt ist die vorherige Installation der Bibliothek auf dem System. Bibliotheken werden auf dem lokalen System in verschiedenen „Repositories“ (Verzeichnisse, Speicherorte) verwaltet. Bevor eine Bibliothek in ein Projekt eingebunden werden kann, muss sie mit einer definierten Versionsnummer auf dem lokalen System in ein solches Repository installiert werden.
  - **Ausnahme**: Projekte, die als Bibliothek referenziert werden.
    Siehe SPS-Projekt als referenzierte Bibliothek verwenden.
- Ist eine verwendete Bibliotheksversion nicht im Repository installiert, wird dies über ein Hinweissymbol im Projektbaum an der Referenz verdeutlicht.
- Die Installation von Bibliotheken erfolgt im Bibliotheksrepository [▶ 285].

Bibliotheksverwaltung

- Der Bibliotheksverwalter [▶ 289] bietet einen guten Überblick über die im Projekt verwendeten SPS-Bibliotheksreferenzen und kann genutzt werden, um Bibliotheken bzw. Platzhalter in ein Projekt einzubinden. Durch die Einbindung können die durch die Bibliotheksreferenz bereitgestellten Elemente im Projekt verwendet werden.
- Bibliotheksreferenzen, die als Unterbibliotheken in einer anderen Bibliothek referenziert sind, werden im Bibliotheksverwalter ebenfalls dargestellt. Es gibt allerdings auch „versteckte Bibliotheken“ (siehe Abschnitt Befehl Eigenschaften [▶ 377]).
- Bei Verwendung einer Bibliothek wird immer eine eindeutige Version der Bibliothek referenziert. Welche dies ist, wird als effektive Version angegeben. Falls die Bibliothek mit einer festen Version angefügt wurde (z. B. 3.3.0.0), wird das Projekt immer diese Bibliotheksversion nutzen, auch wenn von dieser Bibliothek bereits neuere Versionen existieren sollten. Das Projekt kann über die Einstellung „immer neueste“/„*“ aber auch automatisch immer die neueste Version einer Bibliothek nutzen. In

diesem Fall verwendet TwinCAT immer die neueste im Bibliotheksrepository gefundene Version der Bibliothek. Weitere Informationen sowie ein Beispiel finden Sie unter Befehl Immer neueste Version verwenden [▶ 381].

- Es ist nicht möglich, die gleiche Version derselben Bibliothek mehr als einmal zu einem Bibliotheksverwalter hinzufügen. Eine Version einer Bibliothek kann in einem Bibliotheksverwalter entweder als Bibliothek oder als Platzhalter referenziert werden.
- Liegt die Bibliothek nicht in übersetzter Form (*.compiled-library) vor, sondern ist die *.library-Datei vorhanden, so können die im Bibliotheksverwalter aufgeführten Elemente der Bibliothek durch einen Doppelklick auf den jeweiligen Eintrag geöffnet werden.
- Sie können Bibliotheksreferenzen in Form einer Bibliothek oder in Form eines Platzhalters zum Bibliotheksverwalter hinzufügen und so in Ihre Anwendung einbinden (siehe Abschnitt Befehl Bibliothek hinzufügen [▶ 373]). Es sollten nach Möglichkeit Platzhalter verwendet werden. Weitere Informationen finden Sie im Abschnitt Bibliotheksplatzhalter [▶ 293].
- Wenn ein Bibliotheksmodul im Projekt angesprochen wird, werden die Bibliotheken und Repositories in der Reihenfolge durchsucht, in der sie im Bibliotheksrepository [▶ 285] aufgeführt sind. Weitere Informationen finden Sie im Abschnitt Eindeutiger Zugriff auf Bibliotheksmodule oder -variablen [▶ 280].

Bibliotheksdokumentation

TwinCAT stellt Ihnen vielfältige Möglichkeiten der Bibliotheksdokumentation zur Verfügung. Informationen hierzu finden Sie im Abschnitt Bibliotheksdokumentation [▶ 296].

Sehen Sie im Folgenden außerdem Informationen zu diesen Themen:

- Referenzierte Bibliotheken
- Bibliotheksversionen
- Eindeutiger Zugriff auf Bibliotheksmodule und -variablen
- TwinCAT 2.x PLC Control Bibliotheken
- Externe und interne Bibliotheken bzw. Bibliotheksmodule, spätes Binden

Referenzierte Bibliotheken

- Eine Bibliothek kann andere Bibliotheken einbinden (referenzierte Bibliotheken), wobei die Verschachtelung beliebig tief sein kann. Wenn eine solche „Vater“-Bibliothek dann selbst in ein Projekt eingebunden wird, stehen die in ihr referenzierten Bibliotheken dort ebenfalls zur Verfügung.
- Bibliotheksreferenzen sollten unbedingt über Bibliotheksplatzhalter [▶ 293] definiert werden, um Probleme zu vermeiden, die durch Versionsabhängigkeiten und die Notwendigkeit, herstellerspezifische Bibliotheken zu verwenden, entstehen können.
- In den Eigenschaften [▶ 377] jeder referenzierten Bibliothek, können Sie festlegen, wie sie sich später, wenn sie über die „Vater“-Bibliothek in ein Projekt eingebunden wird, verhalten soll; beispielsweise ob sie im Bibliotheksverwalter „verborgen“ werden soll.

Bibliotheksversionen

- Mehrere Versionen einer Bibliothek können gleichzeitig auf dem System installiert sein.
- Mehrere Versionen einer Bibliothek können gleichzeitig im Projekt eingebunden sein. Es ist nicht ratsam, das zu tun. In diesem Fall muss unbedingt jeder der Bibliotheken ein eindeutiger Namensraum zugeordnet sein und der Zugriff auf die Symbole muss qualifiziert erfolgen. Beispiele: V1.Send, V2.Send
- Die Version bzw. Auflösung von Bibliotheken bzw. Platzhaltern kann im Eigenschaftenfenster [▶ 377] konfiguriert werden. Die Auflösung von Platzhaltern kann außerdem im Platzhalter-Dialog angepasst werden.

- Insbesondere wenn andere Bibliotheken in einer Bibliothek referenziert werden, aber auch um ein Projekt kompatibel zu machen, wird dringend empfohlen, Platzhalter zu verwenden. Damit können Probleme aufgrund von Versionsabhängigkeiten oder der Notwendigkeit, herstellerspezifische Bibliotheken zu verwenden, vermieden werden.

Eindeutiger Zugriff auf Bibliotheksmodule oder -variablen

- Wenn im Projekt und in Bibliotheken mehrere Module oder Variablen gleichen Namens verfügbar sind, sollte der Zugriff auf eine Modulkomponente eindeutig sein. Bezogen auf Bibliotheken wird die Eindeutigkeit durch Hinzufügen des Namensraums der Bibliothek vor dem Modulnamen erreicht.
- Die Standardeinstellung für den **Namensraum** einer Bibliothek entspricht dem Bibliothekstitel. Für eine Bibliothek kann hingegen auch explizit ein davon abweichender Namensraum definiert werden: entweder allgemein für die Bibliothek bei der Bibliothekserstellung [▶ 282] in den Projekteigenschaften [▶ 948] oder für den lokalen Gebrauch der Bibliothek in einem Projekt im Eigenschaftenfenster [▶ 377] der Bibliotheksreferenz.
  Der Namensraum der Bibliothek muss als Präfix des Bezeichners verwendet werden, damit ein eindeutiger Zugriff auf ein Modul möglich ist, das mehrfach im Projekt vorhanden ist.
- Beispiel:
  - Die Bibliothek Lib1 wird in einem Applikationsprojekt eingebunden.
  - Die Funktion F_Sample ist sowohl in der Bibliothek Lib1 als auch in der Applikation deklariert.
  - Um einen eindeutigen Zugriff auf die beiden Funktionen zu implementieren, wird der Namensraum der Bibliothek vor dem Aufruf der Bibliotheksfunktion hinzugefügt. Dadurch wird eindeutig die Funktion der Bibliothek Lib1 angesprochen.
  - Aufruf der Applikationsfunktion `nResult := F_Sample(nInput := nVar);`
  - Aufruf der Bibliotheksfunktion `nResult := Lib1.F_Sample(nInput := nVar);`

TwinCAT 2.x PLC Control Bibliotheken

- Bibliotheken, die mit TwinCAT 2.x PLC Control (*.lib) erstellt wurden, werden weiterhin unterstützt.
- Ein „altes“ Bibliotheksprojekt (*.lib) kann in TwinCAT 3 PLC geöffnet und in eine „neue Bibliothek“ (*.library/*.compiled-library) konvertiert werden.
- Wenn ein altes Projekt geöffnet wird, das alte Bibliotheken referenziert, kann gewählt werden, ob diese Referenzen beibehalten, durch andere ersetzt oder gelöscht werden sollen. Sollen sie beibehalten werden, werden die betreffenden Bibliotheken in das neue Format konvertiert und dabei automatisch in das „System“-Bibliotheksrepository installiert. Falls sie die nötigen Projektinformationen nicht enthalten, können diese unmittelbar nachgetragen werden. Die Vorgehensweise, gemäß derer eine bestimmte alte Bibliothek bei der Konvertierung eines alten Projekts gehandhabt wurde, kann in den Projektoptionen gespeichert werden. Somit muss, sollte die gleiche Bibliothek erneut bei der Konvertierung eines alten Projekts in Erscheinung treten, die Vorgehensweise nicht wiederholt definiert werden, sondern wird automatisch ausgeführt.
- Im Abschnitt Neues Element hinzufügen [▶ 937] wird die Vorgehensweise beim Öffnen und Konvertieren von Projekten und Bibliotheken beschrieben.

Externe und interne Bibliotheksmodule, spätes Binden

- Bei externen Bibliotheksmodulen handelt es sich um Firmware-Funktionen, deren Implementierung sich nicht in der SPS-Bibliothek befindet. Für Bibliotheken mit Firmware-Funktionen gilt, dass die Firmware auf dem Zielsystem vorliegen muss und erst gebunden wird, wenn die Applikation dort läuft.

13.1 Empfehlungen und Hinweise

Im Folgenden sind einige Empfehlungen und Hinweise im Zusammenhang mit der Bibliotheksverwendung beschrieben.

Empfehlung von Bibliotheksplatzhaltern

Durch die Verwendung von Bibliotheksplatzhaltern ist die Anpassung von verwendeten Bibliotheksversionen mit äußerst geringem Aufwand verbunden, sodass sich der Engineering-Prozess von Projekten und Bibliotheken sehr flexibel gestaltet. Daher wird die Verwendung von Platzhaltern gegenüber der Verwendung von Bibliotheken empfohlen.

Weitere Informationen finden Sie unter Bibliotheksplatzhalter [▶ 293].

Möglicher Online-Change durch Verwendung von „immer neuesten“ Versionen

Beispiel: Sie aktivieren und starten ein Projekt, in welchem die Bibliothek „LibA“ als „immer neueste“ Version referenziert ist. Zum Zeitpunkt des Downloads ist Version 1.0.0.0 die neueste Version dieser Bibliothek im Bibliotheksrepository. Wenn Sie anschließend eine neuere Version von „LibA“ ins Bibliotheksrepository installieren (z. B. Version 1.0.1.0), stellt diese Version nun die neueste Version der Bibliothek dar. Wenn Sie sich mit dem Applikationsprojekt, welches Sie nicht aktiv geändert haben, einloggen, erkennt TwinCAT eine Änderung der Applikation, da „LibA“, die als „immer neueste“ verwendet wird, nicht mehr in der Version 1.0.0.0, sondern jetzt in der Version 1.0.1.0 referenziert wird. Es wird Ihnen beim Einloggen also korrekterweise ein Online-Change oder Download angeboten, obwohl Ihnen möglicherweise keine Projektänderung bewusst ist.

Diese Irritation durch automatische Änderung der effektiven Version einer Bibliothek (verursacht durch die Einstellung „immer neueste Version“) ist zum einen unnötig und sollte verhindert werden. Zum anderen sollten die Projektsourcen und die verwendeten Komponenten nach der Inbetriebnahme bzw. nach Projektierungsabschluss für Nachverfolgungszwecke eingefroren und gesichert werden.

Daher wird dringend empfohlen, alle verwendeten Bibliotheksreferenzen (sowohl die Bibliotheksreferenzen direkt auf Projektebene als auch die intern in Bibliotheken verwendeten Bibliotheksreferenzen) nach Projektierungsabschluss (d. h. vor Auslieferung) auf eine feste effektive Version zu stellen.

Beachten Sie, dass dieser Hinweis nicht nur für Anwenderbibliotheken, sondern auch für Beckhoff-Bibliotheken gilt, da mit der Installation eines neuen XAE- oder RM-Setups möglicherweise neue Versionen von referenzierten Beckhoff-Bibliotheken installiert werden.

Während der Entwicklungsphase ist es hingegen durchaus sinnvoll, die Option „immer neueste Version“ zu verwenden, da dann immer automatisch mit den neuesten verfügbaren Bibliotheksversionen gearbeitet wird.

Weitere Informationen zum Festlegen von Bibliotheksversionen auf eine feste effektive Version finden Sie im Abschnitt Befehl Effektive Version verwenden [▶ 380].

Weitere Informationen sowie ein weiteres Beispiel zu dem Verhalten von „immer neuesten“ Versionen finden Sie im Abschnitt Befehl Immer neueste Version verwenden [▶ 381].

Weitergabe des Quellcodes von Anwenderbibliotheken

Wenn Sie bei den Einstellungen für das Target- oder das File/E-Mail-Archiv konfiguriert haben, dass in einem dieser Archive Source-Bibliotheken enthalten sein sollen, beachten Sie bei der Weitergabe/ Auslieferung des Zielsystems oder bei Weitergabe des File/E-Mail-Archivs, dass die im Projekt verwendeten Source-Bibliotheken (*.library) in lesbarer Quellcode-Form im ZIP-Archiv enthalten sind. Beachten Sie dies sowohl bei der Referenzierung von Bibliotheken als auch bei der Weitergabe des Zielsystems oder von Archiven.

Wenn Sie den lesbaren Quellcode von (Anwender-)Bibliotheken nicht im Target- oder File/E-Mail-Archiv weitergeben möchten, können Sie diese Bibliotheken entweder als kompilierte Bibliotheken (*.compiled-library) speichern und referenzieren oder Sie können das Hinzufügen von Source-Bibliotheken zum Target- oder File/E-Mail-Archiv in den Projekteinstellungen deaktivieren.

Weitere Informationen zu den Konfigurationsmöglichkeiten des Target- und File/E-Mail-Archivs finden Sie im Abschnitt SPS-Projekteinstellungen > Registerkarte Settings [▶ 969].

Informationen zu dem Thema Sourcecode-Verschlüsselung finden Sie in der Dokumentation Security Management.

13.2 Bibliothekserstellung

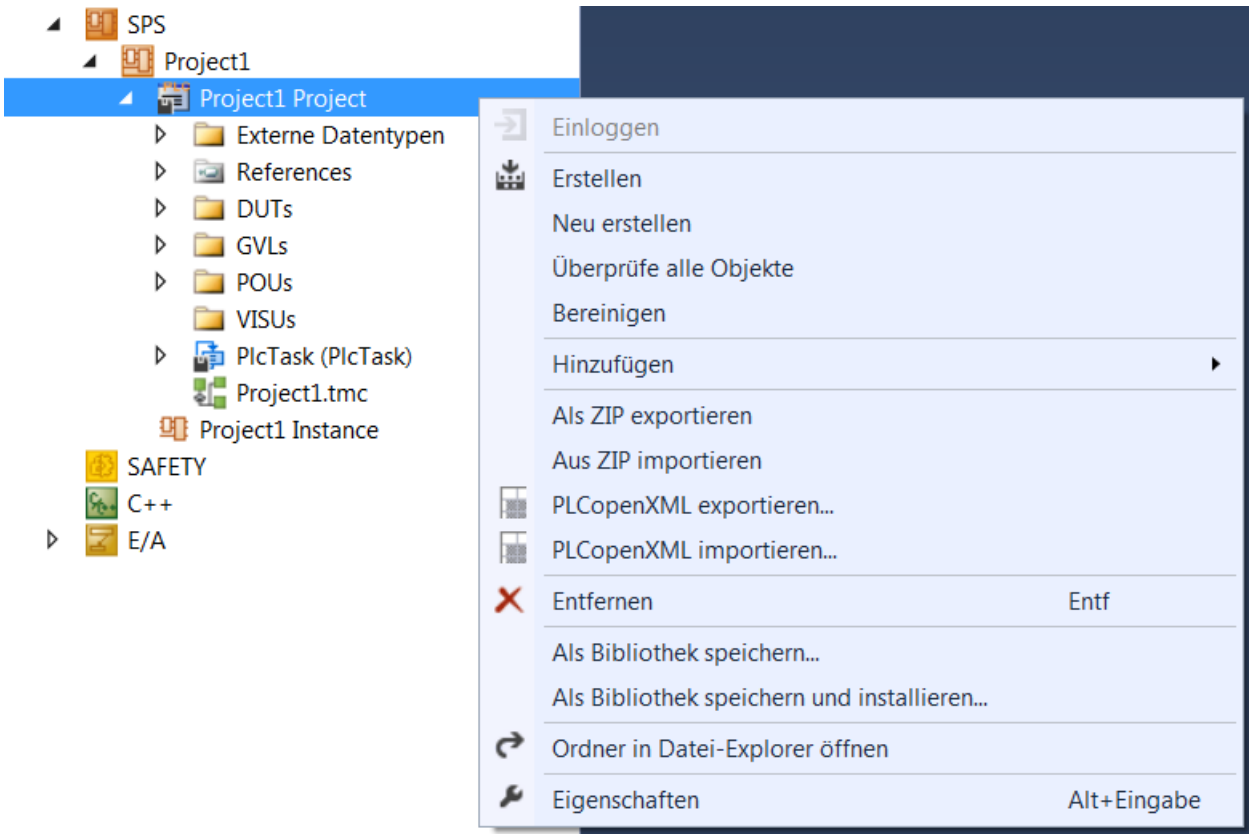
- Ein TwinCAT-3-PLC-Projekt kann als Bibliothek (<Projektname>.library/.compiled-library) gespeichert und gegebenenfalls gleichzeitig in das Bibliotheksrepository [▶ 285] installiert werden. Wenn gezielt ein Bibliotheksprojekt erstellt werden soll, ist zu empfehlen, im Dialog **Neues Projekt** direkt die Vorlage **Leeres SPS-Projekt** auszuwählen.
- Um ein Projekt als Bibliothek speichern zu können, müssen in den Projekteigenschaften [▶ 948] ein **Titel**, eine **Versionsnummer** und der **Firmenname** eingetragen werden.
- Wenn die Bibliothek andere Bibliotheken einbindet, sollte überlegt werden, wie sich diese referenzierten Bibliotheken später, wenn die „Vater“-Bibliothek in ein Projekt eingebunden wird, verhalten sollten. Dabei geht es um Versionshandhabung, Namensraum, Sichtbarkeit und Zugriffsmöglichkeiten, die teilweise im Eigenschaften-Dialog der einzelnen referenzierten Bibliothek konfiguriert werden können. Wenn die Bibliothek später bei ihrer Einbindung in ein Projekt immer auf eine andere Bibliothek verweisen soll, kann beim Definieren der Referenz ein Platzhalter eingesetzt werden.
- Wenn die Bibliotheksmodule vor Ansicht und Zugriff geschützt werden sollen, kann ein Bibliotheksprojekt in einem vorkompilierten Format (<projektname>.compiled-library) gespeichert werden (siehe Befehl Als Bibliothek speichern [▶ 282]).
- Datenstrukturen einer Bibliothek können als Bibliotheks-intern markiert werden. Diese nicht-öffentlichen Objekte tragen die Zugriffskennzeichnung INTERNAL oder das Attribut 'hide' [▶ 841] und erscheinen daher nicht innerhalb des Bibliotheksverwalters [▶ 289], der „Komponenten Auflisten“-Funktionalität oder des Eingabe-Assistenten.
- Um dem Anwender der Bibliothek auf einfache Weise Informationen zu einem Bibliotheksbaustein bereitzustellen, kann der Deklaration eines Bausteinparameters ein entsprechender Kommentar hinzugefügt werden. Dieser wird dann später, wenn die Bibliothek in ein Projekt eingebunden ist, im Bibliotheksverwalter [▶ 289] auf dem Registerblatt **Dokumentation** dargestellt. Beachten Sie außerdem die Möglichkeiten der Bibliotheksdokumentation [▶ 296].
- Die folgenden Befehle zum Speichern einer Bibliothek stehen im Kontextmenü des SPS-Projekts zur Verfügung:
  - Befehl Als Bibliothek speichern [▶ 282]
  - Befehl Als Bibliothek speichern und installieren [▶ 284]

13.2.1 Befehl Als Bibliothek speichern

Funktion: Der Befehl öffnet den Standarddialog zum Speichern eines SPS-Projekts als SPS-Bibliothek.

Aufruf: Kontextmenü des SPS-Projektobjekts (<SPS-Projektname>Project) im Projektmappen-Explorer

Ein SPS-Projekt kann als SPS-Bibliothek gespeichert werden, um Quellcode für andere Applikationen als Bibliothek und damit über eine definierte Schnittstelle zur Verfügung zu stellen. Der Befehl zum Speichern einer Bibliothek ist im Kontextmenü des SPS-Projekts verfügbar.

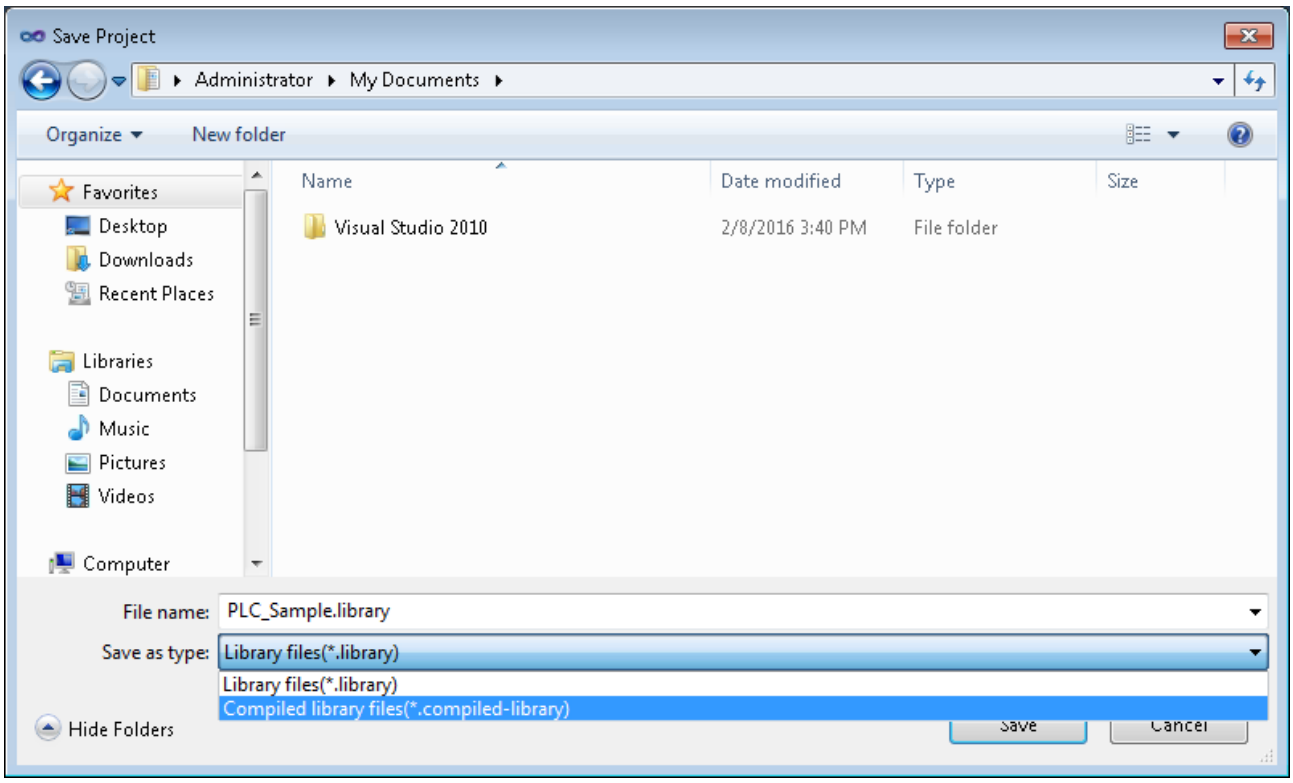


Der Befehl öffnet den Standarddialog zum Speichern einer Datei im Dateisystem. Automatisch wird der bisherige Projektname angeboten – dieser kann bei Bedarf auch verändert werden. Beim Speichern eines Projekts als Bibliothek kann zwischen zwei Bibliotheksdateiformaten gewählt werden:

- *.library (Source-Bibliothek)
  - Eine Source-Bibliothek können Sie mithilfe des Befehls **Bestehendes Element hinzufügen**, der auf dem SPS-Knoten innerhalb des Projektbaums verfügbar ist, öffnen (zur Einsicht und/oder zur Bearbeitung).
  - Sie können mithilfe der üblichen Debug-Funktionalitäten in eine Source-Bibliothek „hineinsteppen“.
- *.compiled-library (übersetzte Bibliothek)
  - Mit dieser Dateierweiterung kann ein Bibliotheksprojekt in kompiliertem Format gespeichert werden. Dabei wird ein verschlüsseltes Abbild des Precompile-Kontexts der Bibliothek abgelegt, was bedeutet, dass die Implementierungen der Bibliotheksbausteine nicht mehr zugänglich oder sichtbar sind.
  - Sie können eine übersetzte Bibliothek daher weder öffnen noch debuggen.
  - In der weiteren Handhabung verhalten sich *.compiled-library-Dateien genauso wie *.library-Dateien. Sie können sie also auf die gleiche Art und Weise installieren und referenzieren.
  - Durch Verwendung einer übersetzten Bibliothek kann zum einen der Quellcode einer Bibliothek geschützt werden und zum anderen ergibt sich der Vorteil kürzerer Ladezeiten und kleinerer Bibliotheksdateien.

Schrittweises Durchlaufen des Codes nicht möglich

Die üblichen Debug-Funktionalitäten können auf eine kompilierte Bibliothek (*.compiled-library) nicht angewendet werden. Es ist somit nicht möglich, in einen Bibliotheksbaustein einer *.compiled-library per Debugging hineinzuspringen.

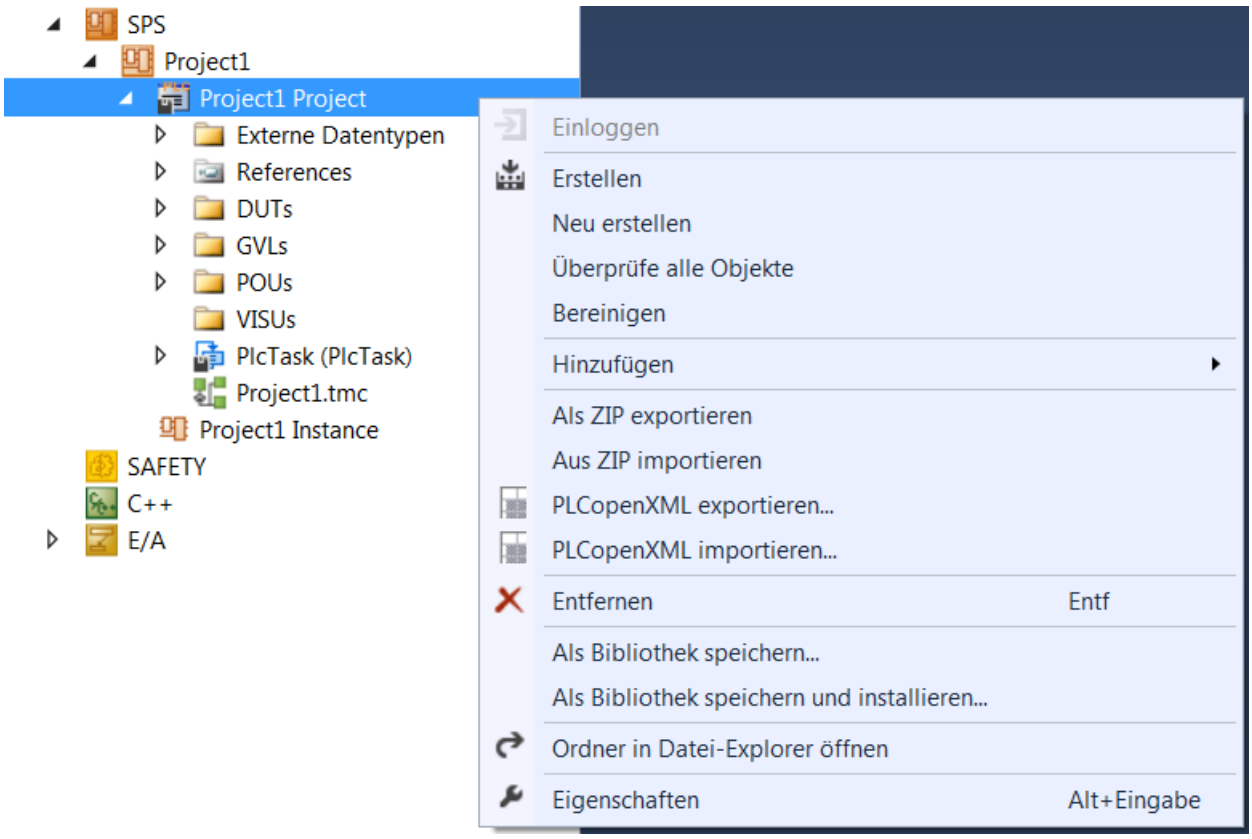


13.2.2 Befehl Als Bibliothek speichern und installieren

Funktion: Der Befehl öffnet den Standarddialog zum Speichern eines SPS-Projekts als SPS-Bibliothek. Zusätzlich installiert der Befehl die gespeicherte Bibliothek in das Bibliotheksrepository. Die Bibliothek kann damit direkt über den Bibliotheksverwalter in ein Projekt eingefügt werden.

Aufruf: Kontextmenü des SPS-Projektobjekts (<SPS-Projektname>Project) im Projektmappen-Explorer

Dieser Befehl speichert das SPS-Projekt als SPS-Bibliothek und installiert sie ins Bibliotheksrepository [▶ 285]. Der Befehl zum Speichern und Installieren einer Bibliothek ist im Kontextmenü des SPS-Projekts verfügbar.



Die zusätzlich zur Speicherung durchgeführte Installation der Bibliothek ist eine Erweiterung zu dem Befehl Als Bibliothek speichern [▶ 282], da die Bibliothek gleichzeitig auf dem lokalen System installiert wird. Dadurch steht die Bibliothek via Bibliotheksverwalter unmittelbar zum Einfügen in einem Projekt zur Verfügung.

13.3 Bibliotheksinstallation

- Voraussetzung für die Verwendung einer Bibliothek in einem Projekt ist die vorherige Installation der Bibliothek auf dem System. Bibliotheken werden auf dem lokalen System in verschiedenen „Repositories“ (Verzeichnisse, Speicherorte) verwaltet. Bevor eine Bibliothek in ein Projekt eingebunden werden kann, muss sie mit einer definierten Versionsnummer auf dem lokalen System in ein solches Repository installiert werden.
  - **Ausnahme**: Projekte, die als Bibliothek referenziert werden.
    Siehe SPS-Projekt als referenzierte Bibliothek verwenden.
- Ist eine verwendete Bibliotheksversion nicht im Repository installiert, wird dies über ein Hinweissymbol im Projektbaum an der Referenz verdeutlicht.
- Die Installation von Bibliotheken erfolgt im Bibliotheksrepository [▶ 285].

13.3.1 Bibliotheksrepository

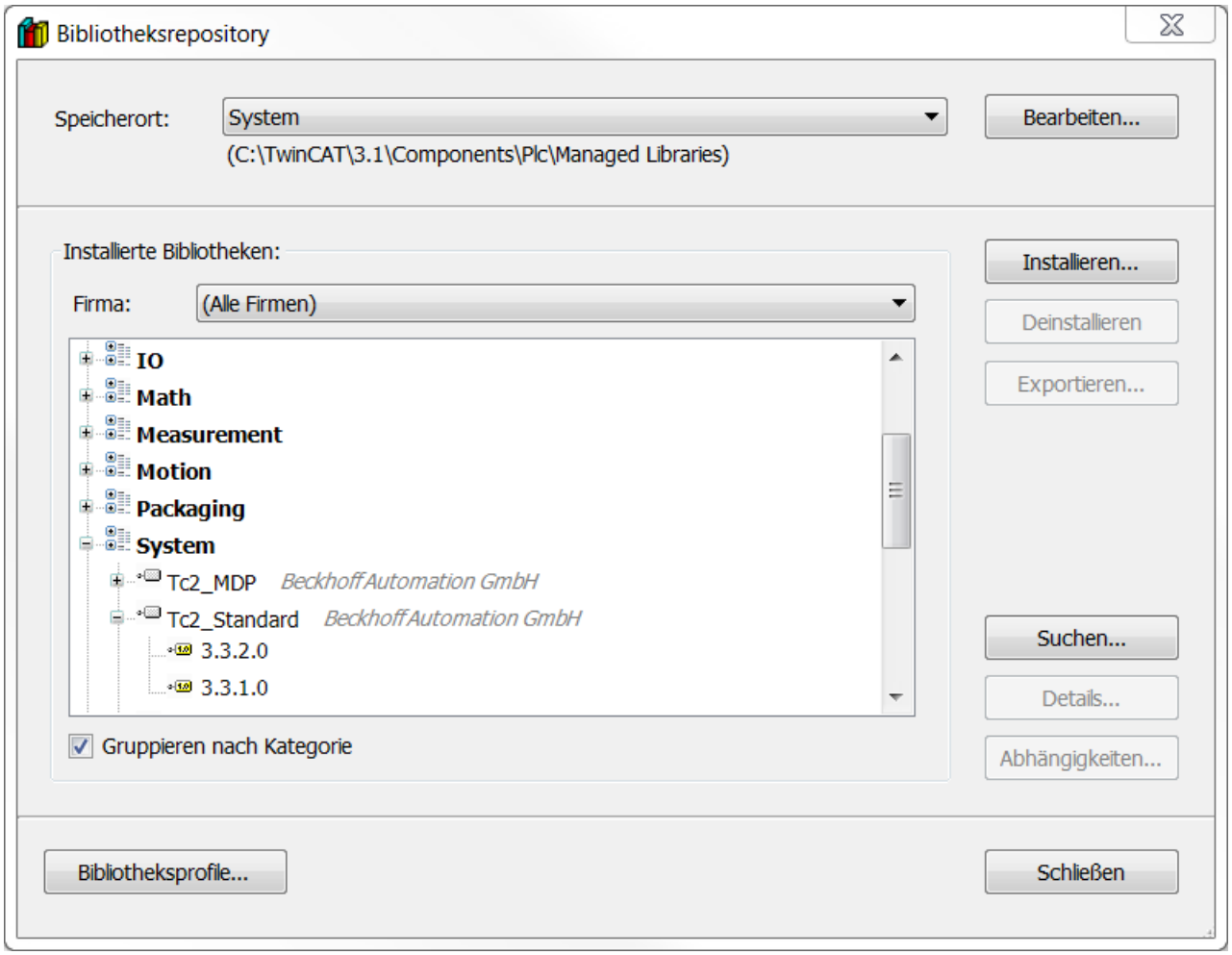
Funktion: Das Bibliotheksrepository kann für die Definition von Speicherorten und für die Installation oder Deinstallation von Bibliotheken verwendet werden.

Aufruf:

- Menü **PLC**
- Kontextmenü des Objekts **References** im SPS-Projektbaum
- Schaltfläche im Bibliotheksverwalter [▶ 289] (Symbol:)
- Über den erweiterten Dialog **Bibliothek suchen** nach Verwendung von dem Befehl Bibliothek ohne Platzhalterauflösung hinzufügen [▶ 374]

Um eine Bibliothek nutzen zu können, muss sie im Repository installiert sein. Für die Beckhoff-Bibliotheken geschieht dies im Allgemeinen bereits bei der TwinCAT 3 Installation oder bei der Installation von TwinCAT 3 Functions. Bibliotheken können als Quellbibliothek (*.library) oder als kompilierte Bibliothek (*.compiled-library) installiert sein. Beckhoff liefert kompilierte Bibliotheken aus. Ist eine verwendete Bibliotheksversion nicht im Repository installiert, wird dies bereits über ein Hinweis-Symbol im Projektbaum an der Referenz verdeutlicht.

Das Bibliotheksrepository enthält alle installierten Bibliotheken. Diese Auflistung kann entsprechend der Bibliothekskategorien (Option **Gruppieren nach Kategorie** ist aktiviert) oder alphabetisch nach den Bibliothekstiteln (Option **Gruppieren nach Kategorie** ist deaktiviert) sortiert und angezeigt werden. Bei der Sortierung nach Bibliothekskategorien erscheinen die Kategorien als Knoten, ein Klick auf einen Knoten öffnet die Liste der zugehörigen Bibliotheken bzw. Unterkategorien, ein Klick auf einen Bibliotheksnamen öffnet die Liste der installierten Bibliotheksversionen.



Schaltflächen und Befehle

Die folgenden Schaltflächen und Befehle sind im **Bibliotheksrepository** verfügbar.

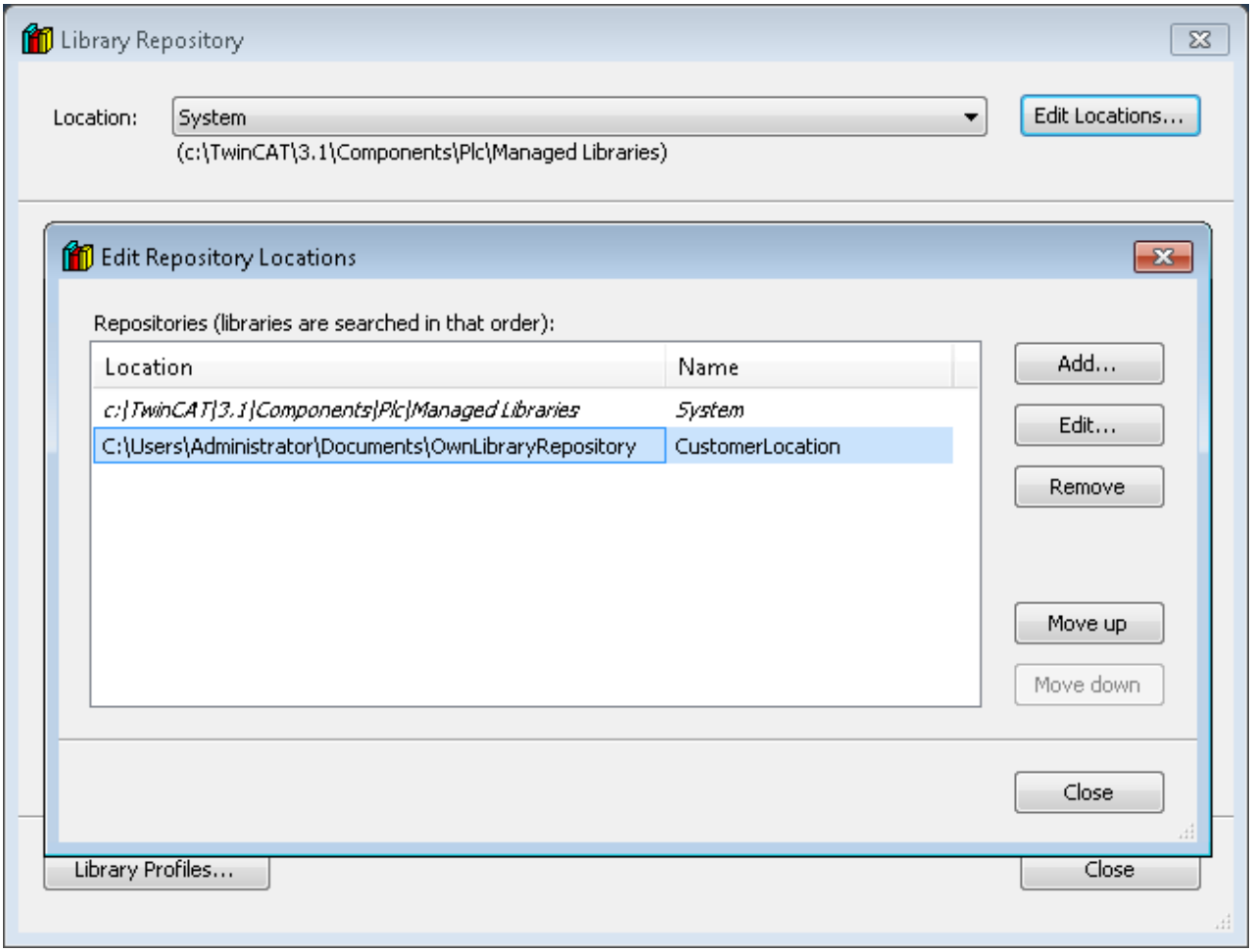
Speicherort

Anzeige des Verzeichnisses auf dem lokalen System, in dem die Bibliotheksdateien liegen. Die Bibliotheken dieses Speicherorts sind im Bereich **Installierte Bibliotheken** aufgelistet. Falls auf dem System mehrere Repository-Verzeichnisse vorhanden sind, kann an dieser Stelle ein Repository-Verzeichnis zur Verwaltung ausgewählt werden.	
Bearbeiten	Öffnet den Dialog **Repository-Speicherorte bearbeiten**.

Sie können für neue Repositorys nur leere Verzeichnisse oder ein bereits bestehendes gültiges Repository verwenden.

Repository-Speicherorte bearbeiten

Auflistung der Repositories mit Speicherort und Name.	
Hinzufügen	Legt durch Angabe eines Repository-Namens und -Verzeichnisses ein neues Repository an. Öffnet den Dialog **Speicherort für Repository**. Das ausgewählte Verzeichnis (Eingabefeld **Ort**) muss leer sein oder ein bereits bestehendes gültiges Repository sein. **Name** ist das Eingabefeld für einen symbolischen Repository-Namen.
Bearbeiten	Öffnet den Dialog **Speicherort für Repository** (siehe „Hinzufügen“) zur Bearbeitung des aktuell selektierten Bibliotheksrepositorys.
Entfernen	Es erscheint eine Abfrage, ob nur der Eintrag aus der Liste der Repositories entfernt werden soll, oder ob auch das Verzeichnis mit den Bibliotheksdateien aus dem Dateisystem gelöscht werden soll. Wenn Sie das Verzeichnis löschen möchten, müssen Sie dies bestätigen.
Hochschieben	Befehl zum Ändern der Repository-Reihenfolge, indem das aktuell selektierte Repository um einen Platz nach oben verschoben wird.
Herunterschieben	Befehl zum Ändern der Repository-Reihenfolge, indem das aktuell selektierte Repository um einen Platz nach unten verschoben wird.



Installierte Bibliotheken

Liste der Bibliotheken in Baumstruktur. Darstellung jeder Bibliothek mit Kategorie, Name, Firma und Version.	
Firma	Auswahlliste zur Filterung der angezeigten Bibliotheken.
Installieren	Öffnet den Dialog **Bibliothek auswählen** zur Auswahl einer Bibliotheksdatei im Dateisystem. Die ausgewählte Bibliotheksdatei vom Typ *.library oder *.compiled-library wird in dem lokalen Bibliotheksrepository installiert. Anschließend kann die Bibliothek in Projekten verwendet werden.
Deinstallieren	Deinstalliert die selektierte Bibliotheksversion. Anschließend kann sie in Projekten nicht mehr verwendet werden.
Exportieren	Über diesen Befehl können Sie eine Bibliothek, die in Ihrem Bibliotheksrepository installiert ist, exportieren und an einem gewünschten Speicherort abgelegen. Der Befehl öffnet den Dialog **Bibliothek exportieren** zur Auswahl eines Speicherorts. Die Bibliothek kann in dem Bibliotheksformat exportiert werden, in dem sie im Repository installiert wurde (als Source-Bibliothek *.library bzw. als kompilierte Bibliothek *.compiled-library).
Suchen	Öffnet den Dialog **Bibliothek suchen** zum Suchen von Bibliotheksnamen oder Bibliothekselementen. Per Doppelklick auf ein Suchergebnis oder per Fokussieren eines Suchergebnisses + [Öffnen], wird der Finden-Dialog geschlossen und die zu dem Suchergebnis gehörende Bibliothek wird im Bibliotheksrepository markiert.
Details	Öffnet für die ausgewählte Version einer Bibliothek den Dialog Details [▶ 376].
Abhängigkeiten	Öffnet für die ausgewählte Version einer Bibliothek den Dialog Abhängigkeiten [▶ 377].
Gruppieren nach Kategorie	•  : Gruppierung nach Bibliothekskategorien •  : Alphabetische Sortierung Die Kategorien werden durch externe Beschreibungsdateien „*.libcat.xml“ definiert.

Bibliotheksprofile

Ein Bibliotheksprofil definiert, mit welcher Bibliotheksversion TwinCAT einen Bibliotheksplatzhalter auflöst, wenn eine bestimmte Compiler-Version im Projekt gesetzt ist.	
Importieren	Importiert eine *.libraryprofile-Datei. Wenn der Import bereits vorhandene Platzhalter-Einträge beinhaltet, erscheint eine Abfrage, ob TwinCAT diese überschreiben soll.
Exportieren	Exportiert eine xml-Datei mit der Erweiterung „.libraryprofile“ mit den Zuordnungen der selektierten Platzhaltereinträge. Sie können auch nur einen einzigen Eintrag einer Compiler-Version selektieren.

Konvertierte TwinCAT-2-Bibliotheken im Bibliotheksrepository

Das Bibliotheksrepository enthält konvertierte TwinCAT-2-Bibliotheken (Tc2_<LibraryName>) und neue TwinCAT-3-Bibliotheken (Tc3_<LibraryName>), die in TwinCA 2 nicht verfügbar sind.

Mit diesen Bibliotheken ist es möglich, ein TwinCAT-2-Projekt, das TwinCAT-2-Bibliotheken nutzt, in ein TwinCAT-3-Projekt, das kompatible TwinCAT-3-Bibliotheken nutzt, konvertieren zu können, ohne viel am SPS-Code ändern zu müssen. Deshalb sind die Tc2_<LibraryName>-Bibliotheken konvertierte TC2-Bibliotheken. Die Benennung der Bibliotheken in TwinCAT 3 ist ähnlich derer in TwinCAT 2. Zur Vereinfachung sind in manchen Fällen einige Bibliotheken zusammengefasst worden. Der Projektkonverter weist automatisch die TC2-Bibliotheken einer TwinCAT 2 .pro-Datei den TC3-Bibliotheken in einem TwinCAT-3-Projekt zu.

Eine Liste der Zuweisungen der SPS-Bibliotheken TC2-Bibliothek → TC3-Bibliothek ist in den TwinCAT-Optionen abgelegt und dort erweiterbar:
Tools\Options\TwinCAT\PLC Environment\Libraries

Ältere Bibliotheken müssen vor der Projektkonvertierung ausgetauscht werden (i.e. MC → MC2, COMlib → COMlibv2, PLCSYSTEM → TcSystem, ...).

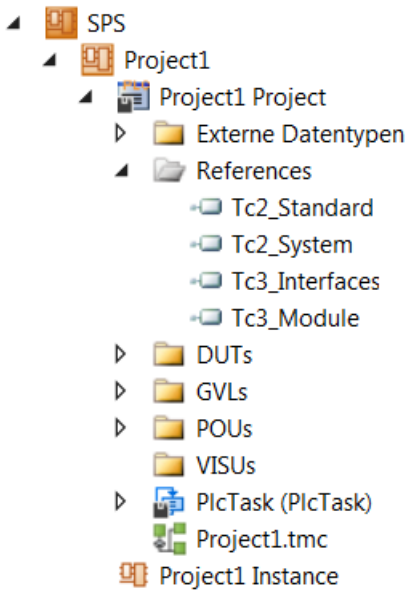
13.4 Bibliotheksverwaltung

- Der Bibliotheksverwalter [▶ 289] bietet einen guten Überblick über die im Projekt verwendeten SPS-Bibliotheksreferenzen und kann genutzt werden, um Bibliotheken bzw. Platzhalter in ein Projekt einzubinden. Durch die Einbindung können die durch die Bibliotheksreferenz bereitgestellten Elemente im Projekt verwendet werden.
- Bibliotheksreferenzen, die als Unterbibliotheken in einer anderen Bibliothek referenziert sind, werden im Bibliotheksverwalter ebenfalls dargestellt. Es gibt allerdings auch „versteckte Bibliotheken“ (siehe Abschnitt Befehl Eigenschaften [▶ 377]).
- Bei Verwendung einer Bibliothek wird immer eine eindeutige Version der Bibliothek referenziert. Welche dies ist, wird als effektive Version angegeben. Falls die Bibliothek mit einer festen Version angefügt wurde (z. B. 3.3.0.0), wird das Projekt immer diese Bibliotheksversion nutzen, auch wenn von dieser Bibliothek bereits neuere Versionen existieren sollten. Das Projekt kann über die Einstellung „immer neueste“/„*“ aber auch automatisch immer die neueste Version einer Bibliothek nutzen. In diesem Fall verwendet TwinCAT immer die neueste im Bibliotheksrepository gefundene Version der Bibliothek. Weitere Informationen sowie ein Beispiel finden Sie unter Befehl Immer neueste Version verwenden [▶ 381].
- Es ist nicht möglich, die gleiche Version derselben Bibliothek mehr als einmal zu einem Bibliotheksverwalter hinzufügen. Eine Version einer Bibliothek kann in einem Bibliotheksverwalter entweder als Bibliothek oder als Platzhalter referenziert werden.
- Liegt die Bibliothek nicht in übersetzter Form (*.compiled-library) vor, sondern ist die *.library-Datei vorhanden, so können die im Bibliotheksverwalter aufgeführten Elemente der Bibliothek durch einen Doppelklick auf den jeweiligen Eintrag geöffnet werden.
- Sie können Bibliotheksreferenzen in Form einer Bibliothek oder in Form eines Platzhalters zum Bibliotheksverwalter hinzufügen und so in Ihre Anwendung einbinden (siehe Abschnitt Befehl Bibliothek hinzufügen [▶ 373]). Es sollten nach Möglichkeit Platzhalter verwendet werden. Weitere Informationen finden Sie im Abschnitt Bibliotheksplatzhalter [▶ 293].
- Wenn ein Bibliotheksmodul im Projekt angesprochen wird, werden die Bibliotheken und Repositories in der Reihenfolge durchsucht, in der sie im Bibliotheksrepository [▶ 285] aufgeführt sind. Weitere Informationen finden Sie im Abschnitt Eindeutiger Zugriff auf Bibliotheksmodule oder -variablen [▶ 280].

13.4.1 Bibliotheksverwalter

Funktion: Der Bibliotheksverwalter dient dem Einbinden und Verwalten von Bibliotheken in einem Projekt und bietet einen guten Überblick über die im Projekt verwendeten SPS-Bibliotheken.

Aufruf: Doppelklick auf das Objekt **References** im SPS-Projektbaum



Übersetzungsfehler bezüglich des Bibliotheksverwalters werden in einer eigenen Kategorie im Meldungsfenster ausgegeben.

Oberer Teil des Bibliotheksverwalters

Der obere Teil des Dialogs zeigt die aktuell im Projekt eingebundenen Bibliotheken.	
Name	Titel, Version und der Firmenname, wie sie in den Projekteigenschaften [▶ 948] während der Bibliothekserstellung [▶ 282] definiert wurden.
Namensraum	Die Standardeinstellung für den Namensraum einer Bibliothek entspricht dem Bibliothekstitel. Für eine Bibliothek kann hingegen auch explizit ein davon abweichender Namensraum definiert werden: entweder allgemein für die Bibliothek bei der Bibliothekserstellung [▶ 282] in den Projekteigenschaften [▶ 948] oder für den lokalen Gebrauch der Bibliothek in einem Projekt im Eigenschaftenfenster der Bibliotheksreferenz [▶ 377]. Der Namensraum der Bibliothek muss als Präfix des Bezeichners verwendet werden, damit ein eindeutiger Zugriff auf ein Modul möglich ist, das mehrfach im Projekt vorhanden ist.
Effektive Version	Effektive Version der Bibliothek, auf die referenziert wird. Wenn die Bibliothek als „immer neueste“/„*“ verwendet wird, wird an dieser Stelle die tatsächlich verwendete Bibliotheksversion angezeigt. Weitere Informationen zu der Auflösung von Platzhaltern finden Sie im Abschnitt Platzhalter [▶ 294].

- Bibliotheken, die automatisch durch ein Plug-In im Projekt eingefügt wurden (z. B. Visualisierungs- oder UML-Bibliotheken), sind in grauer Schrift dargestellt. Manuell eingefügte Bibliotheken (**Bibliothek hinzufügen**…) werden in schwarzer Schrift dargestellt.
- Ein Symbol vor dem Bibliotheksnamen zeigt den Typ der Bibliothek an:
  - : TwinCAT-3-PLC-Bibliothek (enthält Versionsinformationen)
  - : Die referenzierte Datei konnte nicht gefunden werden oder ist keine gültige Bibliotheksdatei (siehe entsprechende Meldung in der Ansicht **Fehlerliste**). Siehe in diesem Fall: Befehl Bibliothek erneut laden [▶ 375]
- Wenn eine Bibliothek Abhängigkeiten [▶ 377] von anderen Bibliotheken hat, werden diese – wenn sie gefunden werden – automatisch ebenfalls eingebunden und mit vorangestelltem Symbol in einem Unterbaum des Eintrags dargestellt. Ein solcher Unterbaum kann über ein Plus- oder Minuszeichen geöffnet bzw. geschlossen werden. Als Beispiel sehen Sie unten in der Abbildung die Bibliothek „Tc2_Standard“ als direkte Bibliothek und als Unterbibliothek der Bibliothek „Tc2_System“.

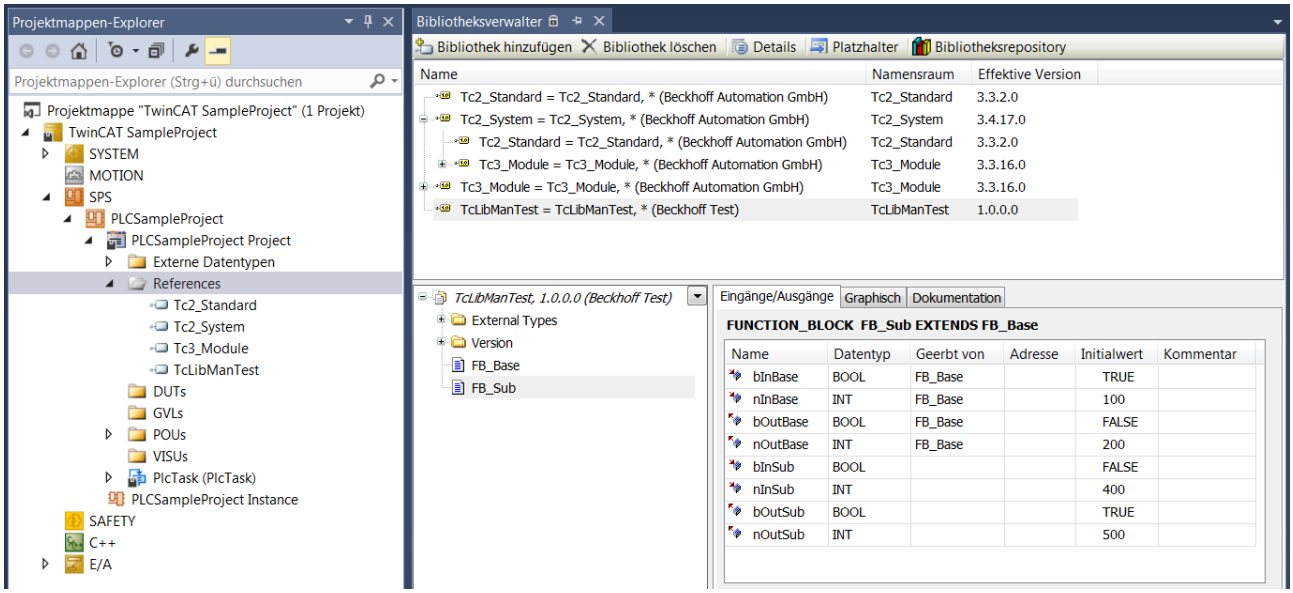
Schaltflächen und Befehle

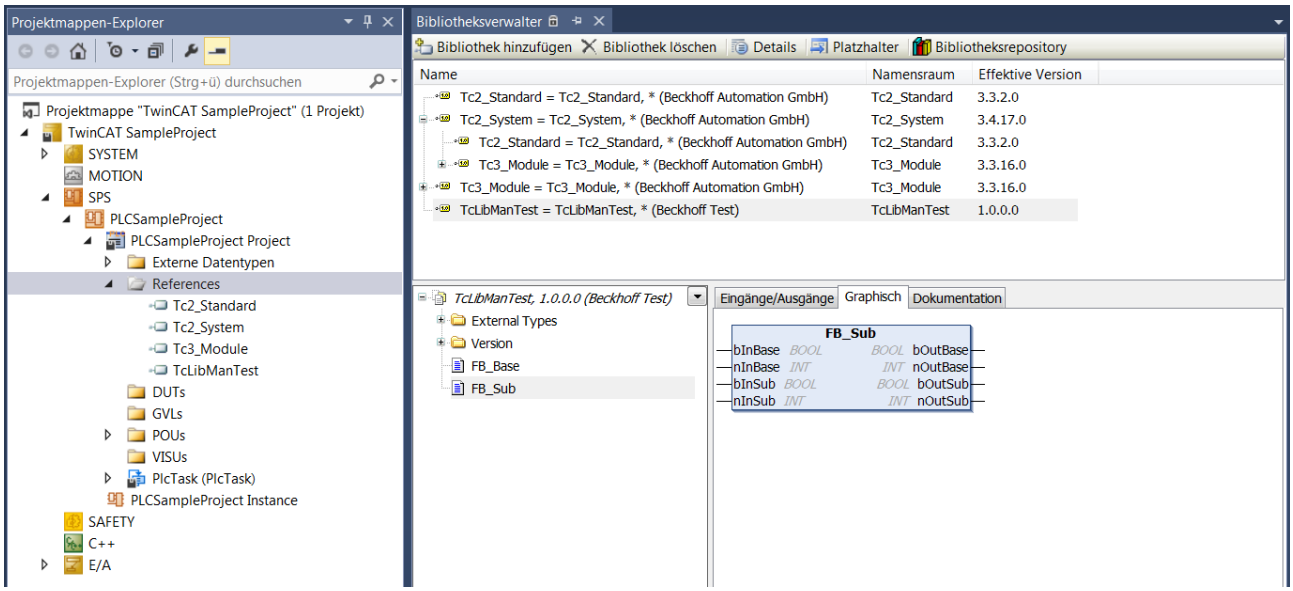
Die folgenden Schaltflächen und Befehle sind im oberen Bereich des Bibliotheksverwalters verfügbar.

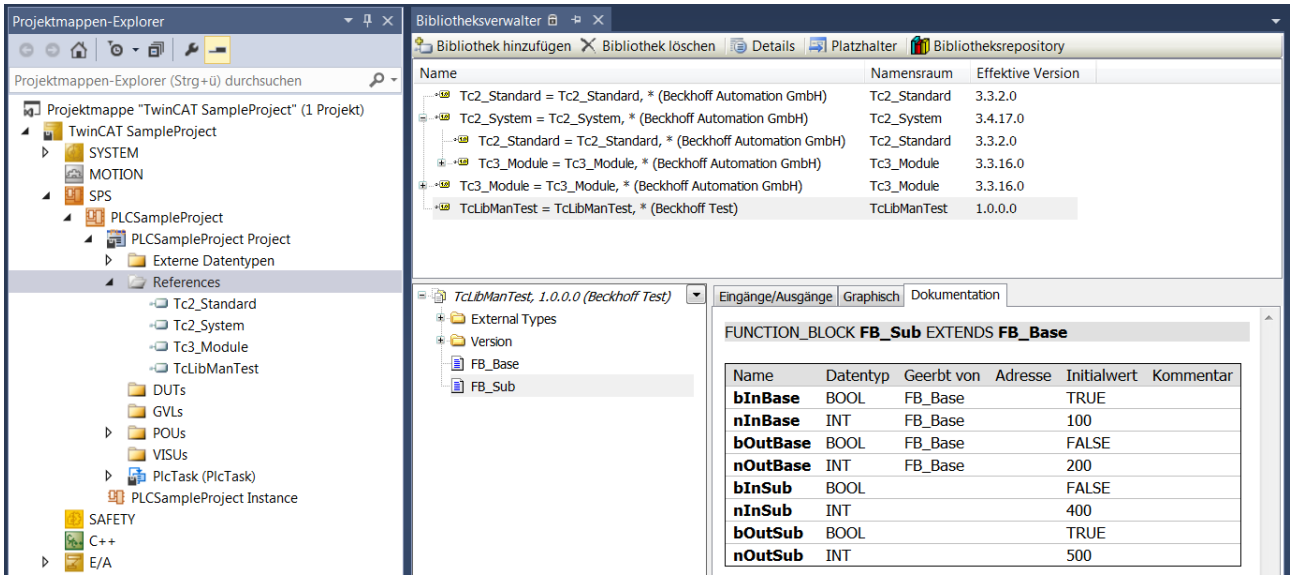
Bibliothek hinzufügen	Einbinden einer Bibliothek oder eines Platzhalters ins Projekt. Sehen Sie hierzu auch Befehl Bibliothek hinzufügen [▶ 373].
Bibliothek entfernen	Der Befehl entfernt die selektierte Bibliothek aus dem Bibliotheksverwalter.
Details	Öffnet für die ausgewählte Bibliothek den Dialog Details [▶ 376].
Platzhalter	Öffnet den Dialog Platzhalter [▶ 294].
Bibliotheksrepository	Öffnet den Dialog Bibliotheksrepository [▶ 285].
Bibliothek erneut laden	Dieser Befehl ist im Editorfenster des Bibliotheksverwalters verfügbar, wenn eine Bibliothek ausgewählt ist, deren Laden beim Öffnen des Projekts fehlgeschlagen ist. Sehen Sie hierzu auch Befehl Bibliothek erneut laden [▶ 375]

Unterer Teil des Bibliotheksverwalters

Im unteren linken Teil des Editors werden die Module der Bibliothek dargestellt, die im oberen Teil des Editors ausgewählt ist. Über ein Menüsymbol, das rechts neben dem Bibliotheksnamen dargestellt ist, stehen übliche Sortier- und Suchfunktionen zur Verfügung. Im unteren rechten Teil finden sich die folgenden Registerkarten.	
Eingänge/Ausgänge	Die Komponenten des gerade links ausgewählten Bibliotheksobjekts werden in einer Tabelle dargestellt, mit (Variablen-)Name, Datentyp, ggf. Basisbaustein, Adresse, Initialwert und Kommentar, wie in der Bibliothek definiert. Das Symbol, das den jeweiligen Variablen vorangestellt ist, zeigt an, ob es sich um eine Eingangs-, Ausgangs- oder Ein-/Ausgangsvariable handelt. • Eingangsvariable: Symbol mit Pfeil, der nach unten rechts zeigt • Ausgangsvariable: Symbol mit Pfeil, der nach oben links zeigt • Ein-/Ausgangsvariable: Symbol mit Pfeil, der sowohl nach unten rechts als auch nach oben links zeigt Das Symbol für Methodenrückgabewerte entspricht dem Symbol für Ausgangsvariablen.
Bibliotheksparameter	Diese Registerkarte ist nur verfügbar, wenn es sich bei dem gerade links ausgewählten Bibliotheksobjekt um eine Parameterliste handelt. Die Variablen der Parameterliste werden in einer Tabelle dargestellt, mit Name, Datentyp, Wert (editierbar) und Kommentar, wie in der Bibliothek definiert. Über die Spalte **Wert (editierbar)** können Sie den Wert der globalen Konstanten durch einen projektspezifischen Wert ersetzen. Weitere Informationen finden Sie im Abschnitt Objekt Parameterliste [▶ 75].
Graphisch	Graphische Darstellung des Bausteins
Dokumentation	Die Komponenten des gerade links ausgewählten Bibliotheksobjekts werden in einer Tabelle dargestellt, mit (Variablen-)Name, Datentyp, ggf. Basisbaustein, Adresse, Initialwert und dem Kommentar, der ggf. bei der Bibliothekserstellung [▶ 282] der Deklaration beigefügt wurde. Somit ist das Einfügen solcher Kommentare ein einfacher Weg, dem Anwender automatisch Dokumentation eines Bausteins zur Verfügung zu stellen. Detaillierte Informationen hierzu finden Sie im Abschnitt Bibliotheksdokumentation [▶ 296].


Projektmappen-Explorer
Projektmappen-Explorer (Strg+ü) durchsuchen
Projektmappe "TwinCAT SampleProject" (1 Projekt)
TwinCAT SampleProject
SYSTEM
MOTION
SPS
PLCSampleProject
PLCSampleProject Project
Externe Datentypen
References
Tc2_Standard
Tc2_System
Tc3_Module
TcLibManTest
DUTs
GVLs
POUs
VISUs
PlcTask (PlcTask)
PLCSampleProject Instance
SAFETY
C++
E/A
Bibliotheksverwalter
Bibliothek hinzufügen
Bibliothek löschen
Details
Platzhalter
Bibliotheksrepository
Name
Namensraum
Effektive Version
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH) Tc2_Standard 3.3.2.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH) Tc2_System 3.4.17.0
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH) Tc2_Standard 3.3.2.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH) Tc3_Module 3.3.16.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH) Tc3_Module 3.3.16.0
TcLibManTest = TcLibManTest, * (Beckhoff Test) TcLibManTest 1.0.0.0
TcLibManTest, 1.0.0.0 (Beckhoff Test)
External Types
Version
FB_Base
FB_Sub
Eingänge/Ausgänge
Graphisch
Dokumentation
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
Name Datentyp Geerbt von Adresse Initialwert Kommentar
bInBase BOOL FB_Base TRUE
nInBase INT FB_Base 100
bOutBase BOOL FB_Base FALSE
nOutBase INT FB_Base 200
bInSub BOOL FALSE
nInSub INT 400
bOutSub BOOL TRUE
nOutSub INT 500


Projektmappen-Explorer
Projektmappen-Explorer (Strg+ü) durchsuchen
Projektmappe "TwinCAT SampleProject" (1 Projekt)
TwinCAT SampleProject
SYSTEM
MOTION
SPS
PLCSampleProject
PLCSampleProject Project
Externe Datentypen
References
Tc2_Standard
Tc2_System
Tc3_Module
TcLibManTest
DUTs
GVLs
POUs
VISUs
PlcTask (PlcTask)
PLCSampleProject Instance
SAFETY
C++
E/A
Bibliotheksverwalter
Bibliothek hinzufügen
Bibliothek löschen
Details
Platzhalter
Bibliotheksrepository
Name
Namensraum
Effektive Version
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH) Tc2_Standard 3.3.2.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH) Tc2_System 3.4.17.0
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH) Tc2_Standard 3.3.2.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH) Tc3_Module 3.3.16.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH) Tc3_Module 3.3.16.0
TcLibManTest = TcLibManTest, * (Beckhoff Test) TcLibManTest 1.0.0.0
TcLibManTest, 1.0.0.0 (Beckhoff Test)
External Types
Version
FB_Base
FB_Sub
Eingänge/Ausgänge
Graphisch
Dokumentation
FB_Sub
bInBase BOOL
nInBase INT
bInSub BOOL
nInSub INT
BOOL bOutBase
INT nOutBase
BOOL bOutSub
INT nOutSub


Projektmappen-Explorer
Projektmappen-Explorer (Strg+ü) durchsuchen
Projektmappe "TwinCAT SampleProject" (1 Projekt)
TwinCAT SampleProject
SYSTEM
MOTION
SPS
PLCSampleProject
PLCSampleProject Project
Externe Datentypen
References
Tc2_Standard
Tc2_System
Tc3_Module
TcLibManTest
DUTs
GVLs
POUs
VISUs
PlcTask (PlcTask)
PLCSampleProject Instance
SAFETY
C++
E/A
Bibliotheksverwalter
Bibliothek hinzufügen
Bibliothek löschen
Details
Platzhalter
Bibliotheksrepository
Name
Namensraum
Effektive Version
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH) Tc2_Standard 3.3.2.0
Tc2_System = Tc2_System, * (Beckhoff Automation GmbH) Tc2_System 3.4.17.0
Tc2_Standard = Tc2_Standard, * (Beckhoff Automation GmbH) Tc2_Standard 3.3.2.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH) Tc3_Module 3.3.16.0
Tc3_Module = Tc3_Module, * (Beckhoff Automation GmbH) Tc3_Module 3.3.16.0
TcLibManTest = TcLibManTest, * (Beckhoff Test) TcLibManTest 1.0.0.0
TcLibManTest, 1.0.0.0 (Beckhoff Test)
External Types
Version
FB_Base
FB_Sub
Eingänge/Ausgänge
Graphisch
Dokumentation
FUNCTION_BLOCK FB_Sub EXTENDS FB_Base
Name Datentyp Geerbt von Adresse Initialwert Kommentar
bInBase BOOL FB_Base TRUE
nInBase INT FB_Base 100
bOutBase BOOL FB_Base FALSE
nOutBase INT FB_Base 200
bInSub BOOL FALSE
nInSub INT 400
bOutSub BOOL TRUE
nOutSub INT 500

13.5 Bibliotheksplatzhalter

Ein Bibliotheksplatzhalter ist ein Platzhalter, der auf eine bestimmte Bibliothek verweist. Der Platzhalter kann dabei entweder auf eine feste Version oder auf die „immer neueste“ Version dieser Bibliothek aufgelöst werden.

Die Anleitung, wie Sie einen neuen Bibliotheksplatzhalter erstellen, finden Sie im Kapitel Befehl Bibliothek hinzufügen.

Die Auflösung aller im Projekt befindlichen Platzhalter stellen Sie auf Applikationsebene ein. Das bedeutet, dass Sie auf Applikationsebene sowohl die Auflösung von den Platzhaltern einstellen können, die direkt auf Applikationsebene eingebunden sind, als auch von solchen, die innerhalb von referenzierten Bibliotheken verwendet werden.
Sie können also von außen festlegen, auf welche Bibliothek ein Bibliotheksplatzhalter aufgelöst werden soll, der innerhalb einer anderen Bibliothek eingebunden ist. Es ist nicht nötig, die äußere Bibliothek erneut zu speichern, um die Versionen der intern verwendeten Bibliotheksplatzhalter zu ändern. Weitere Informationen, wie die Auflösung von Platzhaltern festgelegt werden kann, finden Sie im Abschnitt Platzhalterauflösung ändern [▶ 295].

Aufgrund dieser Möglichkeiten ist die Anpassung von verwendeten Bibliotheksversionen mit äußerst geringem Aufwand verbunden, sodass sich der Engineering-Prozess von Projekten und Bibliotheken sehr flexibel gestaltet.

Wenn eine Bibliotheksreferenz in ein Projekt eingebunden wird, sollte somit möglichst ein Platzhalter anstelle einer Bibliothek verwendet werden.

Beispiel

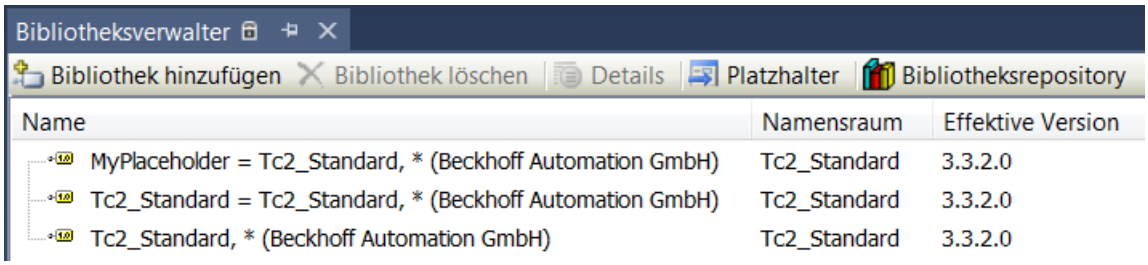
- Ein Applikationsprojekt benötigt die Bibliotheken LibA und LibB.
- Die Bibliothek LibB benötigt ihrerseits auch die Module von LibA.
- LibA wird damit als direkte Bibliothek in der Applikation, aber auch als Unterbibliothek von LibB verwendet.
- Durch die Referenzierung von LibA als Platzhalter – sowohl in der Applikation als auch in LibB – kann durch die Definition der Platzhalterauflösung im Applikationsprojekt festgelegt werden, welche Version der LibA im gesamten Projekt (Applikation und LibB) verwendet werden soll.
- Falls also beispielsweise eine Änderung innerhalb von LibA benötigt wird, wobei die Schnittstellen von LibA unverändert bleiben und die Kompatibilität mit der Vorgängerversion sichergestellt ist, kann dieser neue Bibliotheksstand im gesamten Projekt (Applikation und LibB) auf einfache Weise genutzt werden. Dafür wird die geänderte LibA mit einer neuen Version erstellt und installiert (z. B. Änderung der Version von 1.0.0.0 auf 1.0.0.1). Falls der Platzhalter LibA im Applikationsprojekt als eine feste Bibliotheksversion, also als 1.0.0.0, aufgelöst wurde, wird die Platzhalterauflösung entsprechend auf LibA-Version 1.0.0.1 geändert. Falls der Platzhalter LibA im Applikationsprojekt hingegen als „neueste Version“ definiert wurde, ist keine Anpassung nötig – nach Installation von LibA-Version 1.0.0.1 wird diese automatisch referenziert und im gesamten Projekt (Applikation und LibB) verwendet.

Erkennen von Bibliotheken und Platzhaltern

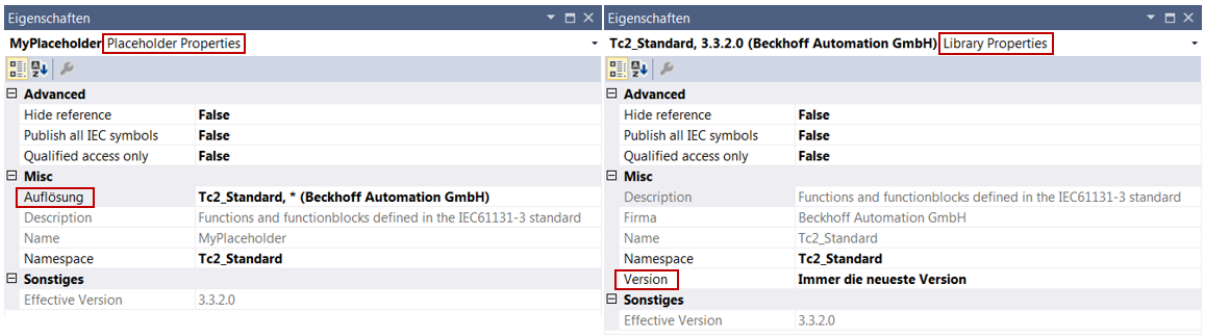
Ob eine Bibliotheksreferenz als Bibliothek oder als Platzhalter eingebunden ist, kann anhand von Unterschieden im Bibliotheksverwalter [▶ 289] und im Eigenschaftenfenster [▶ 377] erkannt werden.

- Bibliotheksverwalter: Bei einem Platzhalter wird der Bibliotheksname dem Platzhalternamen zugewiesen, sodass ersichtlich ist, auf welche „reelle“ Bibliothek der Platzhalter verweist (z. B. MyPlaceholder = Tc2_Standard, * (Beckhoff Automation GmbH)). Bei einer Bibliothek ist dies nicht der Fall: hier wird nur der Bibliotheksname angezeigt (Tc2_Standard, * (Beckhoff Automation GmbH)).

In dem folgenden Screenshot handelt es sich bei den ersten beiden Einträgen im Bibliotheksverwalter um Platzhalter, der dritte Eintrag signalisiert die Verwendung einer Bibliothek.



- Eigenschaftenfenster: Das Eigenschaftenfenster zeigt direkt in der obersten Zeile an, ob es sich um Platzhalter- oder Bibliothekseigenschaften handelt.
  Die in dem Eigenschaftenfenster angezeigten Eigenschaften unterscheiden sich entsprechend: ein Platzhalter wird auf eine bestimmte Bibliothek samt bestimmter Bibliotheksversion **aufgelöst** – bei der Bibliothek ist bereits festgelegt, um welche Bibliothek es sich handelt, lediglich die **Version** kann noch angepasst werden. In beiden Fällen kann im Eigenschaftenfenster der Namensraum konfiguriert werden.



13.5.1 Platzhalter

Funktion: Der Befehl öffnet den Dialog **Platzhalter**. In der Platzhalterübersicht werden alle im Projekt befindlichen Bibliotheksplatzhalter und deren aktuelle Definition der Auflösung aufgelistet. Zudem können Sie über diesen Dialog Platzhalter verwalten und Versionen für Platzhalterbibliotheken spezifizieren (z. B. Auflösung auf eine bestimmte Bibliotheksversion oder auf „immer neueste“/„*“). Die Versionsauflösung ist an dieser Stelle auch für Platzhalter möglich, welche nur intern in anderen Bibliotheken verwendet werden.

Aufruf:

- Kontextmenü des Objekts **References** im SPS-Projektbaum
- Schaltfläche im Bibliotheksverwalter [▶ 289] (Symbol: )

Platzhalter werden standardmäßig auf die „immer neueste“/„*“ Version einer Bibliothek aufgelöst. In dem Platzhalter-Dialog können Sie jede Platzhalterauflösung, die in dem Projekt direkt oder intern verwendet wird, auf eine andere Version der Bibliothek umlenken oder den Platzhalter auf eine andere Bibliothek auflösen.

Aufbau des Dialogs

Der Dialog **Platzhalter** ist in drei Spalten aufgeteilt.

Name	Name des Platzhalters
Bibliothek	Name der Bibliothek, auf die der Platzhalter aufgelöst wird • Fett-Markierung der Bibliotheksspalte: Die Bibliothek ist im Dialog **Platzhalter** fett markiert, wenn der Platzhalter nicht entsprechend der Standardauflösung aufgelöst wird (i.d.R. „immer neueste“/ „*“). Dies ist dann der Fall, wenn er auf eine anwenderspezifische Auflösung geändert wurde. Die Spalte **Info** steht in diesem Fall auf „Aufgelöst durch Platzhalterumlenkung“. • Änderung der Bibliothek, auf die der Platzhalter aufgelöst werden soll: Per Doppelklick auf ein Feld in der Spalte **Bibliothek** öffnet sich eine Auswahlliste unterhalb der selektierten Zeile, über den Sie die Bibliothek ändern können, auf die der Platzhalter aufgelöst werden soll. Weitere Informationen hierzu finden Sie unter Platzhalterauflösung ändern [▶ 295].
Info	Information über die Art der Platzhalterauflösung • Die Info „Aufgelöst durch Bibliotheksprofil“ (englisch: „Resolved by library profile“) bedeutet, dass ein (Beckhoff-)Platzhalter entsprechend der Vorgabe in der Bibliothek und damit entsprechend der Standardauflösung aufgelöst wird. • Die Info „Aufgelöst durch Standard-Bibliothek“ (englisch: „Resolved by default library“) bedeutet, dass ein Platzhalter entsprechend seiner Standardauflösung aufgelöst wird („immer neueste“/„*“). • Die Info „Aufgelöst durch Platzhalterumlenkung“ (englisch: „Resolved by placeholder redirection“) bedeutet, dass die Auflösung eines Platzhalters in diesem Projekt geändert wurde. Es wird nicht die Standardauflösung verwendet, sondern die, die der Anwender selbst eingestellt hat und die in der SPS-Projektdatei gespeichert wird.

13.5.2 Platzhalterauflösung ändern

Es gibt zwei Möglichkeiten, wie Sie die Bibliothek einstellen können, auf die ein Platzhalter aufgelöst werden soll:

- über den Dialog **Platzhalter**
- über das Fenster **Eigenschaften**

Über den Platzhalter-Dialog können Sie alle im Projekt befindlichen Platzhalter einstellen, d. h. sowohl die Platzhalter, die direkt auf Applikationsebene eingebunden sind, als auch die Platzhalter, die innerhalb von referenzierten Bibliotheken verwendet werden.

Über das Eigenschaftenfenster können Sie nur die auf Applikationsebene vorhandenen Platzhalter einstellen.

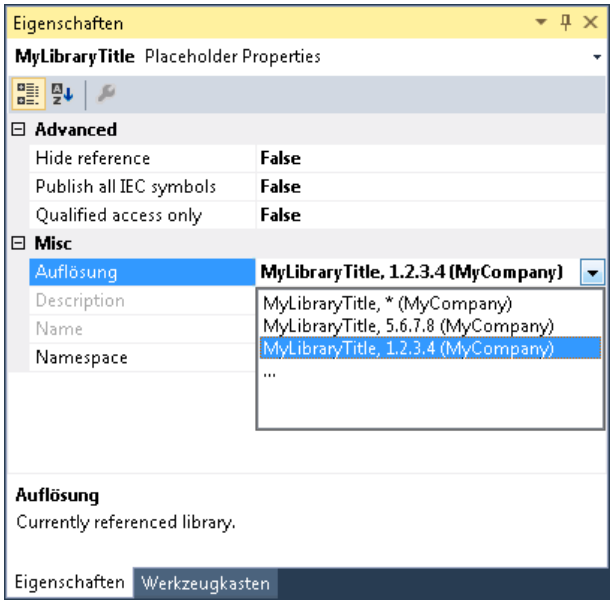
Möglichkeit 1: Über den Dialog Platzhalter

- Öffnen Sie den Dialog Platzhalter [▶ 294].
- Doppelklicken Sie auf das Feld der Spalte **Bibliothek**, das zu der Zeile des Platzhalters gehört, dessen Auflösung Sie ändern möchten.
- Unterhalb der selektierten Zeile öffnet sich ein Dialog, über den Sie die gewünschte Bibliothek auswählen können. Der Dialog stellt Ihnen die folgenden Möglichkeiten zur Verfügung.

Zurücksetzen auf Standard	Dieser Befehl steht nur für Platzhalter zur Verfügung, die aktuell nicht entsprechend ihrer Standardauflösung aufgelöst werden. Dies ist z. B. der Fall, wenn Sie einen Beckhoff-Platzhalter hinzufügen und die Platzhalterauflösung auf eine andere Bibliotheksversion umstellen. Über den Befehl **Zurücksetzen auf Standard** können Sie einen solchen Platzhalter wieder entsprechend der Standardauflösung auflösen. Hierbei wird i.d.R. die gleichnamige Bibliothek mit „immer neuester“/“*“ Version gewählt.
Andere Versionen der bislang ausgewählten Bibliothek	Wenn in diesem Bereich die Bibliotheksversion angezeigt wird, auf die Sie den Platzhalter auflösen möchten, können Sie auf diese Version klicken. Der Platzhalter wird daraufhin auf die gleiche Bibliothek, aber auf eine andere Version dieser Bibliothek aufgelöst.
Andere Bibliothek	Wenn Sie einen Platzhalter auf eine andere Bibliothek auflösen möchten, können Sie über **Andere Bibliothek** einen Dialog öffnen, in dem Sie die gewünschte Bibliothek aus allen installierten Bibliotheken auswählen können. Wenn Sie in diesem Dialog die Option **Alle Versionen anzeigen** aktivieren, können Sie nicht nur die gewünschte Bibliothek festlegen, sondern auch eine spezifische Version dieser Bibliothek auswählen.

Möglichkeit 2: Über das Fenster Eigenschaften

- Öffnen Sie das Fenster Eigenschaften [▶ 377].
- Selektieren Sie einen Platzhalter im Projektbaum unterhalb des References-Knotens.
- Im Eigenschaftenfenster werden die Eigenschaften dieses Platzhalters angezeigt.
- Das Feld **Auflösung** stellt eine Drop-down-Liste bereit, die die verfügbaren Versionen der bislang ausgewählten Bibliothek und den Eintrag „…“ enthält.
  - Wenn in diesem Bereich die Bibliotheksversion angezeigt wird, auf die Sie den Platzhalter auflösen möchten, können Sie auf diese Version klicken. Der Platzhalter wird daraufhin auf die gleiche Bibliothek, aber auf eine andere Version dieser Bibliothek aufgelöst.
  - Wenn Sie den Platzhalter auf eine andere Bibliothek auflösen möchten, können Sie über „…“ einen Dialog öffnen, in dem Sie die gewünschte Bibliothek aus allen installierten Bibliotheken auswählen können. Wenn Sie in diesem Dialog die Option **Alle Versionen anzeigen** aktivieren, können Sie nicht nur die gewünschte Bibliothek festlegen, sondern auch eine spezifische Version dieser Bibliothek auswählen.



13.6 Bibliotheksdokumentation

Mithilfe von Kommentaren können Sie Bibliotheksobjekte und einzelne Elemente von Bibliotheksobjekten dokumentieren, um dem Anwender die Funktion und Verwendungsweise der über die Bibliothek bereitgestellten Programmelemente zu erläutern.

Die über die Kommentare erstellte Dokumentation wird im Bibliotheksverwalter, in dem die Bibliothek eingebunden ist, in den Registerkarten **Ein/Ausgänge** und **Bibliotheksparameter** (Beschreibung einzelner Elemente von Bibliotheksobjekten) sowie in der Registerkarte **Dokumentation** (Generelle Beschreibung von Bibliotheksobjekten) als Text oder als Tabelle angezeigt. (Siehe Grundlagen [▶ 297])

TE1030 | TwinCAT 3 Documentation Generation

Für eine attraktivere Darstellung und einen größeren Funktionsumfang im Bereich der Dokumentation empfehlen wir die Verwendung des Dokumentationstools TE1030 | TwinCAT 3 Documentation Generation.

Unterstützung von reStructuredText bis einschließlich TC3.1 Build 4024

Bis einschließlich Build 4024 steht alternativ das Dokumentationsformat reStructuredText zur Verfügung.

Siehe auch:

- Bibliotheksverwalter [▶ 289]

13.6.1 Grundlagen

TwinCAT bietet die Möglichkeit, Bibliotheksobjekte mithilfe von Kommentaren zu dokumentieren. Dabei können Sie sowohl die Bibliotheksobjekte selbst als auch die einzelnen Elemente der Bibliotheksobjekte kommentieren:

- Generelle Beschreibung von Bibliotheksobjekten [▶ 297]
- Beschreibung einzelner Elemente von Bibliotheksobjekten [▶ 298]

Innerhalb der Kommentare können Sie beschreiben, welchen Zweck das Objekt/Element hat und wie es vom Bibliotheksanwender genutzt werden kann bzw. soll. Die Dokumentation wird im Bibliotheksverwalter, in dem die Bibliothek eingebunden ist, angezeigt. Die Kommentare können einzeilig (Kommentaroperator „//…“) oder mehrzeilig (Kommentaroperator „(*…*)“) sein.

Generelle Beschreibung von Bibliotheksobjekten

Sie können Bibliotheksobjekte beschreiben, indem Sie einen Kommentar oberhalb der Deklarationszeile des Bibliotheksobjekts hinzufügen. Falls sich oberhalb der Deklarationszeile auch Attribute befinden, können Sie den Kommentar entweder ober- oder unterhalb der Attribute positionieren.

Die generelle Beschreibung von Bibliotheksobjekten wird im Bibliotheksverwalter [▶ 289] in der Registerkarte **Dokumentation** angezeigt.

Folgende Objekte können mithilfe eines generellen Kommentars beschrieben werden:

- POU-Objekte (Programm, Funktionsbaustein, Funktion)
- Schnittstellen
- Methoden, Eigenschaften
- Globale Variablenlisten, Parameterlisten
- DUT-Objekte (Aufzählung, Struktur, Union, Alias)

Beispiele für die Positionierung der Kommentare:

Generelle Beschreibung eines Funktionsbausteins

```
// General FB comment
FUNCTION_BLOCK FB_Lib
VAR
…
```

Generelle Beschreibung einer Globalen Variablenliste

```
// General GVL comment. The comment can be placed over possible attributes.
{attribute 'qualified_only'}
VAR_GLOBAL
    fGlobal1     : REAL;
END_VAR
```

Generelle Beschreibung einer Parameterliste

```
{attribute 'qualified_only'}
// General parameter list comment. The comment can be placed below possible attributes.
VAR_GLOBAL CONSTANT
    cParameter1  : LREAL := 12.34;
END_VAR
```

Erweiterte Bibliotheksdokumentation (reStructuredText)

Wenn Sie in den Projekteigenschaften der Bibliothek das Dokumentationsformat **reStructuredText** auswählen und den Kommentartext zur generellen Beschreibung von Bibliotheksobjekten entsprechend der Syntax von reStructuredText auszeichnen, stehen Ihnen weitere vielfältige Möglichkeiten der Dokumentation zur Verfügung. (Siehe Erweitert – reStructuredText [▶ 302])

Beschreibung einzelner Elemente von Bibliotheksobjekten

Neben den Bibliotheksobjekten selbst, können Sie die einzelnen Elemente des Bibliotheksobjekts separat beschreiben, indem Sie die Elemente jeweils mit einem Kommentar versehen. Die Kommentare einzelner Variablen können Sie entweder in der Zeile über der Variablendeklaration oder in derselben Zeile hinter der Variablendeklaration positionieren. Den Rückgabetyp einer Methode oder Funktion können Sie kommentieren, indem Sie den Kommentar in der Zeile der Methodendeklaration hinter dem Rückgabetyp einfügen.

Mithilfe der Tabelle, die im Bibliotheksverwalter [▶ 289] in den Registerkarten **Eingänge/Ausgänge** und **Dokumentation** abgebildet ist, wird die Dokumentation der einzelnen Elemente eines Bibliotheksobjekts übersichtlich dargestellt. Wenn es sich bei dem Bibliotheksobjekt um eine Parameterliste handelt, ist die Tabelle in den Registerkarten **Bibliotheksparameter** und **Dokumentation** dargestellt.

Die Tabelle stellt die Kommentare der folgenden Elemente im Bibliotheksverwalter dar:

- Kommentare einzelner Variablen
  - Ein-/Ausgänge eines POU-Objekts (Programm, Funktionsbaustein, Funktion)
  - Ein-/Ausgänge einer Methode
  - Globale Variablen (Globale Variablenliste)
  - Globale Konstanten (Globale Variablenliste, Parameterliste)
  - Variablen eines DUT-Objekts (Aufzählungs-, Struktur-, Unionvariablen)
- Kommentar des Rückgabetyps (Methode, Funktion)

Beispiele für die Positionierung der Kommentare:

Beschreibung einzelner Funktionsbausteinvariablen

```
FUNCTION_BLOCK FB_Lib
VAR_INPUT
    // Comment for the first input placed over the declaration
    bInput1  : BOOL;
    bInput2  : BOOL;        // Comment for the second input placed behind the declaration
END_VAR
```

Beschreibung des Rückgabetyps einer Methode

```
// General method comment
METHOD SampleMethod : BOOL  // Comment for the method's return value
VAR_INPUT
    bInput   : BOOL;        // Comment for input variable
END_VAR
```

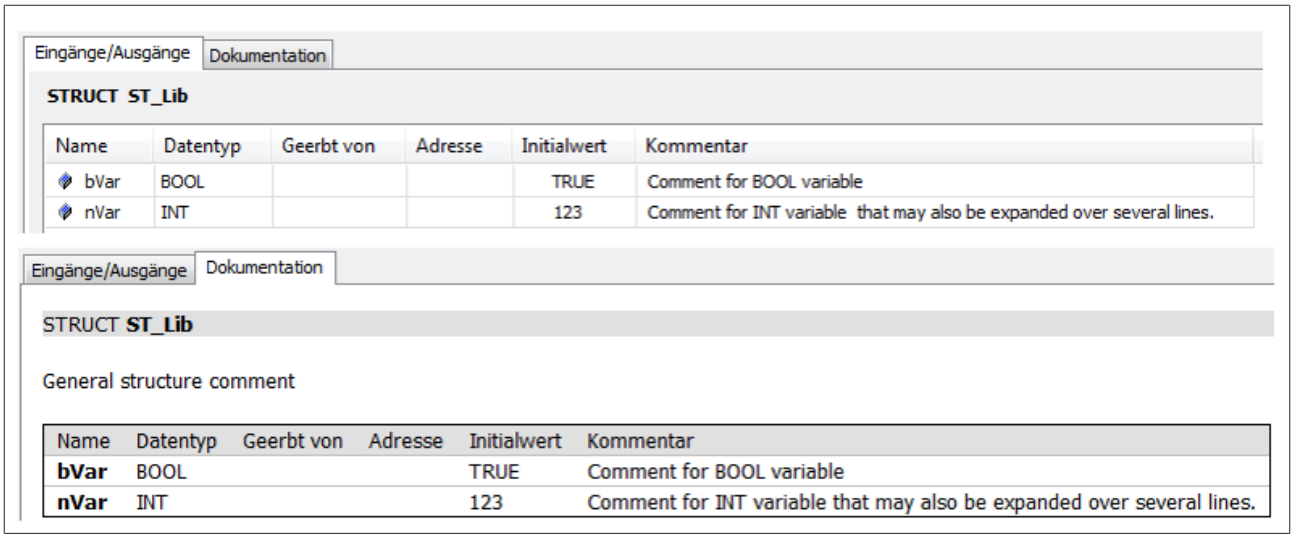
Beispiele

Struktur

Im folgenden Beispiel sind sowohl die Struktur selbst als auch die einzelnen Variablen der Struktur mit einem Kommentar versehen. Die Kommentare werden im Bibliotheksverwalter angezeigt, wenn die Struktur im unteren Teil des Bibliotheksverwalters fokussiert ist.

```
// General structure comment
TYPE ST_Lib :
STRUCT
```

```
    bVar    : BOOL := TRUE;  // Comment for BOOL variable
    nVar    : INT  := 123;   (* Comment for INT variable that may also be
                                expanded over several lines. *)
END_STRUCT
END_TYPE
```

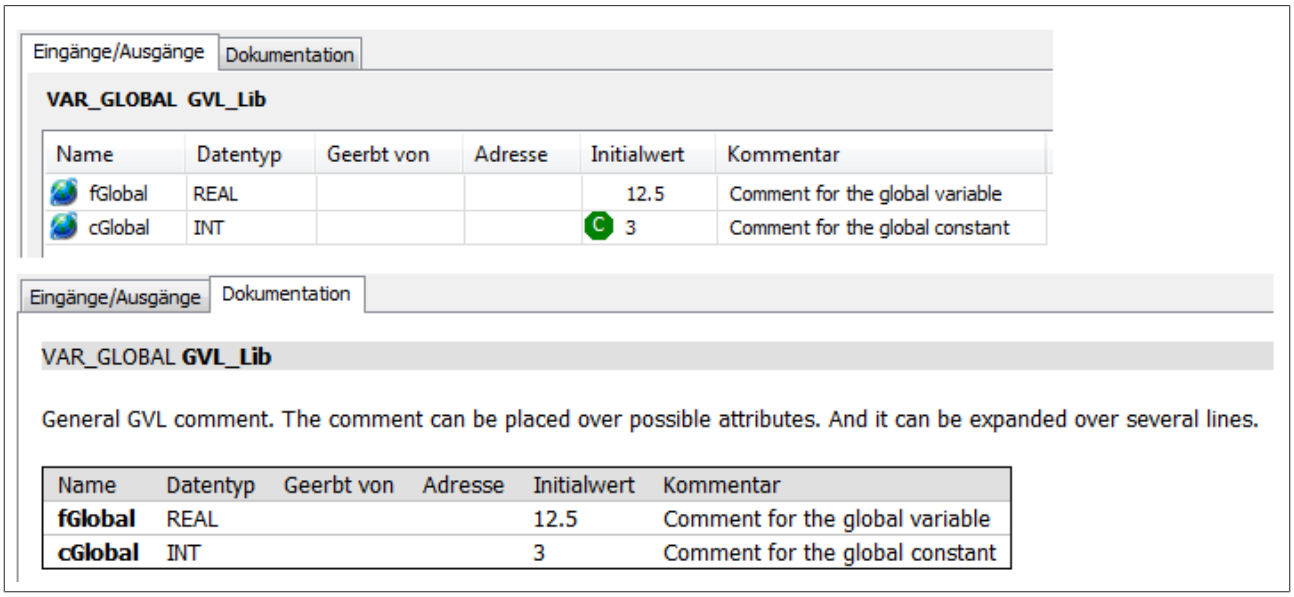


Globale Variablenliste (GVL)

Im folgenden Beispiel sind sowohl die GVL selbst als auch die einzelnen Variablen der GVL mit einem Kommentar versehen. Die Kommentare werden im Bibliotheksverwalter angezeigt, wenn die GVL im unteren Teil des Bibliotheksverwalters fokussiert ist.

Der Kommentar der GVL selbst befindet sich in diesem Beispiel oberhalb des verwendeten Attributs. Alternativ kann der Kommentar auch unterhalb des Attributs positioniert werden.

```
(* General GVL comment.
The comment can be placed over possible attributes.
And it can be expanded over several lines. *)
{attribute 'qualified_only'}
VAR_GLOBAL
    fGlobal    : REAL := 12.5;  // Comment for the global variable
END_VAR
VAR_GLOBAL CONSTANT
    cGlobal    : INT := 3;      // Comment for the global constant
END_VAR
```



Funktionsbaustein mit Methode

Im folgenden Beispiel verfügt ein Funktionsbaustein über Ein-/Ausgangsvariablen sowie über eine Methode. Der Funktionsbaustein und die Methode selbst, die einzelnen Ein-/Ausgangsvariablen des Funktionsbausteins und der Methode sowie der Rückgabetyp der Methode sind mit einem Kommentar versehen. Die Kommentare werden im Bibliotheksverwalter angezeigt, wenn der Funktionsbaustein oder die Methode im unteren Teil des Bibliotheksverwalters fokussiert ist.

Funktionsbaustein:

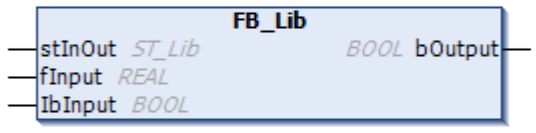
```
// General FB comment
FUNCTION_BLOCK FB_Lib
VAR_IN_OUT
    // Comment for the in-out-variable
    stInOut             : ST_Lib;
END_VAR
VAR_INPUT
    // Comment for this REAL input
    fInput              : REAL := 15.7;
    IbInput    AT%I*    : BOOL;           // Comment for this allocated input
END_VAR
VAR_OUTPUT
    bOutput             :  BOOL := TRUE;  // Comment for this output
END_VAR
VAR
END_VAR
```

Eingänge/Ausgänge | Graphisch | Dokumentation

FUNCTION_BLOCK FB_Lib

Name	Datentyp	Geerbt von	Adresse	Initialwert	Kommentar
stInOut	ST_Lib				Comment for the in-out-variable
fInput	REAL			15.7	Comment for this REAL input
IbInput	BOOL		%I*		Comment for this allocated input
bOutput	BOOL			TRUE	Comment for this output

Eingänge/Ausgänge | Graphisch | Dokumentation



Eingänge/Ausgänge | Graphisch | Dokumentation

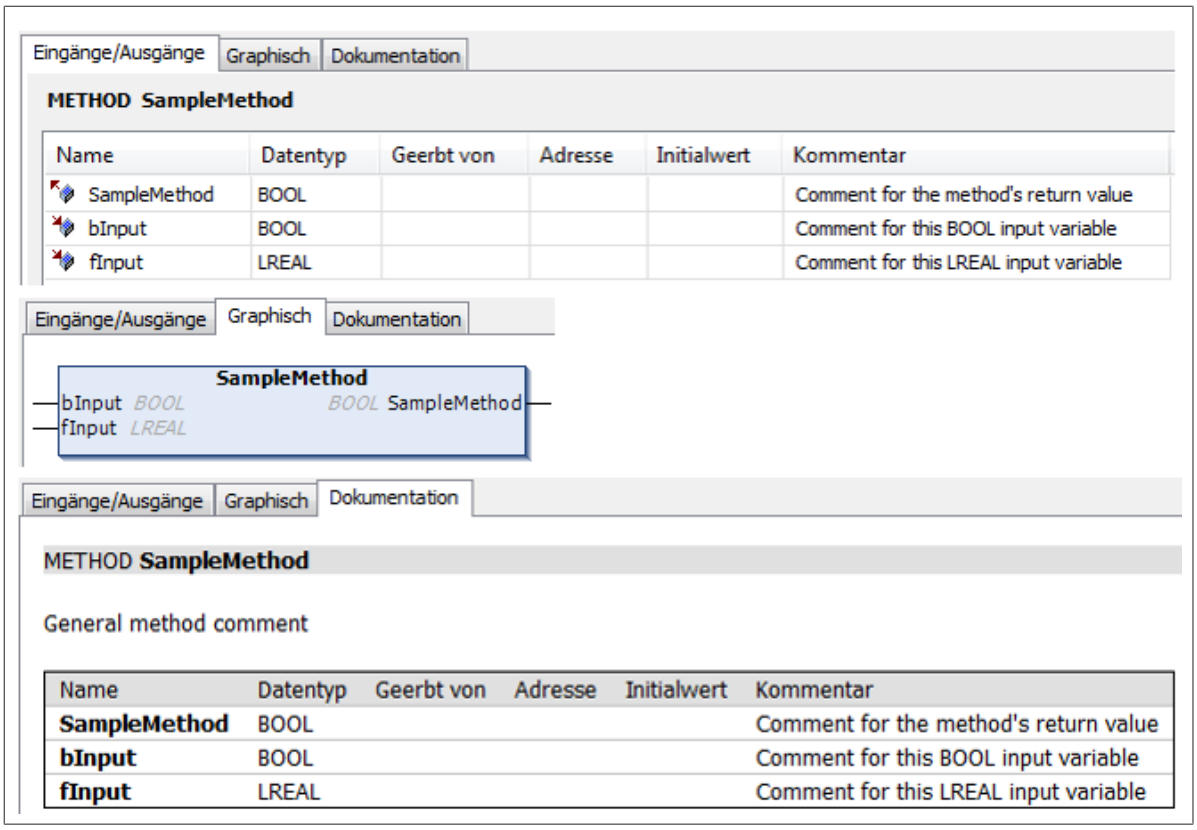
FUNCTION_BLOCK **FB_Lib**

General FB comment

Name	Datentyp	Geerbt von	Adresse	Initialwert	Kommentar
stInOut	ST_Lib				Comment for the in-out-variable
fInput	REAL			15.7	Comment for this REAL input
IbInput	BOOL		%I*		Comment for this allocated input
bOutput	BOOL			TRUE	Comment for this output

Methode:

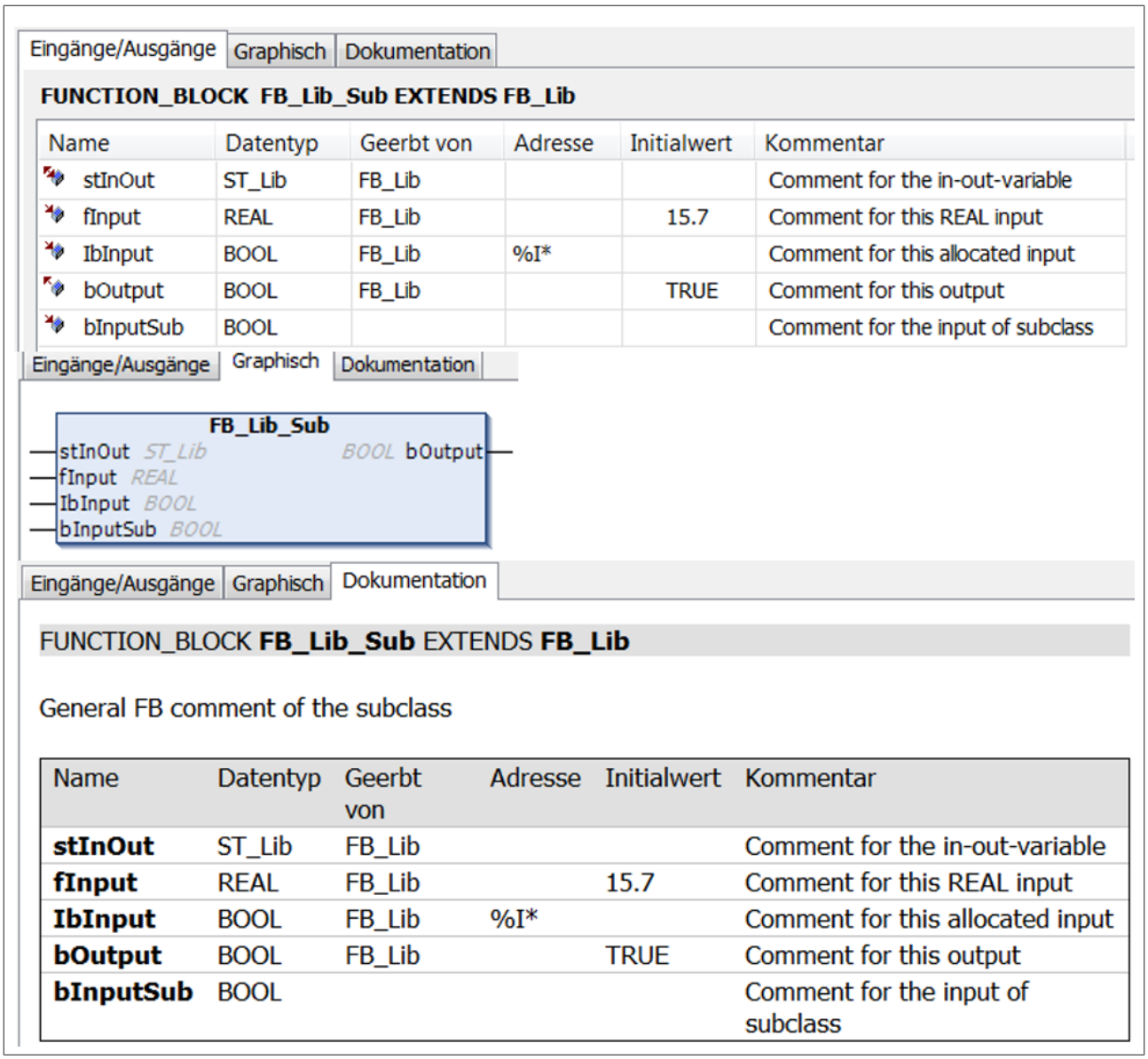
```
// General method comment
METHOD SampleMethod : BOOL   // Comment for the method's return value
VAR_INPUT
    bInput      : BOOL;      // Comment for this BOOL input variable
    fInput      : LREAL;     // Comment for this LREAL input variable
END_VAR
```



Funktionsbaustein als Unterklasse

Im folgenden Beispiel ist ein Funktionsbaustein als Unterklasse des oben beschriebenen Funktionsbausteins deklariert. Die Unterklasse selbst sowie die zusätzliche Eingangsvariable sind mit einem Kommentar versehen. Diese Kommentare werden im Bibliotheksverwalter angezeigt, wenn die Unterklasse im unteren Teil des Bibliotheksverwalters fokussiert ist. Zusätzlich werden für die Unterklasse die Variablen der Oberklasse samt Kommentar angezeigt.

```
// General FB comment of the subclass
FUNCTION_BLOCK FB_Lib_Sub EXTENDS FB_Lib
VAR_INPUT
    bInputSub  : BOOL;        // Comment for the input of subclass
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```



13.6.2 Erweitert – reStructuredText

TE1030 | TwinCAT 3 Documentation Generation

Für eine attraktivere Darstellung und einen größeren Funktionsumfang im Bereich der Dokumentation empfehlen wir die Verwendung des Dokumentationstools TE1030 | TwinCAT 3 Documentation Generation.

Unterstützung von reStructuredText bis einschließlich TC3.1 Build 4024

Bis einschließlich Build 4024 steht alternativ das Dokumentationsformat reStructuredText zur Verfügung.

Für eine attraktivere Darstellung der Dokumentation von Bibliotheksobjekten im Bibliotheksverwalter können Sie in den Projekteigenschaften der Bibliothek das Dokumentationsformat **reStructuredText** auswählen (SPS-Projekteigenschaften [▶ 947] > Kategorie Common [▶ 948]).

reStructuredText ist eine einfach zu lesende Auszeichnungssprache, die einfache Konstrukte verwendet, um die Struktur eines Dokuments und spezielle Textelemente wie zum Beispiel Abschnittsüberschriften, Aufzählungslisten und Hervorhebungen zu kennzeichnen. Explizitere Konstrukte werden verwendet, um Grafiken und Hinweise einzubinden oder um Textelementen eine Funktion zuzuweisen (Hyperlinks).

Textelemente, die Sie im Kommentar oberhalb der Deklarationszeile eines Bibliotheksobjekts entsprechend der Syntax von reStructuredText auszeichnen, werden bei der Bibliothekserstellung im Bibliotheksverwalter in der Registerkarte **Dokumentation** strukturiert und formatiert dargestellt und optional mit einer Funktion versehen.

Der Kommentar oberhalb der Deklarationszeile folgender Objekte wird interpretiert:

- POU-Objekte (Programm, Funktionsbaustein, Funktion)
- Schnittstellen
- Methoden, Eigenschaften
- Globale Variablenlisten, Parameterlisten
- DUT-Objekte (Aufzählung, Struktur, Union, Alias)

Siehe auch:

- Bibliothekserstellung [▶ 282]
- Bibliotheksverwalter [▶ 289]

13.6.2.1 Übersicht

Kommentare bestehen in reStructuredText aus verschiedenen (verschachtelten) Textkörperelementen und können in Abschnitte mit Überschriften gegliedert werden.

Beginnend mit einem einführenden Beispiel wird in den nachfolgenden Abschnitten ein Überblick über die Syntax des reStructuredText-Markups gegeben.

Im ersten Teil der Beschreibung werden dazu die wesentlichen Konzepte und die Syntax von reStructuredText vorgestellt. Im zweiten eher anwendungsorientierten Teil der Beschreibung werden verschiedene Möglichkeiten der Kommentargestaltung unter Nutzung verschiedener Syntaxelemente erläutert.

Die richtige Verwendung der Syntax ist Voraussetzung für eine fehlerfreie Darstellung im Bibliotheksverwalter.

Die dargestellten Informationen basieren auf den Inhalten der Dokutils-Dokumentation (reStructuredText Markup Spezifikation).

Beispiel	• Beispiele [▶ 305]
Allgemeine Hinweise (Syntaxelemente [▶ 311])	• Leerzeilen und Einrückung [▶ 312] • Einfache und komplexere Markup [▶ 315] • Explizite Markup-Blöcke [▶ 316] • Direktiven [▶ 316] • Inline-Markup [▶ 317] • Escaping-Mechanismus [▶ 319] • Referenznamen [▶ 319]
Kommentarstruktur (Kommentarstruktur [▶ 321])	• Abschnitte [▶ 321] • Übergänge [▶ 323]
Textblöcke (Textblöcke [▶ 324])	• Textblock (Absatz) [▶ 324] • Eingerückter Textblock (Blockzitat) [▶ 325] • Zeilenorientierter Textblock (Zeilenblock) [▶ 326]
Listen (Listen [▶ 328])	• Ungeordnete Aufzählungsliste [▶ 328] • Geordnete (nummerierte) Aufzählungsliste [▶ 329] • Definitionsliste [▶ 331] • Feldliste [▶ 332]
Tabellen (Tabellen [▶ 333])	• Einfache Tabelle [▶ 334] • Gittertabelle [▶ 336] • CSV-Tabelle [▶ 337] • Listentabelle [▶ 338]
Hyperlinks (Hyperlinks [▶ 339])	• Interne Hyperlinks [▶ 341] • Externe Hyperlinks [▶ 343] ◦ Alleinstehende Hyperlinks [▶ 345] ◦ Eingebettete URIs und Aliases [▶ 345] • Indirekte Hyperlinks [▶ 346] • Inline-Hyperlinks [▶ 348] • Anonyme Hyperlinks [▶ 349] • Fußnoten [▶ 350] • Zitate [▶ 358]
Substitution	• Substitution [▶ 360]
Hinweiselemente (Hinweiselemente [▶ 362])	• Spezifische Hinweise [▶ 362] • Generische Hinweise [▶ 363]
Bilder	• Bilder [▶ 364]
Codeblock	• Codeblock [▶ 370]
Interne Kommentare	• Interne Kommentare [▶ 371]
Schriftstil (Schriftstil [▶ 372])	• Hervorgehobener Text (kursiv) [▶ 372] • Stark hervorgehobener Text (fett) [▶ 372] • Inline-Literale (Konstantschrift) [▶ 373]

13.6.2.2 Beispiele

13.6.2.2.1 TwinCAT 3 Beispielprojekt

Über diesen https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644044171.zip können Sie ein TwinCAT 3 Beispielprojekt herunterladen, das die Codebeispiele dieser Dokumentation jeweils als eigenständige Funktionsbausteine enthält.

Speichern Sie das SPS-Projekt als Bibliothek und binden Sie diese Bibliothek in ein neues Projekt ein, um die Darstellung der Dokumentation im Bibliotheksverwalter mit dem ursprünglichen Code zu vergleichen. (Siehe Bibliothekserstellung [▶ 282])

Der Ordner „LibPOUs“ des Beispielprojekts teilt sich wie folgt auf:

TC3-Beispielprojekt		Dokumentationsartikel
Ordner	**Name des Funktionsbausteins**	
A_Samples		
	FB_DocuSample_FunctionBlock	Dokumentation eines Funktionsbausteins [▶ 311]
	FB_DocuSample_SyntaxReminder	reStructuredText Syntaxreminder [▶ 307]
B_DocuElements		
Code block	FB_Libdoc_CodeBlock	Codeblock [▶ 370]
Comment structure	FB_Libdoc_Sections	Abschnitte [▶ 321]
	FB_Libdoc_Transitions	Übergänge [▶ 323]
Font style	FB_Libdoc_FontStyle	Schriftstil [▶ 372]
Hyperlinks	FB_Libdoc_AnonymousHyperlinks	Anonyme Hyperlinks [▶ 349]
	FB_Libdoc_Citation	Zitate [▶ 358]
	FB_Libdoc_ExternalHyperlinks	Externe Hyperlinks [▶ 343] Alleinstehende Hyperlinks [▶ 345] Eingebettete URIs und Aliases [▶ 345]
	FB_Libdoc_IndirectHyperlinks	Indirekte Hyperlinks [▶ 346]
	FB_Libdoc_InlineHyperlinks	Inline-Hyperlinks [▶ 348]
	FB_Libdoc_InternalHyperlinks	Interne Hyperlinks [▶ 341]
	FB_Libdoc_LinkToAnotherObject	Link auf ein anderes Objekt [▶ 359]
Hyperlinks\Footnotes	FB_Libdoc_AutomaticallyNumberedFootnotes	Automatisch nummerierte Fußnoten [▶ 352]
	FB_Libdoc_AutomaticallySymbolFootnotes	Automatische Generierung von Symbolen für Fußnoten [▶ 356]
	FB_Libdoc_ManuallyAndAutomaticallyNumberedFootnotes	Manuell und automatisch nummerierte Fußnoten [▶ 357]
	FB_Libdoc_ManuallyNumberedFootnotes	Manuell nummerierte Fußnoten [▶ 350]
	FB_Libdoc_NamedAutomaticallyNumberedFootnotes	Benannte automatisch nummerierte Fußnoten [▶ 354]
Images	FB_Libdoc_Images	Bilder [▶ 364]
Internal comments	FB_Libdoc_InternalComments	Interne Kommentare [▶ 371]
Lists	FB_Libdoc_DefinitionList	Definitionsliste [▶ 331]
	FB_Libdoc_FieldList	Feldliste [▶ 332]
	FB_Libdoc_OrderedNumberedEnumerationList	Geordnete (nummerierte) Aufzählungsliste [▶ 329]
	FB_Libdoc_UnorderedEnumerationList	Ungeordnete Aufzählungsliste [▶ 328]
Note elements	FB_Libdoc_GenericNotes	Generische Hinweise [▶ 363]
	FB_Libdoc_SpecificNotes	Spezifische Hinweise [▶ 362]
Substitution	FB_Libdoc_Substitution_Images	Substitution [▶ 360]
	FB_Libdoc_Substitution_ReplacementText	Substitution [▶ 360]
Tables	FB_Libdoc_CSVTable	CSV-Tabelle [▶ 337]
	FB_Libdoc_GridTable	Gittertabelle [▶ 336]
	FB_Libdoc_ListTable	Listentabelle [▶ 338]
	FB_Libdoc_SimpleTables	Einfache Tabelle [▶ 334]

TC3-Beispielprojekt		
Ordner	**Name des Funktionsbausteins**	**Dokumentationsartikel**
Text blocks	FB_Libdoc_BlockQuote	Eingerückter Textblock (Blockzitat) [▶ 325]
	FB_Libdoc_LineBlock	Zeilenorientierter Textblock (Zeilenblock) [▶ 326]
	FB_Libdoc_Paragraph	Textblock (Absatz) [▶ 324]

Dokumente hierzu

TC3_reStrText_Sample.zip (Resources/zip/9007206028223627.zip)

13.6.2.2.2 reStructuredText Syntaxreminder

Das nachfolgende Beispiel gibt einen Überblick über die Verwendung der reStructuredText-Syntax im Kommentar eines Bibliotheksobjekts und die zugehörige Darstellung im Bibliotheksverwalter.
(Im Beispielprojekt [▶ 305]: A_Samples\FB_DocuSample_SyntaxReminder)

Code

```
(*
================================================
The reStructuredText Cheat Sheet: Syntax Reminder
================================================
:Info: See <https://infosys.beckhoff.de/.
:Author: Beckhoff <support@beckhoff.com>
:Date: 2017-12-14

.. NOTE:: This syntax reminder is based on the docutils documentation

Section Structure
=================

A reStructuredText comment is made up of body elements, and may be
structured into sections. Section titles are underlined or overlined and underlined.
Sections contain body elements and/or subsections.

Paragraphs are flush-left and separated by blank lines.

Body Elements
=============
Grid table:

+------------------------------+--------------------------------------+
|Block quotes consist of       |Literal block, preceded by "::"::     |
|indented body elements:       |                                      |
|                              |    If (a=TRUE) Then                  |
|    Block quotes are indented.|      b=3                             |
|                              |                                      |
+------------------------------+--------------------------------------+
|| Line blocks preserve line breaks and indents.                       |
||    Useful for adornment-free lists;                                 |
||    long lines can be wrapped                                        |
|     with continuation lines.                                        |
+---------------------------------------------------------------------+

Simple table:

===============  ============================================================
List Type        Examples
===============  ============================================================
Bullet list      - This is a bullet list
                 - Items begin with "-", "+", or "*"
Enumerated list  1. This is an enumerated list.
                 2. Items use any variation of "1.", "A)", and "(i)"
                 #. Item can also be auto-enumerated
Definition list  Term is flush-left : optional classifier
                     Definition is indented, no blank line between
                     continue
Field list       :Field name: field body
===============  ============================================================
```

```
Inline Markup
=============
*emphasis*, **strong emphasis**, ``inline literal text``

| Standalone hyperlink: http://beckhoff.de
| Named reference: Beckhoff_
| Anonymous reference: `Anonymous reference`__
| Footnote reference: [1]_
| Citation reference: [CIT2002]_
| Substitution: |substitution|
| Inline internal target: `Inline internal target`_

Explicit Markup
===============

===============  ========================================================
Explicit Markup       Examples (visible in the source code)
===============  ========================================================
Footnote         .. [1] Manually numbered or [#] auto-numbered
                        (even [#label]) or [*] auto-symbol
Citation         .. [CIT2002] This is a citation.
Hyperlink Target .. _Beckhoff: http://beckhoff.de
                 .. _indirect target: Beckhoff_
                 .. _internal target:
Anonymous Target __ http://beckhoff.de
Directive ("::") .. image::, .. code::
Substitution Def .. |substitution| replace:: This is a substitution
Comment          .. This text will not be shown
===============  ========================================================

Directives
==========

===============  ========================================================
Directive Name   Description
===============  ========================================================
attention        Specific admonition ("caution", "danger",
                 "error", "hint", "important", "note", "tip", "warning")
admonition       Generic titled admonition
image            .. image:: C:\Tc3LibDocImages\SampleLib1\logo.gif
                    :width: 40
code             .. code::

                    if(a=true) then b=3;
list-table       Create a table from a uniform two-level bullet list
csv-table        Create a table from CSV data
===============  ========================================================
*)

FUNCTION_BLOCK FB_DocuSample_SyntaxReminder
VAR_INPUT
    nVarInA : INT;        //First input variable
    nVarInB : INT := 5;   //Second input variable
END_VAR
VAR_OUTPUT
    nVarOut : INT;        //Output variable
END_VAR
```

Darstellung im Bibliotheksverwalter

Inputs/Outputs | Graphical | Documentation

FB_DocuSample_SyntaxReminder (FB)

FUNCTION_BLOCK FB_DocuSample_SyntaxReminder

The reStructuredText Cheat Sheet: Syntax Reminder

Info: See <https://infosys.beckhoff.de/.
Author: Beckhoff <support@beckhoff.com>
Date: 2017-12-14

Note

This syntax reminder is based on the docutils documentation

Section Structure

A reStructuredText comment is made up of body elements, and may be structured into sections. Section titles are underlined or overlined and underlined. Sections contain body elements and/or subsections.

Paragraphs are flush-left and separated by blank lines.

Body Elements

Grid table:

Block quotes consist of indented body elements:
 	
Block quotes are indented.
	Literal block, preceded by "::":
	
	`If (a=TRUE) Then`
	`  b=3`

Line blocks preserve line breaks and indents. Useful for adornment-free lists; long lines can be wrapped with continuation lines.	

Simple table:

List Type	Examples
Bullet list	▪ This is a bullet list ▪ Items begin with "-", "+", or "*"
Enumerated list	1. This is an enumerated list. 2. Items use any variation of "1.", "A)", and "(i)" 3. Item can also be auto-enumerated
Definition list	Term is flush-left : *optional classifier* Definition is indented, no blank line between continue
Field list	**Field name:** field body

Inline Markup

emphasis, **strong emphasis**, `inline literal text`

Standalone hyperlink: http://beckhoff.de
Named reference: Beckhoff
Anonymous reference: Anonymous reference
Footnote reference: [1]
Citation reference: [CIT2002]
Substitution: This is a substitution
Inline internal target: `Inline internal target`_

Explicit Markup

Explicit Markup	Examples (visible in the source code)
Footnote	[1] Manually numbered or [#] auto-numbered (even [#label]) or [*] auto-symbol
Citation	[CIT2002] This is a citation.
Hyperlink Target	

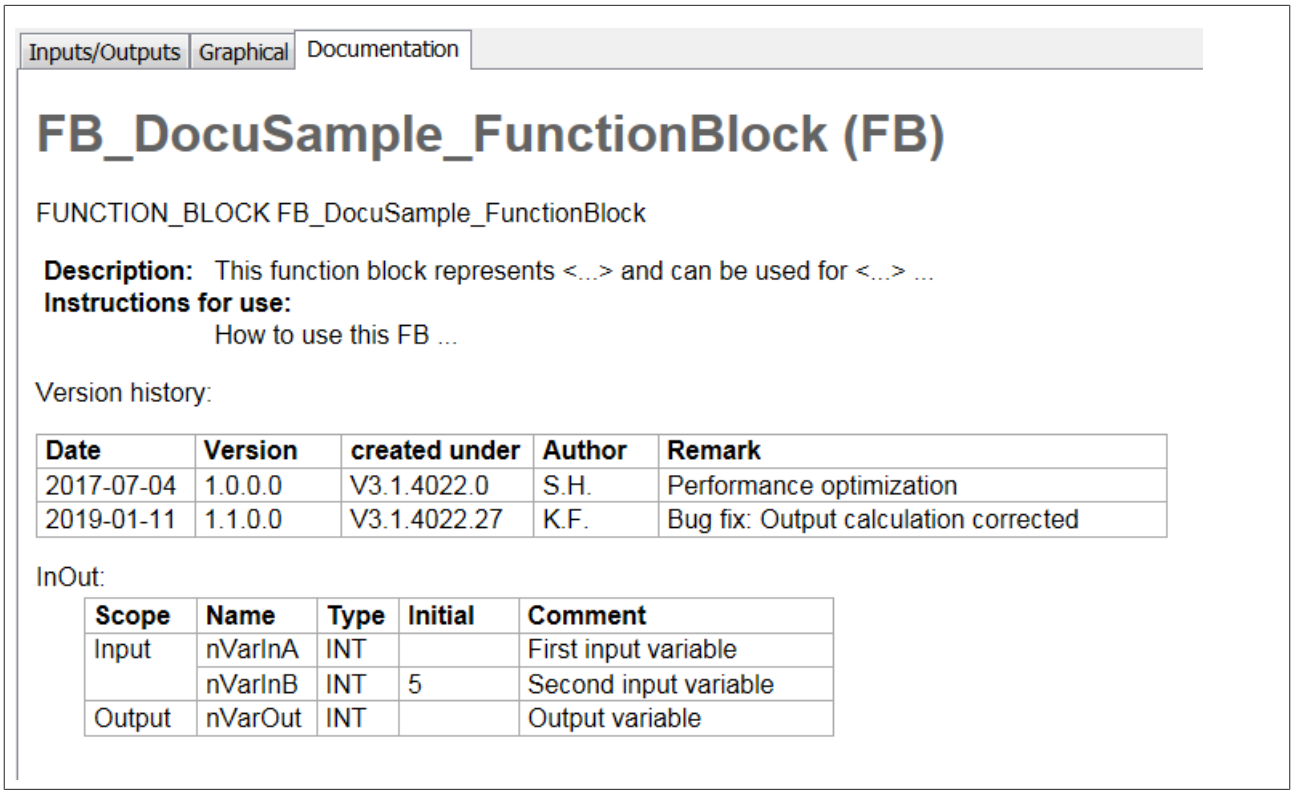
13.6.2.2.3 Dokumentation eines Funktionsbausteins

Das nachfolgende Beispiel zeigt die Dokumentation eines Funktionsbausteins unter Verwendung der reStructuredText-Syntax im Kommentar des Funktionsbausteins und die zugehörige Darstellung im Bibliotheksverwalter.
(Im Beispielprojekt [▶ 305]: A_Samples\FB_DocuSample_FunctionBlock)

Code

```
(*
:Description: This function block represents <...> and can be used for <...> ...
:Instructions for use: How to use this FB ...

Version history:

+-------------+------------+----------------+----------+-----------------------------------------+
|Date         | Version    | created under  | Author   | Remark                                  |
+=============+============+================+==========+=========================================+
|2017-07-04   | 1.0.0.0    | V3.1.4022.0    | S.H.     | Performance optimization                |
+-------------+------------+----------------+----------+-----------------------------------------+
|2019-01-11   | 1.1.0.0    | V3.1.4022.27   | K.F.     | Bug fix: Output calculation corrected   |
+-------------+------------+----------------+----------+-----------------------------------------+
*)

FUNCTION_BLOCK FB_Libdoc_FontStyle
VAR_INPUT
    nVarInA : INT;       //First input variable
    nVarInB : INT :=5    //Second input variable
END_VAR
VAR_OUTPUT
    nVarOut : INT;       //Output variable
END_VAR
```

Darstellung im Bibliotheksverwalter



13.6.2.3 Syntaxelemente

Zur Auszeichnung von Wörtern, Phrasen und Absätzen werden in reStructuredText verschiedene Syntaxelemente und Konstrukte verwendet:

- Leerzeilen und Einrückung [▶ 312]

- Einfache und komplexe Markup [▶ 315]
- Explizite Markup-Blöcke [▶ 316]
  - Direktiven [▶ 316]
- Inline-Markup [▶ 317]

Um die Standardbedeutung der für die Markup verwendeten Zeichen zu erhalten, gibt es in reStructuredText einen Escaping-Mechnismus:

- Escaping-Mechanismus [▶ 319]

An die Bezeichner von Hyperlinks (Referenznamen) werden bestimmte Anforderungen gestellt:

- Referenznamen [▶ 319]

13.6.2.3.1 Leerzeilen und Einrückung

Die Trennung und Verschachtelung von Textkörperelementen erfolgt in reStructuredText über folgende Gestaltungselemente:

- Leerzeilen [▶ 312]
- Absatz- und Zeilenwechsel [▶ 312]
- Einrückung [▶ 312]

Leerzeilen

- Leerzeilen werden in einem reStructuredText zur Trennung von Absätzen und anderen Textkörperelementen verwendet (siehe Beispiele [▶ 312]).
- Mehrere aufeinanderfolgende Leerzeilen entsprechen einer einzelnen Leerzeile. (Ausnahme: In einem Codeblock [▶ 370] bleiben alle Leerzeilen erhalten.)
- Leerzeilen können weggelassen werden, wenn ein Markup oder eine Einrückung die Trennung von Textkörperelementen eindeutig macht.
- Die erste Zeile im Kommentar wird so behandelt, als ob ihr eine Leerzeile vorangestellt wäre, und die letzte Zeile im Kommentar wird so behandelt, als würde ihr eine Leerzeile folgen.

Absatz- und Zeilenwechsel

- Absatzwechsel werden durch Leerzeilen gekennzeichnet (siehe Beispiele [▶ 312]).
- Innerhalb von Absätzen [▶ 324] wird der Text fortlaufend dargestellt und automatisch umgebrochen, wenn die Fensterbreite des Bibliotheksverwalters erreicht wird. Einfache Zeilenumbrüche im Kommentar bewirken keinen Zeilenumbruch in der Darstellung. Um Zeilenwechsel zu realisieren, werden zeilenorientierte Textblöcke (Zeilenblöcke) [▶ 326] verwendet.

Einrückung

- Einrückungen werden verwendet, um verschachtelte Inhalte zu kennzeichnen (siehe Beispiele [▶ 312]).
- Jede Textzeile, dessen Einrückung kleiner ist als die der aktuellen Ebene, beendet die aktuelle Ebene der Einrückung.
- Für die Einrückung können Leerzeichen oder Tabulatoren verwendet werden. Da alle Einrückungen signifikant sind, muss der Grad der Einrückung konsistent sein.

Beispiele

Textblock (Absatz)

Wenn ein Absatz oder ein anderes Konstrukt aus mehr als einer Textzeile besteht, müssen die Zeilen linksbündig ausgerichtet sein.

```
This is a paragraph. The lines of
this paragraph are aligned at the left.

    This paragraph has problems. The
lines are not left-aligned.
```

This is a paragraph. The lines of this paragraph are aligned at the left.

This paragraph has problems. The

lines are not left-aligned.

Siehe auch: Textblock (Absatz) [▶ 324]

Eingerückter Textblock (Blockzitat)

Die Einrückung ist die einzige Kennzeichnung für Blockzitate (eingerückter Textblock).

- Eine Leerzeile zwischen einem Textblock (Absatz oder Blockzitat) und einem nachfolgenden eingerückten Textblock (Blockzitat) ist optional. Durch die Einrückung des nachfolgenden Textblocks erfolgt am Ende der vorherigen Textblocks ein Umbruch. Wenn eine Leerzeile im Kommentar eingefügt wird, erfolgt der Umbruch und die Leerzeile bleibt in der Darstellung erhalten.
- Zwischen einem eingerückten Textblock (Blockzitat) und einem nachfolgenden nicht eingerückten Textblock (Absatz oder Blockzitat) wird automatisch, unabhängig davon, ob im Kommentar eine Leerzeile eingefügt ist oder nicht, eine Leerzeile dargestellt. Die aktuelle Ebene der Einrückung wird beendet.
- Zwischen zwei Textblöcken (Blockzitate) auf einer Einrückungsebene muss im Kommentar eine Leerzeile eingefügt werden, um die Textblöcke voneinander zu trennen.

```
This is a top-level paragraph.

    This paragraph belongs to a first-level block quote.

    This is the second paragraph of the first-level block quote.

        This paragraph belongs to a second-level block quote.

Another top-level paragraph.

        This paragraph belongs to a second-level block quote.

    This paragraph belongs to a first-level block quote. The
    second-level block quote above is inside this first-level
    block quote.
```

This is a top-level paragraph.

This paragraph belongs to a first-level block quote.

This is the second paragraph of the first-level block quote.

This paragraph belongs to a second-level block quote.

Another top-level paragraph.

This paragraph belongs to a second-level block quote.

This paragraph belongs to a first-level block quote. The second-level block quote above is inside this first-level block quote.

Siehe auch: Eingerückter Textblock (Blockzitat) [▶ 325]

Zeilenorientierter Textblock (Zeilenblock)

Mit einem Zeilenblock kann ein Zeilenwechsel realisiert werden. Durch Einrückungen können einzelne Zeilen eingerückt werden.

```
| Each new line begins with
| a vertical bar ("|").
|     Line breaks and initial indents
|     are preserved.
| Continuation lines are wrapped
  portions of long lines; they begin
  with spaces in place of vertical bars.
```

Each new line begins with
a vertical bar ("|").
    Line breaks and initial indents
    are preserved.
Continuation lines are wrapped portions of long lines; they begin with spaces in place of vertical bars.

Siehe auch: Zeilenorientierter Textblock (Zeilenblock) [▶ 326]

Einfache und komplexe Markup

Mehrere Konstrukte beginnen mit einem Markup. Der Körper des Konstrukts muss dann relativ zum Markup eingerückt werden.

Bei Konstrukten mit einfachen Markup (ungeordnete und geordnete Aufzählungslisten, Fußnoten, Zitate, Hyperlink-Ziele, Direktiven und Kommentare) wird der Grad der Einrückung des Körpers durch die Position der ersten Textzeile bestimmt, die auf derselben Zeile beginnt wie das Markup.

```
- This is the first line of a bullet list
  item's paragraph. All lines must align
  relative to the first line. [1]_

    This indented paragraph is interpreted
    as a block quote.

Because it is not sufficiently indented,
this paragraph does not belong to the list
item.

.. [1] Here's a footnote. The second line is aligned
   with the beginning of the footnote label. The ".."
   marker is what determines the indentation.
```

- This is the first line of a bullet list item's paragraph. All lines must align relative to the first line. [1]

  This indented paragraph is interpreted as a block quote.

Because it is not sufficiently indented, this paragraph does not belong to the list item.

[1] Here's a footnote. The second line is aligned with the beginning of the footnote label. The ".." marker is what determines the indentation.

Bei Konstrukten mit komplexeren Markup bestimmt in der Regel die Einrückung der ersten Zeile nach dem Markup den linken Rand des Textkörpers. Wenn das Markup sehr lang ist, kann es sinnvoll sein, mit dem Textkörper in der nächsten Zeile zu beginnen. Die Zeile nach dem Markup muss um mindestens ein Leerzeichen eingerückt sein (minimale Einrückung).

```
:Field: This field has a short field name, so aligning the field
        body with the first line is feasible.

:This field has a long field name: It would
    be very difficult to align the field body with the left edge
    of the first line.

:This field also has a long field name:
    It would be very difficult to align the field body with the left edge
    of the first line. It may even be preferable not to begin the
    body on the same line as the marker.
```

Field: This field has a short field name, so aligning the field body with the first line is feasible.

This field has a long field name:
It would be very difficult to align the field body with the left edge of the first line.

This field also has a long field name:
It would be very difficult to align the field body with the left edge of the first line. It may even be preferable not to begin the body on the same line as the marker.

Siehe auch:

- Listen [▶ 328]
- Explizite Markup-Blöcke [▶ 316]

13.6.2.3.2 Einfache und komplexere Markup

reStructuredText umfasst einfache und komplexere Markup. Im Gegensatz zu den einfachen Markup können komplexere Markup beliebigen Text enthalten.

Ohne Markup:

Verwendung	Markup	Beispiel
Textblöcke (Absätze) [▶ 324]	-	`Paragraph`

Einfache Markups werden für folgende Konstrukte verwendet:

Verwendung	Markup	Beispiel

Eingerückter Textblock (Blockzitat) [▶ 325]	Einfache Einrückung	`Paragraph`
		
		`    Block quote`

Zeilenorientierter Textblock (Zeilenblock) [▶ 326]	Vertikaler Balken	`| Line block`
Ungeordnete Aufzählungsliste [▶ 328]	Aufzählungszeichen	`- Item`
Geordnete Aufzählungsliste [▶ 329]	Zähler	`1. Item`

Einfache Tabelle [▶ 334]	Horizontale Balken	`========`
		`Column 1`
		`========`
		`Line 1`
		`--------`

Gittertabelle [▶ 336]	Horizontale und vertikale Balken	`+----------+`
		`| Column 1 |`
		`+==========+`
		`| Line 1   |`
		`+----------+`

Komplexere Markup werden für folgende Konstrukte verwendet:

Verwendung	Markup	Beispiel
Definitionsliste [▶ 331]	Beliebiger Text gefolgt von einem Doppelpunkt und Einrückung	`Term 3 : classifier` `    Definition 3`
Feldliste [▶ 332]	Beliebiger Text von Doppelpunkten eingeschlossen	`:Organization: Beckhoff Automation GmbH & Co. KG`

13.6.2.3.3 Explizite Markup-Blöcke

Als explizite Markup-Blöcke werden in reStructuredText Textblöcke bezeichnet,

- deren erste Zeile mit zwei Punkten gefolgt von einem Leerzeichen beginnt („expliziter Markup-Start“)
- deren zweite und nachfolgende Zeilen (falls vorhanden) relativ zur ersten eingerückt sind
- die vor einer nicht eingerückten Zeile enden

Prinzip

```
+-----+-----------------+
|".. "|explicit markup  |
+-----+  block          |
      |                 |
      +-----------------+
```

Zwischen expliziten Markup-Blöcken und anderen Textkörperelementen (z. B. einem vorangehenden Absatz mit Text) ist eine Leerzeile notwendig. Eine Leerzeile zwischen expliziten Markup-Blöcken ist optional.

Beispiel:

```
Paragraph

.. [1] Body elements
.. [2] Body elements
```

Verwendung

Explizite Markup-Blöcke werden für folgende Konstrukte verwendet:

Verwendung	Beispiel
Interne Hyperlink-Ziele [▶ 342]	`.. _target:`
Externe Hyperlink-Ziele [▶ 344]	`.. _target: http://www.beckhoff.de`
Indirekte Hyperlink-Ziele [▶ 347]	`.. _target: hyperlink-reference_`
Anonyme Hyperlink-Ziele [▶ 350]	`.. __: Anonymous-hyperlink-target`
Manuell nummerierte Fußnoten [▶ 350]	`.. [1] Body elements`
Automatisch nummerierte Fußnoten [▶ 352]	`.. [#] Body elements`
Automatische Generierung von Symbolen für Fußnoten [▶ 356]	`.. [*] Body elements`
Zitat (Ziel) [▶ 359]	`.. [CIT] Citation`
Substitutionsdefinition [▶ 360]	`.. |ref| type: definition`
Interne Kommentare [▶ 371]	`.. comment`

Explizite Markup-Blöcke mit weiteren Eigenschaften:

- Direktiven [▶ 316]

13.6.2.3.4 Direktiven

Direktiven gehören zu den expliziten Markup-Blöcken. Es handelt sich dabei um Syntaxelemente, die Auswirkungen auf einen ganzen Absatz haben.

Aufbau

- Direktiven bestehen aus einer Direktivenmarkierung gefolgt von einem Direktivenblock.
- Die Direktivenmarkierung setzt sich aus einem expliziten Markup-Start („.. “), dem Typ der Direktive, zwei Doppelpunkten und einem Leerzeichen zusammen.
- Der Direktivenblock besteht aus drei Teilen, wobei einzelne Direktiven beliebige Kombinationen dieser Teile verwenden:

- Direktivenargument
- Direktivenoptionen
- Direktiveninhalt

- Direktivenargumente können Dateisystempfade, Titeltexte usw. sein.
- Die Angabe der Direktivenoptionen erfolgt über Feldlisten, wobei die möglichen Feldnamen und Inhalte vom Typ der Direktive abhängen.
- Direktivenargumente und -optionen müssen einen zusammenhängenden Block bilden, der mit der ersten oder zweiten Zeile des expliziten Markup-Blocks beginnt.
- Der Anfang des Inhaltsblocks der Direktive wird durch eine Leerzeile kennzeichnet.

Prinzip

```
+-----+----------------------------+
|".. "|directive type"::" directive |
+-----+block                       |
      |                            |
      +----------------------------+
```

Zwischen einer Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig. Eine Leerzeile zwischen Direktiven ist optional.

Beispiel:

```
Paragraph

.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
.. image:: C:\Tc3LibDocImages\SampleLib1\img12.jpg
```

Verwendung

Über Direktiven können folgende Textköperelemente erzeugt werden:

Verwendung	Beispiel
Spezifische Hinweise [▶ 362]	`.. note::`
Generische Hinweise [▶ 363]	`.. admonition:: Title`

Bilder [▶ 364]	`.. image:: C:\Users\SampleUser\Documents\LibraryDocumentationpicture.png`
	`:height: 100 px`
	`:width: 200 px`

Codeblock [▶ 370]	`.. code::`
	
	`Code block`

CSV-Tabelle [▶ 337]	`.. csv-table:: Table`
	`:header: "Column 1", "Column 2"`
	`:widths: 50, 50`
	
	`"Line 1.1","Line 1.2"`

Listentabelle [▶ 338]	`.. list-table:: Table`
	`:widths: 50 50`
	`:header-rows: 1`
	
	`* - Column 1`
	`- Column 2`
	`* - Line 1.1`
	`- Line 1.2`

13.6.2.3.5 Inline-Markup

Inline-Markup werden in reStructuredText zur Auszeichnung von Wörtern oder Phrasen innerhalb eines Textblocks verwendet. Wörter und Phrasen können so formatiert oder mit einer Funktion versehen werden.

Inline-Markup bestehen aus einem Start- und Endzeichen, die das auszeichnende Wort oder die Phrase umschließen.

Verwendung

Inline-Markup werden für folgende Konstrukte verwendet:

- Konstrukte mit identischem Start- und Endzeichen:

Verwendung	Beispiel
Hervorgehobener Text (kursiv) [▶ 372]	`*emphasized text*`
Stark hervorgehobener Text (fett) [▶ 372]	`**strong text**`
Inline-Literale (Konstantschrift) [▶ 373]	``` ``inline literals`` ```
Substitutionsreferenzen [▶ 360]	`|substitution reference|`

- Konstrukte mit unterschiedlichem Start- und Endzeichen:

Verwendung	Beispiel

Hyperlink-Referenz (inline, intern, extern, indirekt)
(Hyperlinks [▶ 339])
	Wort:
	`Target_`
	Phrase:
	`` `Hyperlink target`_ ``

Anonyme Hyperlink-Referenz [▶ 349]	Wort:
	`Target__`
	Phrase:
	`` `Hyperlink target`__ ``

Eingebettete URIs und Aliases [▶ 345]	Eingebetteter URI:
	`` `Beckhoff home page <http://www.beckhoff.de>`_ ``
	Alias:
	`` `link <Beckhoff home page_>`_ ``

Alleinstehende Hyperlinks [▶ 345]	`http://www.beckhoff.de` `support@beckhoff.com`

Fußnotenreferenz
(Fußnoten [▶ 350])
	Manuelle Nummerierung:
	`[1]_`
	Automatische Nummerierung:
	`[#]_`
	Automatische Symbolgenerierung:
	`[*]_`

Zitatreferenz [▶ 358]	`[CIT]_`

Inline-Hyperlinks [▶ 348]	Wort:
	`_Inline-hyperlink-target`
	Phrase:
	`` _`Inline target` ``

Regeln für die Erkennung von Inline-Markup

Inline-Markup können nicht verschachtelt werden. Start- und Endzeichen von Inline-Markup werden nur erkannt, wenn die folgenden Bedingungen erfüllt sind:

1. Dem Startzeichen darf kein Leerzeichen folgen
2. Dem Endzeichen darf kein Leerzeichen vorangestellt werden.
3. Das Startzeichen muss einen Textblock beginnen oder ihm muss unmittelbar ein Leerzeichen oder eines der Zeichen - : / ' " < ([{ vorangestellt werden.
4. Das Endzeichen muss einen Textblock beenden oder ihm muss unmittelbar ein Leerzeichen oder eines der Zeichen ' " . , : ; ! ? -)] } / \ > folgen.
5. Das Endzeichen muss durch mindestens ein Zeichen vom Startzeichen getrennt sein.

6. Sowohl dem Start- als auch dem Endzeichen darf kein Backslash vorangestellt werden (außer dem Endzeichen von Inline-Literalen). Ein Backslash, der einem Start- oder Endzeichen vorausgeht, deaktiviert die Markup-Erkennung, mit Ausnahme des Endzeichens von Inline-Literalen.
7. Wenn einem Startzeichen unmittelbar eines der Zeichen ' " ([{ < vorangestellt wird, darf ihm nicht unmittelbar das entsprechende Zeichen ' ")] } > folgen (nicht möglich: `"*"text"*"`, möglich: `(*(text)*)`).

Inline-Markup auf Zeichenebene

Es ist möglich, einzelne Zeichen innerhalb eines Wortes mit einem Backslash zu markieren, damit beliebiger Text sofort nach dem Inline-Markup folgen kann.

```
Python ``list``\s use square bracket syntax.
```

Python `lists` use square bracket syntax.

Der Backslash verschwindet aus dem bearbeiteten Dokument. Das Wort „list“ erscheint in Konstantschrift, und der Buchstabe „s“ folgt sofort als normaler Text, ohne Zwischenraum.

Beliebiger Text kann dem Inline-Markup durch die Verwendung von Backslash und Leerzeichen vorangestellt werden.

```
Possible in *re*\ ``Structured``\ *Text*, though not encouraged
```

Possible in *re*`Structured`*Text*, though not encouraged.

Die Backslashes und Leerzeichen zwischen „re“, „Structured“ und „Text“ verschwinden aus dem bearbeiteten Kommentar.

Die Verwendung von Backslash für Inline-Markierungen auf Zeichenebene wird nicht empfohlen. Eine solche Verwendung erschwert die Lesbarkeit des unverarbeiteten Kommentars. Nutzen Sie diese Funktion sparsam und nur dort, wo es unbedingt notwendig ist.

Siehe auch: Escaping-Mechanismus [▶ 319]

13.6.2.3.6 Escaping-Mechanismus

Der für Klartextdokumente allgemein verfügbare Zeichensatz ist begrenzt. So können einzelne Markup-Zeichen im geschriebenen Text bereits eine Bedeutung haben und im Text erscheinen, ohne dass sie als Markup gedacht sind. Um die Standardbedeutung der für die Markup verwendeten Zeichen zu erhalten, gibt es in reStructuredText einen Escaping-Mechanismus. Als Escape-Zeichen wird ein Backslash („\“) verwendet.

Es gilt:

- Ein Backslash gefolgt von einem beliebigen Zeichen (außer Leerzeichen in Nicht-URI-Kontexten) bewirkt, dass das Zeichen das Zeichen selbst repräsentiert und bei der Interpretation von Markups keine Rolle spielt. Der Backslash wird aus der Ausgabe entfernt.
- Ein buchstäblicher Backslash wird durch zwei Backslashes in einer Reihe dargestellt.
- In Nicht-URI-Kontexten wird ein Backslash gefolgt von einem Leerzeichen aus dem Kommentar entfernt. Dies ermöglicht eine Inline-Markierung auf Zeichenebene (siehe Inline-Markup auf Zeichenebene [▶ 319]).
- Es gibt zwei Kontexte, in denen Backslashes keine besondere Bedeutung haben: Codeblöcke und Inline-Literale. In diesen Kontexten stellt ein einzelner Backslash einen buchstäblichen Backslash dar, ohne dass er verdoppelt werden muss (siehe Codeblock [▶ 370]).

13.6.2.3.7 Referenznamen

In reStructuredText werden einfache Referenznamen und Phrasenreferenzen unterschieden.

Einfache Referenznamen

Einfache Referenznamen sind einzelne Wörter, die aus alphanumerischen Zeichen (Buchstaben oder Ziffern) und isolierten (nicht zwei benachbarten) Bindestrichen, Unterstrichen, Punkten, Doppelpunkten und Pluszeichen bestehen. Leerzeichen oder andere Zeichen sind nicht erlaubt. Einfache Referenznamen werden bei Zitatbeschriftungen und einigen Hyperlinks verwendet:

```
Want to learn about MyFavoriteProgrammingLanguage_?

.. _MyFavoriteProgrammingLanguage: http://www.python.org
```

oder

```
Want to learn about My_Favorite_Programming-Language_?

.. _My_Favorite_Programming-Language: http://www.python.org
```

Phrasenreferenzen

Als „Phrasenreferenzen“ werden Referenznamen bezeichnet, die Interpunktion verwenden oder deren Namen Phrasen sind (zwei oder mehr durch Leerzeichen getrennte Wörter). Phrasenreferenzen werden ausgedrückt, indem die Phrase in Backquotes eingeschlossen wird und der Text als Referenzname behandelt wird:

```
Want to learn about `my favorite programming language`_?

.. _my favorite programming language: http://www.python.org
```

Want to learn about my favorite programming language?

Eigenschaften

Referenznamen sind leerzeichen-neutral und schreibungsunabhängig. Wenn die Referenznamen intern aufgelöst werden, gilt:

- Leerzeichen werden normalisiert (ein oder mehrere Leerzeichen und Zeilenumbrüche werden als ein einziges Leerzeichen interpretiert)
- Groß- und Kleinschreibung wird normalisiert (alle Buchstaben werden in Kleinbuchstaben umgewandelt)

Beispielsweise sind die folgenden Hyperlink-Referenzen äquivalent:

```
- `A HYPERLINK`_
- `a    hyperlink`_
- `A
  Hyperlink`_

.. _A HYPERLINK: https://beckhoff.de/
```

Durch Anklicken einer der Hyperlink-Referenzen wird die Beckhoff Homepage im Bibliotheksverwalter aufgerufen.

- A HYPERLINK
- a hyperlink
- A Hyperlink

Hyperlinks, Fußnoten und Zitate haben denselben Namensraum für Referenznamen. Die Bezeichnungen von Zitaten (einfache Referenznamen) und manuell nummerierten Fußnoten (Dezimalzahlen) werden für das jeweilige Bibliotheksobjekt in dieselbe Datenbank eingetragen wie andere Hyperlink-Namen.

Das bedeutet, dass innerhalb eines Bibliotheksobjekts auf eine Fußnote (definiert als `.. [1]`), auf die normalerweise durch eine Fußnotenreferenz (`[1]_`) verwiesen wird, auch durch eine einfache Hyperlink-Referenz (`1_`) verwiesen werden. Achten Sie darauf, dass es nicht zu Konflikten mit Referenznamen kommt.

Siehe auch: Mehrdeutigkeit bei impliziten und expliziten Hyperlinks innerhalb eines Bibliotheksobjekts [▶ 340]

13.6.2.4 Kommentarstruktur

Kommentare können auf verschiedene Art und Weise strukturiert werden.

Zum einen können Kommentare in Abschnitte gegliedert werden, die wieder in Abschnitte unterteilt werden und/oder Textkörperelemente enthalten können. Abschnitte werden dabei durch einen Titel gekennzeichnet.

Zum anderen können in einem Kommentar Absätze und Textkörperelemente durch Übergänge getrennt werden. Hierbei werden Trennzeichen zur Texteinteilung verwendet.

13.6.2.4.1 Abschnitte

Abschnitte werden durch einen Titel und eine Nummerierung gekennzeichnet. Der Titeltext wird dabei im Kommentar mit „Unterstrichen“ oder mit „Unterstrichen" und passenden „Überstrichen“ versehen.

Eigenschaften

- Ein Unterstrich/Überstrich ist ein einzelnes Zeichen, das in der ersten Spalte beginnt und eine Zeile bildet, die mindestens bis zum rechten Rand des Titeltextes reicht. Bei Verwendung von Überstrichen müssen Länge und Zeichen mit den Unterstrichen übereinstimmen.
- Das Format von Titeltexten, die nur mit Unterstrichen oder die mit Unterstrichen und Überstrichen versehen werden, unterscheidet sich, auch wenn die gleichen Zeichen verwendet werden.
- Ein Unterstrich/Überstrich-Zeichen kann jedes nicht-alphanumerische 7-Bit-ASCII-Zeichen sein. Die folgenden sind Zeichen sind gültige Unterstrich/Überstrich-Zeichen für Abschnittstitel:
  ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
  Folgende Zeichen werden empfohlen:
  „=“, „-“, „*“
- Die Anzahl an Ebenen von Abschnittstiteln ist auf fünf Ebenen beschränkt.
- Das Unterstrich/Überstrich-Zeichen bestimmt nicht das Format des Titels. Dieses ergibt sich aus der Reihenfolge, in der die unterstrichenen bzw. unter- und überstrichenen Titel angetroffen werden. Der erste angetroffene Stil bildet den äußersten Titel, der zweite Stil ist ein Untertitel, der dritte ein Untertitel und so weiter. Titel, die mit einfachen Bindestrichen („-“) unterstrichen sind, werden unabhängig von der Position immer als Titel erster Ebene in grau und fett dargestellt.

```
Section Title
-------------

1. Section Title
****************

1.1 Section Title
=================

-------------------
1.2.1 Section Title
-------------------

=====================
1.2.1.1 Section Title
=====================

=====================
1.2.1.2 Section Title
=====================

-------------------
1.2.2 Section Title
-------------------

1.2 Section Title
=================

2. Section Title
****************

2.1 Section Title
=================
```

```
Section Title
-------------

================
1. Section Title
================

-----------------
1.1 Section Title
-----------------

1.2.1 Section Title
===================

1.2.1.1 Section Title
*********************

1.2.1.2 Section Title
*********************

1.2.2 Section Title
===================

-----------------
1.2 Section Title
-----------------

=================
2. Section Title
=================

-----------------
2.1 Section Title
-----------------
```

Beide Codeausführungen führen zu folgender Darstellung im Bibliotheksverwalter:

Section Title

1. Section Title

1.1 Section Title

1.2.1 Section Title

1.2.1.1 Section Title

1.2.1.2 Section Title

1.2.2 Section Title

1.2 Section Title

2. Section Title

2.1 Section Title

- Es müssen nicht alle Formate für Abschnittstitel verwendet werden, und es muss auch kein spezifischer Stil für Abschnittstitel verwendet werden. Ein Kommentar muss jedoch in der Verwendung von Abschnittsüberschriften konsistent sein: Sobald eine Hierarchie von Titelstilen erstellt ist, müssen Abschnitte diese Hierarchie verwenden.
- Jedem Abschnittstitel wird automatisch ein Hyperlink-Ziel zugewiesen, auf das verwiesen werden kann. Der Referenzname des Hyperlinks entspricht dem Text der Abschnittsüberschrift (siehe Implizite Hyperlink-Ziele [▶ 340]).

Beispiel

Das folgende Beispiel zeigt die Verwendung von Abschnitten und Überschriften in einem reStructuredText-Kommentar. Eine Leerzeile unter einem Titel ist optional. Alle Textblöcke bis zum nächsten Titel der gleichen oder höheren Ebene sind in einem Abschnitt (oder Unterabschnitt etc.) enthalten.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Comment structure\FB_Libdoc_Sections)

```
(*
Section Title
-------------
This document consists of two main sections and several subsections. All text blocks up to the next
title of the same or higher level
are included in a section (or subsection, etc.).

================
1. Section Title
================
Sections are identified through their titles, which are marked up with adornment:
"underlines" below the title text, or underlines and matching "overlines" above the title.

-----------------
1.1 Section Title
-----------------
A document must be consistent in its use of section titles: once a hierarchy of title styles is esta
blished,
sections must use that hierarchy.
Rather than imposing a fixed number and order of section title adornment styles,
the order enforced will be the order as encountered.
The first style encountered will be an outermost title, the second style will be a subtitle,
the third will be a subsubtitle, and so on.

1.2.1 Section Title
===================
Underline-only adornment styles are distinct
from overline-and-underline styles that use the same character.

1.2.1.1 Section Title
*********************
An underline/overline is a single repeated punctuation character that begins in column 1
and forms a line extending at least as far as the right edge of the title text.

1.2.1.2 Section Title
*********************

1.2.2 Section Title
===================

-----------------
1.2 Section Title
-----------------

=================
2. Section Title
=================

-----------------
2.1 Section Title
-----------------
*)
```

FB_Libdoc_Sections (FB)

FUNCTION_BLOCK FB_Libdoc_Sections

Section Title

This document consists of two main sections and several subsections. All text blocks up to the next title of the same or higher level are included in a section (or subsection, etc.).

1. Section Title

Sections are identified through their titles, which are marked up with adornment: “underlines” below the title text, or underlines and matching “overlines” above the title.

1.1 Section Title

A document must be consistent in its use of section titles: once a hierarchy of title styles is established, sections must use that hierarchy. Rather than imposing a fixed number and order of section title adornment styles, the order enforced will be the order as encountered. The first style encountered will be an outermost title, the second style will be a subtitle, the third will be a subsubtitle, and so on.

1.2.1 Section Title

Underline-only adornment styles are distinct from overline-and-underline styles that use the same character.

1.2.1.1 Section Title

An underline/overline is a single repeated punctuation character that begins in column 1 and forms a line extending at least as far as the right edge of the title text.

1.2.1.2 Section Title

1.2.2 Section Title

1.2 Section Title

2. Section Title

2.1 Section Title

InOut:

Scope	Name	Type
Input	nVarA	INT
	nVarB	INT
Output	nSum	INT

13.6.2.4.2 Übergänge

Anstelle von Zwischenüberschriften können Trennzeichen zwischen Absätzen und Textkörperelementen verwendet werden, um Texteinteilungen zu markieren oder um Änderungen im Betreff oder in der Betonung zu signalisieren. Die Trennung von Absätzen und Textkörperelementen mittels Trennzeichen wird als Übergang bezeichnet.

Eigenschaften

- Ein Übergang sollte einen Kommentar oder einen Abschnitt weder beginnen noch beenden. Außerdem sollten zwei Übergänge nicht unmittelbar untereinander liegen.
- Die Syntax für eine Übergangsmarkierung ist eine horizontale Zeile mit vier oder mehr Zeichen. Gültige Trennzeichen sind wie bei der Titelmarkierung alle nicht-alphanumerische 7-Bit-ASCII-Zeichen (z. B. „-“, „+“, „*“, „=“). Vor und nach der Übergangsmarkierung muss jeweils eine Leerzeile eingefügt werden.
- Im Gegensatz zu den Abschnittstiteln wird keine Hierarchie von Übergangsmarkierungen erstellt. Unterschiede in den Übergangsmarkierungen haben keine Auswirkungen. Es wird empfohlen, einen einheitlichen Stil zu verwenden.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Comment structure\FB_Libdoc_Transitions)

```
(*
A transition marker is a horizontal line
of 4 or more repeated punctuation characters.

------------

A transition should not begin or end a section or document,
nor should two transitions be immediately adjacent.
*)
```

A transition marker is a horizontal line of 4 or more repeated punctuation characters.

A transition should not begin or end a section or document, nor should two transitions be immediately adjacent.

13.6.2.5 Textblöcke

Folgende Textblockarten werden in einem reStructuredText-Kommentar unterschieden:

- Textblock (Absatz) [▶ 324]
- Eingerückter Textblock (Blockzitat) [▶ 325]
- Zeilenorientierter Textblock (Zeilenblock) [▶ 326]

13.6.2.5.1 Textblock (Absatz)

Einfache Textblöcke mit linksbündig ausgerichtetem Text und ohne explizite Auszeichnung werden in reStructuredText als Absätze bezeichnet.

Eigenschaften

- Absätze werden durch Leerzeilen voneinander und von anderen Textkörperelementen getrennt.
- Absätze können Inline-Markup enthalten.

Prinzip

```
+----------------------+
|paragraph             |
+----------------------+
+----------------------+
|paragraph             |
+----------------------+
```

Beispiel

Das folgende Beispiel zeigt zwei Absätze in einem reStructuredText-Kommentar. Innerhalb von Absätzen wird der Text fortlaufend dargestellt und automatisch umgebrochen, wenn die Fensterbreite erreicht wird. (Im Beispielprojekt [▶ 305]: B_DocuElements\Text blocks\FB_Libdoc_Paragraph)

```
(*
This is a paragraph.

Paragraphs line up at their left edges, and are normally separated
by blank lines.
*)
```

This is a paragraph.

Paragraphs line up at their left edges, and are normally separated by blank lines.

Siehe auch:

- Leerzeilen und Einrückung [▶ 312]
- Eingerückter Textblock (Blockzitat) [▶ 325]
- Zeilenorientierter Textblock (Zeilenblock) [▶ 326]

13.6.2.5.2 Eingerückter Textblock (Blockzitat)

Textblöcke, die ohne explizite Auszeichnung relativ zum vorangestellten Text eingerückt sind, werden in reStructuredText als Blockzitate bezeichnet.

Eigenschaften

- Die minimale Einrückung beträgt ein Leerzeichen.
- Die gesamte Markup-Verarbeitung (für Textkörperelemente und Inline-Markup) wird innerhalb des Blockzitats fortgesetzt.

`Unindented paragraph.`
 	
`  Block quote 1.`
 	
`  Block quote 2.`
 	
`  Block quote 3.`
	Unindented paragraph.
	
	Block quote 1.
	
	Block quote 2.
	
	Block quote 3.

`- List item.`
 	
`  Block quote.`
	▪ List item.
	
	Block quote.

Prinzip

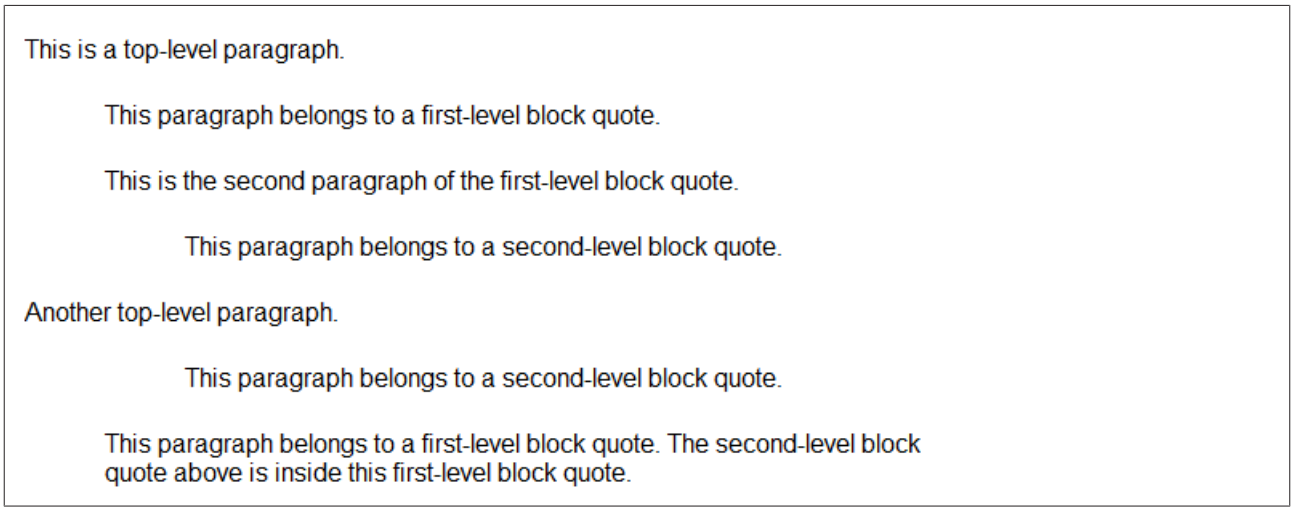
```
+--------------------+
|(current level of)  |
|indentation)        |
+--------------------+
    +---------------------+
    |block quote          |
    |(body elements)      |
    +---------------------+
```

Beispiel

Das folgende Beispiel zeigt verschachtelte Blockzitate.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Text blocks\FB_Libdoc_BlockQuote)

- Eine Leerzeile zwischen einem Textblock (Absatz oder Blockzitat) und einem nachfolgenden eingerückten Textblock (Blockzitat) ist optional. Durch die Einrückung des nachfolgenden Textblocks erfolgt am Ende der vorherigen Textblocks ein Umbruch. Wenn eine Leerzeile im Kommentar eingefügt wird, erfolgt der Umbruch und die Leerzeile bleibt in der Darstellung erhalten.
- Zwischen einem eingerückten Textblock (Blockzitat) und einem nachfolgenden nicht eingerückten Textblock (Absatz oder Blockzitat) wird automatisch, unabhängig davon, ob im Kommentar eine Leerzeile eingefügt wurde oder nicht, eine Leerzeile dargestellt. Die aktuelle Ebene der Einrückung wird beendet.
- Zwischen zwei Textblöcken (Blockzitate) auf einer Einrückungsebene muss im Kommentar eine Leerzeile eingefügt werden, um die Textblöcke voneinander zu trennen.

```
(*
This is a top-level paragraph.

    This paragraph belongs to a first-level block quote.

    This is the second paragraph of the first-level block quote.

        This paragraph belongs to a second-level block quote.

Another top-level paragraph.

        This paragraph belongs to a second-level block quote.

    This paragraph belongs to a first-level block quote. The
    second-level block quote above is inside this first-level
    block quote.
*)
```



Siehe auch:

- Leerzeilen und Einrückung [▶ 312]
- Zeilenorientierter Textblock (Zeilenblock) [▶ 326]

13.6.2.5.3 Zeilenorientierter Textblock (Zeilenblock)

Textblöcke, bei denen jede Zeile mit dem Präfix „|“ gefolgt von einem Leerzeichen beginnt, werden in reStructuredText als Zeilenblöcke bezeichnet. Jedes vertikale Balkenpräfix kennzeichnet eine neue Zeile.

Zeilenblöcke eignen sich unter anderem für Strukturen, bei denen der Zeilenaufbau von Bedeutung ist. Außerdem kann ein Zeilenblock verwendet werden, um Leerzeilen in der Darstellung im Bibliotheksverwalter zu erzwingen oder Textkörperelemente zu trennen.

Eigenschaften

- Durch Einrückungen können geschachtelte Strukturen entstehen.
- Umgebrochene Abschnitte langer Zeilen sind Folgezeilen. Sie beginnen mit einem Leerzeichen anstelle des senkrechten Balkens. Der linke Rand einer Folgezeile muss eingerückt sein, muss aber nicht an dem linken Rand des darüber liegenden Textes ausgerichtet sein.
- Inline-Markup werden verarbeitet.
- Vor und nach einem Zeilenblock muss eine Leerzeile eingefügt werden.

Prinzip

```
+----+------------------+
|"|  "|line              |
+----|continuation line  |
     +------------------+
```

Beispiel

Das folgende Beispiel zeigt die Verschachtelung von Zeilenblöcken und die Struktur von Folgezeilen. (Im Beispielprojekt [▶ 305]: B_DocuElements\Text blocks\FB_Libdoc_LineBlock)

```
(*
| Each new line begins with
| a vertical bar ("|").
|     Line breaks and initial indents
|     are preserved.
| Continuation lines are wrapped
  portions of long lines; they begin
  with spaces in place of vertical bars.
*)
```

Each new line begins with
a vertical bar ("|").
    Line breaks and initial indents
    are preserved.
Continuation lines are wrapped portions of long lines; they begin with spaces in place of vertical bars.

Wenn vor einem Zeilenblock ein anderes Textkörperelement steht, kann der gesamte Zeilenblock eingerückt werden (minimale Einrückung ist ein Leerzeichen).

```
(*
Line blocks are useful where the structure of lines is significant.

| Each new line begins with
| a vertical bar ("|").
|     Line breaks and initial indents
|     are preserved.
| Continuation lines are wrapped
  portions of long lines; they begin
  with spaces in place of vertical bars.

    | Each new line begins with
    | a vertical bar ("|").
    |     Line breaks and initial indents
    |     are preserved.
    | Continuation lines are wrapped
      portions of long lines; they begin
      with spaces in place of vertical bars.
*)
```

```
Line blocks are useful where the structure of lines is significant.

Each new line begins with
a vertical bar ("|").
    Line breaks and initial indents
    are preserved.
Continuation lines are wrapped portions of long lines; they begin with spaces
in place of vertical bars.

        Each new line begins with
        a vertical bar ("|").
            Line breaks and initial indents
            are preserved.
        Continuation lines are wrapped portions of long lines; they
        begin with spaces in place of vertical bars.
```

13.6.2.6 Listen

In reStructuredText werden folgende Listentypen unterschieden:

- Ungeordnete Aufzählungsliste [▶ 328]
- Geordnete (nummerierte) Aufzählungsliste [▶ 329]
- Definitionsliste [▶ 331]
- Feldliste [▶ 332]

13.6.2.6.1 Ungeordnete Aufzählungsliste

Eine ungeordnete Liste besteht aus Textblöcken (Aufzählungspunkten), die mit einem „-“, „+“ oder „*“ gefolgt von einem Leerzeichen beginnen.

Eigenschaften

- Unabhängig vom Aufzählungszeichen wird im Bibliotheksverwalter ein Kästchen als Aufzählungszeichen dargestellt.
- Das Sternchen „*“ darf nicht an erster Stelle im Kommentar verwendet werden.
- Die Listeneinträge sind linksbündig.
- Die Einrückung des Aufzählungszeichens bestimmt, ob es sich um einen Eintrag einer Haupt- oder Teilliste handelt. Das Aufzählungszeichen einer Teilliste beginnt dabei auf Höhe des Textes der Listeneinträge der Hauptliste.
- Zwischen dem Aufzählungszeichen und dem Text der ersten Zeile eines Listeneintrags steht ein Leerzeichen.
- Wenn der Text eines Listeneintrags über mehrere Zeilen geht, muss der Text der nachfolgenden Zeilen auf Höhe des Textes der ersten Zeile beginnen.
- Vor dem ersten Listeneintrag einer Ebene muss eine Leerzeile eingefügt werden. Leerzeilen zwischen den einzelnen Listeneinträgen sowie nach dem letzten Listeneintrag einer Ebene sind optional.
- Im Bibliotheksverwalter werden Haupt- und Teilliste als ein Block ohne Leerzeilen dargestellt.

Prinzip

```
+----+----------------+
|"- "|list item       |
|    |(body elements) |
+----+----------------|
```

Beispiel

Das folgende Beispiel zeigt eine ungeordnete Aufzählungsliste mit zwei Ebenen. Vor den Listeneinträgen der Teilliste befinden sich zwei Leerzeichen.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Lists\FB_Libdoc_UnorderedEnumerationList)

```
(*
This paragraph is not part of the list.

- This is the first bullet list item.  The blank line above the
  first list item is required; blank lines between list items
  (such as below this paragraph) are optional.

  - This is a sublist. The bullet lines up with the left edge
    of the text blocks above. A sublist is a new list so
    requires a blank line above. A blank line below is optional.

  - This is a sublist. The bullet lines up with the left edge
    of the text blocks above. A sublist is a new list so
    requires a blank line above. A blank line below is optional.

- This is the second item of the main list.

- This is the third item of the main list.

This paragraph is not part of the list.
*)
```

This paragraph is not part of the list.

- This is the first bullet list item. The blank line above the first list item is required; blank lines between list items (such as below this paragraph) are optional.
  - This is a sublist. The bullet lines up with the left edge of the text blocks above. A sublist is a new list so requires a blank line above. A blank line below is optional.
  - This is a sublist. The bullet lines up with the left edge of the text blocks above. A sublist is a new list so requires a blank line above. A blank line below is optional.
- This is the second item of the main list.
- This is the third item of the main list.

This paragraph is not part of the list.

13.6.2.6.2 Geordnete (nummerierte) Aufzählungsliste

Eine geordnete (nummerierte) Liste besteht aus Textblöcken (Aufzählungspunkten), die mit einer Ziffer oder einem Buchstaben und einem Trennzeichen gefolgt von einem Leerzeichen beginnen. Die Listeneinträge folgen einer bestimmten Reihenfolge.

Eigenschaften

- Die Listeneinträge sind linksbündig.
- Die Einrückung des Zählers bestimmt, ob es sich um einen Eintrag einer Haupt- oder Teilliste handelt. Der Zähler einer Teilliste beginnt dabei auf Höhe des Textes der Listeneinträge der Hauptliste.
- Zwischen dem Trennzeichen und dem Text steht ein Leerzeichen.
- Wenn der Text eines Listeneintrags über mehrere Zeilen geht, muss der Text der nachfolgenden Zeilen jeweils auf Höhe des Textes der ersten Zeile beginnen.
- Vor dem ersten Listeneintrag einer Ebene muss eine Leerzeile eingefügt werden. Leerzeilen zwischen den einzelnen Listeneinträgen sowie nach dem letzten Listeneintrag einer Ebene sind optional.
- Im Bibliotheksverwalter werden Haupt- und Teilliste als ein Block ohne Leerzeilen dargestellt.
- Folgende Zähler werden erkannt:
  - Arabische Ziffern: 1, 2, 3, ... (keine Obergrenze)

    - Großbuchstaben: A, B, C,..., Z
    - Kleinbuchstaben: a, b, c,..., z
- Zusätzlich kann der Auto-Enumerator „#“ als Zähler verwendet werden, um eine Liste automatisch zu nummerieren. Automatisch nummerierte Listen können mit einer expliziten Aufzählung (arabische Ziffer) beginnen, die die Reihenfolge festlegt. Vollständig automatisch nummerierte Listen verwenden arabische Ziffern und beginnen mit 1.
- Folgende Trennzeichen werden erkannt, im Bibliotheksverwalter wird jedoch immer ein Punkt dargestellt:
    - angehängt mit einem Punkt: „1.“, „A.“, „a.“
    - umgeben von Klammern: „(1)“, „(A)“, „(a)“
    - gefolgt von einer Rechts-Parenthese: „1)“, „A)“, „a)“
- Eine neue Liste wird immer dann gestartet, wenn
    - ein Zähler gefunden wird, der nicht dasselbe Format und denselben Sequenztyp hat wie die aktuelle Liste (z. B. „1.“ „(a)“ erstellt zwei getrennte Listen).
    - die Aufzählungen nicht in der Reihenfolge sind (z. B. „1.“ und „3.“ erzeugt zwei getrennte Listen).
- Die zweite Zeile jeder Aufzählung wird auf Gültigkeit geprüft. Dadurch soll verhindert werden, dass gewöhnliche Absätze fälschlicherweise als Listeneinträge interpretiert werden, wenn sie mit Text beginnen, der mit Zählern identisch ist. Dieser Text wird z. B. als gewöhnlicher Absatz geparst:

  ```
  A. Einstein was a really
  smart dude
  ```

- Mehrdeutigkeit kann jedoch nicht vermieden werden, wenn der Absatz nur aus einer Zeile besteht. Dieser Text wird als Aufzählung in einer Liste geparst:

  ```
  A. Einstein was a really smart dude
  ```

- Wenn ein einzeiliger Absatz mit einem Text beginnt, der mit einem Zähler identisch ist („A.“, „1.“), „(b)“, usw.), muss dem ersten Zeichen ein Backslash vorangestellt werden, damit die Zeile als normaler Absatz interpretiert werden kann:

  ```
  \A. Einstein was a really smart dude
  ```

Prinzip

```
+-----+-----------------+
|"1. "|list item        |
|     |(body elements)  |
+-----+-----------------|
```

Beispiel

Das folgende Beispiel zeigt eine geordnete (nummerierte) Aufzählungsliste mit zwei Ebenen. Vor den Listeneinträgen der Teilliste befinden sich drei Leerzeichen.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Lists\FB_Libdoc_OrderedNumberedEnumerationList)

```
(*
1) This is the first item in the list.

   (a) This is the first subitem (1a).
   (b) This is the second subitem (1b).

2) This is the second item in the list.

   (a) This is the first subitem (2a).
   (b) This is the second subitem (2b).

#) This item is auto-enumerated.
*)
```

1. This is the first item in the list.
    a. This is the first subitem (1a).
    b. This is the second subitem (1b).
2. This is the second item in the list.
    a. This is the first subitem (2a).
    b. This is the second subitem (2b).
3. This item is auto-enumerated.

13.6.2.6.3 Definitionsliste

Eine Definitionsliste beschreibt eine Auflistung von Definitionen. Jeder Eintrag einer Definitionsliste enthält einen Begriff, eine Definition und optional Klassifikatoren.

Definitionslisten können auf verschiedene Arten verwendet werden:

- Als Wörterbuch oder Glossar. Der Begriff ist das Wort selbst, ein Klassifikator kann verwendet werden, um die Verwendung des Begriffs zu kennzeichnen (Substantiv, Verb etc.), und die Definition folgt.
- Zur Beschreibung von Programmvariablen. Der Begriff ist der Variablenname, ein Klassifikator kann verwendet werden, um den Typ der Variablen (String, Integer usw.) anzugeben, und die Definition beschreibt die Verwendung der Variablen im Programm.

Eigenschaften

- Ein Begriff ist ein einfaches einzeiliges Wort oder eine Phrase.
- Eine Definition ist ein Block, der relativ zum Begriff eingerückt ist (die minimale Einrückung beträgt ein Leerzeichen) und kann mehrere Absätze und andere Körperelemente enthalten.
- Optionale Klassifikatoren können dem Begriff auf derselben Zeile folgen, jeweils nach einem „ : “ (Leerzeichen, Doppelpunkt, Leerzeichen).

Prinzip

```
+--------------------------+
|term[" : "classifier]*    |
+----+---------------------+
     |definition           |
     |(body elements)      |
     +---------------------+
```

Beispiel

Das Beispiel zeigt eine Definitionsliste mit Begriffen, Klassifikatoren und Definitionen.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Lists\FB_Libdoc_DefinitionList)

- Innerhalb der Definitionsliste werden weitere Textkörperelemente verwendet (Absatz, ungeordnete Listeneinträge).
- Zwischen einer Begriffszeile und einem Definitionsblock darf keine Leerzeile vorhanden sein (dies unterscheidet Definitionslisten von Blockzitaten).
- Vor dem ersten und nach dem letzten Listeneintrag der Definitionsliste sind Leerzeilen notwendig, können aber auch dazwischen eingefügt werden.

```
(*
Term 1
    Definition 1.

Term 2
    Definition 2, paragraph 1.

    Definition 2, paragraph 2.

Term 3 : classifier
    Definition 3.

Term 4 : classifier one : classifier two
```

```
    - Item 1
    - Item 2
*)
```

Term 1
Definition 1.

Term 2
Definition 2, paragraph 1.

Definition 2, paragraph 2.

Term 3 : *classifier*
Definition 3.

Term 4 : *classifier one* : *classifier two*
- Item 1
- Item 2

13.6.2.6.4 Feldliste

Feldlisten sind Zuordnungen von Feldnamen zu Feldtexten und können für zweispaltige tabellenartige Strukturen verwendet werden.

Eigenschaften

- Ein Feldname kann aus beliebigen Zeichen und mehreren Wörtern bestehen. Doppelpunkten innerhalb eines Feldnamens muss ein Backslash vorangestellt werden, wenn ihnen ein Leerzeichen folgt.
- Inline-Markup werden in Feldnamen verarbeitet.
- Der Feldname bildet zusammen mit einem Doppelpunkt-Präfix und -Suffix den Feldmarker. Dem Feldmarker folgen ein Leerzeichen und der Feldkörper.
- Der Feldkörper kann mehrere Körperelemente enthalten, die relativ zum Feldmarker eingerückt sind. Die erste Zeile hinter dem Feldmarker bestimmt die Einrückung des Feldkörpers (die minimale Einrückung beträgt ein Leerzeichen).

Prinzip

```
+-----------------+------------+
|":"field name":"| fieldbody  |
+--------+-------+            |
         |(body elements)     |
         +--------------------+
```

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Lists\FB_Libdoc_FieldList)

```
(*
:Organization: Beckhoff Automation GmbH & Co. KG
:Contact: info@beckhoff.de
:Address: | Hülshorstweg 20
          | 33415 Verl
          | Germany
:Authors: - Me
          - Myself
          - I
:Version: 1.0
:Status: released
:Date: 2017-12-07

:Copyright:
 © Beckhoff Automation GmbH & Co. KG, Germany.
 The reproduction, distribution and utilization of this document as well as
 the communication of its contents to others without express authorization are prohibited.
 Offenders will be held liable for the payment of damages.
```

```
 All rights reserved in the event of the grant of a patent, utility model or design.
:Abstract: topic
:Indentation:
 Since the field marker may be quite long, the second
 and subsequent lines of the field body do not have to line up
 with the first line, but they must be indented relative to the
 field name marker, and they must line up with each other.
:LongLongFieldname: Since the field marker is quite long,
                    the field body is shown in the line after the marker.
:Parameter nIn: Integer
:Parameter nVarB: Integer
:*Parameter nVarC*: Integer
:``Parameter nVarD``: Integer
*)
```

Organization: Beckhoff Automation GmbH & Co. KG
Contact: info@beckhoff.de
Address: Hülshorstweg 20
33415 Verl
Germany

Authors:
- Me
- Myself
- I

Version: 1.0
Status: released
Date: 2017-12-07
Copyright: © Beckhoff Automation GmbH & Co. KG, Germany. The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited. Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
Abstract: topic
Indentation: Since the field marker may be quite long, the second and subsequent lines of the field body do not have to line up with the first line, but they must be indented relative to the field name marker, and they must line up with each other.
LongLongFieldname: Since the field marker is quite long, the field body is shown in the line after the marker.
Parameter nIn: Integer
Parameter nVarB: Integer
***Parameter nVarC*:** Integer
``Parameter nVarD``: Integer

13.6.2.7 Tabellen

reStructuredText bietet zwei Syntaxen zur Abgrenzung von Tabellenzellen:

- Einfache Tabelle [▶ 334]
- Gittertabelle [▶ 336]

Direktiven bieten weitere Möglichkeiten der Tabellengenerierung:

- CSV-Tabelle [▶ 337]
- Listentabelle [▶ 338]

13.6.2.7.1 Einfache Tabelle

Einfache Tabellen bieten eine kompakte und leicht zu erfassende, aber begrenzte zeilenorientierte Tabellendarstellung für einfache Datensätze.

Eine vollständigere Tabellendarstellung finden Sie im Abschnitt Gittertabellen [▶ 336].

Eigenschaften

- Zelleninhalte sind typischerweise einzelne Absätze, obwohl in den meisten Zellen beliebige Textkörperelemente dargestellt werden können.
- Einfache Tabellen erlauben mehrzeilige Zeilen (in allen außer der ersten Spalte) und die Verbindung von Spalten, jedoch nicht von Zeilen.
- Einfache Tabellen werden mit horizontalen Umrandungen aus den Zeichen „=“ und „-“ beschrieben. Das Gleichheitszeichen („=“) wird für obere und untere Tabellenränder verwendet, um optionale Kopfzeilen vom Tabellenkörper zu trennen. Der Bindestrich („-“) wird verwendet, um Spaltenabstände in einer einzelnen Zeile durch Unterstreichen der verbundenen Spalten zu kennzeichnen und kann optional zur expliziten und/oder visuellen Trennung von Zeilen verwendet werden.
- Wie bei anderen Textkörperelementen sind vor und nach Tabellen Leerzeilen notwendig. Zur Trennung muss zwischen zwei aufeinander folgenden Tabellen ein Absatz mit Text oder ein Absatz mit vertikalem Balken eingefügt werden. Letzterer wird aus der Ausgabe entfernt.
- Linke Tabellenränder sollten an der linken Kante vorangestellter Textblöcke ausgerichtet sein. Wenn Tabellen eingerückt sind, werden sie als Teil eines Blockzitats betrachtet. Die minimale Einrückung beträgt ein Leerzeichen.

Beispiele

(Im Beispielprojekt [▶ 305]: B_DocuElements\Tables\FB_Libdoc_SimpleTables)

Eine einfache Tabelle beginnt mit einem oberen Rand von Gleichheitszeichen und einem oder mehreren Leerzeichen an jeder Spaltenbegrenzung (zwei oder mehr Leerzeichen sind empfohlen). Unabhängig von der Tabellenbreite muss der obere Rand alle Tabellenspalten vollständig beschreiben.

In der Tabelle müssen mindestens zwei Spalten vorhanden sein (um sie von Abschnittsüberschriften zu unterscheiden). Der obere Rand kann von Kopfzeilen gefolgt werden. Die letzte der optionalen Kopfzeilen wird mit „=“ unterstrichen, wieder mit Leerzeichen an den Spaltenrändern. Unterhalb des Trennzeichens für die Kopfzeile darf sich keine Leerzeile befinden. Die untere Begrenzung der Tabelle besteht aus „=“, auch mit Leerzeichen an Spaltengrenzen.

Hier ist z. B. eine Wahrheitstabelle, eine dreispaltige Tabelle mit einer Kopfzeile und vier Körperzeilen:

```
(*
=====  =====  =======
  A      B    A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
*)
```

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

Linien aus Bindestrichen „-“ können verwendet werden, um benachbarte Spalten zu verbinden. Die Linien müssen alle Spalten abdecken und sich an den festgelegten Spaltengrenzen orientieren. Textzeilen mit Bindestrichen dürfen keinen anderen Text enthalten. Eine Bindestrichlinie gilt jeweils für die Zeile unmittelbar darüber. Hier ist z. B. eine Tabelle mit verbundenen Spalten in der Kopfzeile:

```
(*
=====  =====  ======
   Inputs     Output
------------  ------
  A      B    A or B
=====  =====  ======
False  False  False
True   False  True
False  True   True
True   True   True
=====  =====  ======
*)
```

Inputs		Output
A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

Jede Textzeile muss an den Spaltengrenzen Leerzeichen enthalten, es sei denn, die Zellen wurden über die Spalten hinweg miteinander verbunden.

Jede Textzeile beginnt eine neue Tabellenzeile, außer wenn in der ersten Spalte eine leere Zelle vorhanden ist. In diesem Fall wird die Textzeile als Folgezeile interpretiert:

```
(*
=====  =====  ======
   Inputs     Output
------------  ------
  A      B    A or B
=====  =====  ======
False  False  False
       False  True
False  True   True
True   True   True
=====  =====  ======
*)
```

Inputs		Output
A	B	A or B
False	False False	False True
False	True	True
True	True	True

Um in einer einfachen Tabelle eine neue Zeile ohne Text in der ersten Spalte zu beginnen, verwenden Sie

- einen leeren Kommentar („..“), der in der Ausgabe nicht angezeigt wird, oder
- ein Backslash („\“) gefolgt von einem Leerzeichen

Innerhalb von einfachen Tabellen sind Leerzeilen erlaubt. Ihre Interpretation hängt vom Kontext ab. Leerzeilen zwischen Tabellenzeilen werden ignoriert. Leerzeilen innerhalb von mehrzeiligen Zeilen können Absätze oder andere Körperelemente innerhalb von Zellen trennen.

Die rechte Spalte ist unbegrenzt; der Text kann über den Tabellenrand hinausgehen (wie durch die Tabellenränder angezeigt). Es wird jedoch empfohlen, die Rahmen so lang zu gestalten, dass sie den gesamten Text enthalten.

Das folgende Beispiel veranschaulicht:

- Folgezeilen (Tabellenzeile 2 besteht aus zwei Textzeilen, Tabellenzeile 3 aus drei Textzeilen und Tabellenzeile 4 aus vier Textzeilen)
- Leerzeilen zur Trennung von Absätzen (Zeile 3, Spalte 2)
- Text, der über den rechten Rand der Tabelle hinausgeht
- Neue Zeilen, die in der ersten Spalte keinen Text enthalten (Zeile 4)

```
(*
=====  =====
Col 1  Col 2
=====  =====
1      Second column of row 1.
2      Second column of row 2.
       Second line of paragraph.
3      Second column of row 2.

       Second line of paragraph
4      - Second column of row 3.

       - Second item in bullet
         list (row 4, column 2).
\      Row 5; column 1 will be empty.
=====  =====
*)
```

Col 1	Col 2
1	Second column of row 1.
2	Second column of row 2. Second line of paragraph.
3	Second column of row 3. Second line of paragraph
4	■ Second column of row 4. ■ Second item in bullet list (row 4, column 2).
	Row 5; column 1 will be empty.

13.6.2.7.2 Gittertabelle

Gittertabellen bieten eine vollständige Tabellendarstellung. Sie erlauben beliebige Zellinhalte (Textkörperelemente) sowie die Verbindung von Zeilen und Spalten. Insbesondere bei einfachen Datensätzen können Gittertabellen jedoch umständlich zu erstellen sein.

Eine einfachere (aber begrenzte) Tabellendarstellung finden Sie im Abschnitt Einfache Tabelle [▶ 334].

Eigenschaften

- Gittertabellen werden mit einem visuellen Gitter aus den Zeichen „-“, „=“, „|“ und „+“ beschrieben. Der Bindestrich („-“) wird für horizontale Linien (Zeilentrenner) verwendet. Der senkrechte Balken („|“) wird für senkrechte Linien (Spaltentrenner) verwendet. Das Pluszeichen ("+") wird für Schnittpunkte von horizontalen und vertikalen Linien verwendet. Das Gleichheitszeichen („=“) kann verwendet werden, um Kopfzeilen vom Tabellenkörper zu trennen.
- Wie bei anderen Textkörperelementen sind vor und nach Tabellen Leerzeilen notwendig. Zur Trennung muss zwischen zwei aufeinander folgenden Tabellen ein Absatz mit Text oder ein Absatz mit vertikalem Balken eingefügt werden. Letzterer wird aus der Ausgabe entfernt.
- Linke Tabellenränder sollten an der linken Kante vorangestellter Textblöcke ausgerichtet sein. Wenn Tabellen eingerückt sind, werden sie als Teil eines Blockzitats betrachtet. Die minimale Einrückung beträgt ein Leerzeichen.

Beispiele

Das folgende Beispiel veranschaulicht:

- Verbundene Spalten und Zeilen
- Verschiedene Textkörperelemente in Zellen

(Im Beispielprojekt [▶ 305]: B_DocuElements\Tables\FB_Libdoc_GridTable)

```
(*
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | Cells may span columns.          |
+------------------------+------------+---------------------+
| body row 3             | Cells may  | - Table cells       |
+------------------------+ span rows. | - contain           |
| body row 4             |            | - body elements.    |
+------------------------+------------+---------------------+
*)
```

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	Cells may span columns.		
body row 3	Cells may span rows.	■ Table cells ■ contain ■ body elements.	
body row 4			

Beispiel für eine unerwünschte Interaktion zwischen Gitterzeichen und Zellentext

Die folgende Tabelle enthält eine Zelle in Zeile 2, die von Spalte 2 bis Spalte 4 reicht:

```
+--------------+----------+----------+----------+
| row 1, col 1 | column 2 | column 3 | column 4 |
+--------------+----------+----------+----------+
| row 2        |                                |
+--------------+----------+----------+----------+
| row 3        |          |          |          |
+--------------+----------+----------+----------+
```

Wenn ein vertikaler Balken im Text dieser Zelle verwendet wird, kann es zu unbeabsichtigten Effekten kommen, wenn er versehentlich an einer Spaltengrenze ausgerichtet wird:

```
+--------------+----------+----------+----------+
| row 1, col 1 | column 2 | column 3 | column 4 |
+--------------+----------+----------+----------+
| row 2        | Use the command ``ls | more``.  |
+--------------+----------+----------+----------+
| row 3        |          |          |          |
+--------------+----------+----------+----------+
```

Mehrere Lösungen sind möglich, die darauf abzielen, die Kontinuität die Gitterstruktur zu durchbrechen.

Eine Möglichkeit besteht darin, den Text zu verschieben, indem vor dem Zellentext ein zusätzliches Leerzeichen hinzugefügt wird. Das Leerzeichen wird in der Ausgabe entfernt.

```
+--------------+----------+----------+----------+
| row 1, col 1 | column 2 | column 3 | column 4 |
+--------------+----------+----------+----------+
| row 2        |  Use the command ``ls | more``.  |
+--------------+----------+----------+----------+
| row 3        |          |          |          |
+--------------+----------+----------+----------+
```

Eine andere Möglichkeit besteht darin, eine zusätzliche Zeile zu Zeile 2 hinzuzufügen. Die Leerzeile wird in der Ausgabe entfernt.

```
+--------------+----------+----------+----------+
| row 1, col 1 | column 2 | column 3 | column 4 |
+--------------+----------+----------+----------+
| row 2        | Use the command ``ls | more``.  |
|              |                                |
+--------------+----------+----------+----------+
| row 3        |          |          |          |
+--------------+----------+----------+----------+
```

13.6.2.7.3 CSV-Tabelle

Die Direktive `csv-table` kann verwendet werden, um eine Tabelle aus CSV-Daten (comma-separated values) zu erzeugen.

Beschreibung	Die Direktivenmarkierung besteht aus einem expliziten Markup-Start („.. “) gefolgt vom Typ der Direktive (`csv-table`) und zwei Doppelpunkten. (Siehe auch: Direktiven [▶ 316])
Prinzip	`.. csv-table::`
Eigenschaften	• Zwischen der Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig..

Optionen

Optional kann der Zeilenblock eine flache Liste mit Tabellenoptionen enthalten. Folgende Optionen werden erkannt:

widths : integer [integer...]	Gewichtung der Spaltenbreiten Eine durch Komma oder Leerzeichen getrennte Liste relativer Spaltenbreiten. Standardmäßig sind die Spalten gleich breit (100%/ #Spalten).
header-rows : integer	Kopfzeilentabelle Die Anzahl der Zeilen, die als Tabellenkopf verwendet werden sollen. Standardwert ist 0.
stub-columns : integer	Kopfspaltentabelle Die Anzahl der Tabellenspalten, die als Tabellenkopf verwendet werden sollen. Standardwert ist 0.
header : CSV data	Ergänzende Daten für den Tabellenkopf, die unabhängig von und vor jeder Kopfzeile hinzugefügt werden.

Beispiel

Das folgende Beispiel zeigt eine Kopfzeilentabelle mit vier Spalten und zwei Zeilen.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Tables\FB_Libdoc_CSVTable)

```
(*
.. csv-table:: Property list
   :header: "Items", "Property 1", "Property 2", "Property 3"
   :widths: 10, 15, 15, 15

   "Item 1", 1.67, angular, red
   "Item 2", "not specified", round, blue
*)
```

oder

```
(*
.. csv-table:: Property list
   :header-rows: 1
   :widths: 10, 15, 15, 15

   "Items", "Property 1", "Property 2", "Property 3"
   "Item 1", 1.67, angular, red
   "Item 2", "not specified", round, blue
*)
```

Property list

Items	Property 1	Property 2	Property 3
Item 1	1.67	angular	red
Item 2	not specified	round	blue

13.6.2.7.4 Listentabelle

Die Direktive `list-table` ermöglicht es, eine Tabelle aus Daten in einer zweistufigen Aufzählungsliste zu erstellen.

Beschreibung	Die Direktivenmarkierung besteht aus einem expliziten Markup-Start („.. “) gefolgt vom Typ der Direktive (`list-table`) und zwei Doppelpunkten. (Siehe auch: Direktiven [▶ 316])
Prinzip	`.. list-table::`
Eigenschaften	• Die Teillisten der Aufzählungsliste müssen die gleiche Anzahl an Listeneinträgen enthalten. • Zwischen der Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig.

Optionen

Optional kann der list-table-Zeilenblock eine flache Liste mit Tabellenoptionen enthalten. Folgende Option wird erkannt:

widths : integer [integer...]	Gewichtung der Spaltenbreiten Eine durch Komma oder Leerzeichen getrennte Liste relativer Spaltenbreiten. Standardmäßig sind die Spalten gleich breit (100%/ #Spalten).
header-rows : integer	Kopfzeilentabelle Die Anzahl der Zeilen, die als Tabellenkopf verwendet werden sollen. Standardwert ist 0.
stub-columns : integer	Kopfspaltentabelle Die Anzahl der Spalten, die als Tabellenkopf verwendet werden sollen. Standardwert ist 0.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Tables\FB_Libdoc_ListTable)

```
(*
.. list-table:: Property list
    :widths: 50 50 50
    :header-rows: 1

    * - Items
      - Property 1
      - Property 2
    * - Item 1
      - angular
      - red
    * - Item 2
      - round
      - blue
*)
```

Property list

Items	Property 1	Property 2
Item 1	angular	red
Item 2	round	blue

13.6.2.8 Hyperlinks

Hyperlinks verweisen auf eine andere Stelle innerhalb oder außerhalb des Kommentars. Sie bestehen in der Regel aus zwei Teilen: Hyperlink-Referenz (Quelle) und Hyperlink-Ziel (Ziel). Wenn sich im Textkörper ein Quell-Link befindet, muss auch irgendwo anders im Kommentar ein Ziel-Link vorhanden sein (Ausnahme: Alleinstehende Hyperlinks [▶ 345]).

reStructuredText unterscheidet explizite, implizite und inline Hyperlink-Ziele.

Es ist außerdem möglich, einen Link auf die Dokumentation eines anderen Bibliotheksobjekts zu setzen, welches sich ebenfalls in dieser Bibliothek befindet (siehe Link auf ein anderes Objekt [▶ 359]).

Explizite Hyperlink-Ziele

Explizite Hyperlink-Ziele verweisen auf einen Abschnitt innerhalb der Bausteindokumentation oder auf eine externe Seite und können miteinander verbunden werden. Sie können benannt oder anonymisiert sein. Im Gegensatz zu den benannten Hyperlinks wird bei anonymen Hyperlinks der Referenzname nicht verwendet, um die Referenz mit ihrem Ziel abzugleichen (siehe Anonyme Hyperlinks [▶ 349]).

- Interne Hyperlinks [▶ 341] (Verlinkung auf eine Stelle innerhalb des Kommentars bzw. innerhalb der Bausteindokumentation)
- Externe Hyperlinks [▶ 343] (Verlinkung auf eine Website oder Verknüpfung mit dem E-Mail-Programm)

    - Eingebettete URIs und Aliases [▶ 345]
    - Alleinstehende Hyperlinks [▶ 345]
- Indirekte Hyperlinks [▶ 346] (Verlinkung von expliziten Hyperlink-Zielen)

Inline Hyperlink-Ziele

Inline Hyperlink-Ziele verweisen in den laufenden Text eines Kommentars bzw. einer Bausteindokumentation.

- Inline-Hyperlinks [▶ 348]

Implizite Hyperlink-Ziele

Implizite Hyperlink-Ziele werden durch Abschnittsüberschriften, Fußnoten und Zitate generiert. Im Gegensatz zu expliziten Hyperlink-Zielen erzeugen Abschnittsüberschriften, Fußnoten und Zitate automatisch ein Hyperlink-Ziel auf sich selbst; sie enthalten in ihrer Definition keinen Linkblock. Der Referenzname entspricht der Abschnittsüberschrift bzw. dem Fußnoten- oder Zitatlabel. Ansonsten verhalten sich implizite Hyperlinks identisch zu expliziten Hyperlinks.

- Fußnoten [▶ 350]
- Zitate [▶ 358]
- Abschnitte [▶ 321]

Mehrdeutigkeit bei impliziten und expliziten Hyperlinks innerhalb eines Bibliotheksobjekts

- Explizite und implizite Hyperlink-Ziele mit dem gleichen Referenznamen: Die Hyperlinks funktionieren nicht.

Fehlermeldung: Duplicate target name, cannot be used as a unique reference: "1" ("Beckhoff").

Beispiel:

```
This in an explicit internal hyperlink reference: 1_

For more information see [1]_

--------

.. _1:

This is an explicit internal hyperlink target.

.. [1] Footnote
```

oder

```
See Beckhoff_.

This is an explicit hyperlink to Beckhoff_.

--------

.. _Beckhoff: http:\\www.beckhoff.de

Beckhoff
========
```

- Doppelte implizite Hyperlink-Ziele: Die Hyperlinks funktionieren nicht.

Fehlermeldung: Duplicate target name, cannot be used as a unique reference: "chapter a".

Beispiel:

```
Chapter 1
=========

Chapter a
*********

Chapter 2
=========
```

```
Chapter a
*********

--------

See `Chapter a`_
```

- Doppelte explizite Hyperlink-Ziele: Die Hyperlinks funktionieren nicht. Ausnahme: Duplizierte externe Hyperlink-Ziele (identische Referenznamen und referenzierte URIs) sind konfliktfrei und werden nicht entfernt.

Fehlermeldung: Duplicate target name, cannot be used as a unique reference: "1".

Beispiel:

```
This in an explicit internal hyperlink reference: 1_

This in another explicit internal hyperlink reference: 1_

--------

.. _1:

This is an explicit internal hyperlink target.

.. _1:

This is another explicit internal hyperlink target.
```

Siehe auch: Referenznamen [▶ 319]

13.6.2.8.1 Interne Hyperlinks

Interne Hyperlinks ermöglichen es, innerhalb eines Kommentars eine Stelle mit einer anderen zu verbinden. Das Hyperlink-Ziel zeigt dabei immer auf das ihm nachfolgende Textkörperelement.

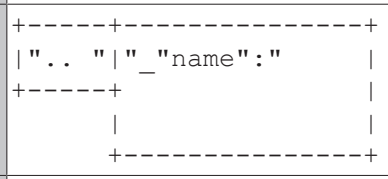
Hyperlink-Referenz

Beschreibung	Die Hyperlink-Referenz besteht aus einem Referenznamen gefolgt von einem Unterstrich:
	`reference-name_`
	Phrasenreferenzen müssen in Backquotes angegeben werden:
	`` `reference name`_ ``
	(Siehe auch: Referenznamen [▶ 319])

Start- und Endzeichen	• Kein Startzeichen, Endzeichen = „_“ • Startzeichen = „`“, Endzeichen = „`_“ (Phrasenreferenzen) (Siehe auch: Inline-Markup [▶ 317])

Hyperlink-Ziel

Beschreibung	Das Hyperlink-Ziel besteht aus einem expliziten Markup-Start („.. “), einem Unterstrich, dem Referenznamen und einem Doppelpunkt:
	`.. _reference-name:`
	Eine Phrasenreferenz im Hyperlink-Ziel kann optional in Backquotes eingeschlossen werden:
	`.. _`reference name`:`
	`.. _reference name:`
	(Siehe auch: Explizite Markup-Blöcke [▶ 316], Referenznamen [▶ 319])

Prinzip	
Eigenschaften	• Interne Hyperlink-Ziele haben einen leeren Linkblock. • Zwischen dem expliziten Markup-Block und dem nachfolgenden Textkörperelement ist eine Leerzeile notwendig. Leerzeilen zwischen expliziten Markup-Blöcken sind optional.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_InternalHyperlinks)

Einfaches internes Hyperlink-Ziel

Durch Anklicken der Hyperlink-Referenz `target_` wird das Textkörperelement unterhalb von `.. _target:` angezeigt.

```
(*
Clicking on this internal hyperlink will take us to the target_
below.

.. _target:

The hyperlink targets above point to this paragraph.
*)
```

Clicking on this internal hyperlink will take us to the target below.

The hyperlink targets above point to this paragraph.

Verschachtelung von internen Hyperlink-Zielen

Der Hyperlink funktioniert auch, wenn das interne Hyperlink-Ziel in einem eingerückten Textblock „verschachtelt“ ist. So können zum Beispiel Hyperlink-Ziele auf einzelne Listenelemente gesetzt werden (mit Ausnahme des ersten, da ein vorhergehendes internes Hyperlink-Ziel für die gesamte Liste gilt):

```
(*
Clicking on this internal hyperlink will take us to the `third item`_ of the bullet list.

* First list item
* Second list item

  .. _third item:

* Third list item, with hyperlink target.
*)
```

Clicking on this internal hyperlink will take us to the third item of the bullet list.

- First list item
- Second list item
- Third list item, with hyperlink target.

Verkettung von internen Hyperlink-Zielen

Interne Hyperlink-Ziele können „verkettet“ werden. Mehrere benachbarte interne Hyperlink-Ziele zeigen dann auf dasselbe Element:

```
(*
Clicking on this internal hyperlink will take us to target1_
and clicking on this internal hyperlink will take us to target2_.
Both targets point the the same paragraph.

.. _target1:
.. _target2:

The targets "target1" and "target2" are synonyms; they both
point to this paragraph.
*)
```

Clicking on this internal hyperlink will take us to target1 and clicking on this internal hyperlink will take us to target2. Both targets point to the same paragraph.

The targets “target1” and “target2” are synonyms; they both point to this paragraph.

Interne Hyperlink-Ziele können auch in den laufenden Text eingefügt werden (siehe: Inline-Hyperlinks [▶ 348]).

13.6.2.8.2 Externe Hyperlinks

Externe Hyperlinks ermöglichen es, aus dem Kommentar bzw. aus der Bausteindokumentation heraus auf eine externe Webseite zu verlinken oder ein E-Mail-Programm zu öffnen.

Hyperlink-Referenz

Beschreibung	Die Hyperlink-Referenz besteht aus einem Referenznamen gefolgt von einem Unterstrich:
	`reference-name_`
	Phrasenreferenzen müssen in Backquotes angegeben werden:
	`` `reference name`_ ``
	(Siehe auch: Referenznamen [▶ 319])

Start- und Endzeichen	• Kein Startzeichen, Endzeichen = „_“ • Startzeichen = „`“, Endzeichen = „`_“ (Phrasenreferenzen) (Siehe auch: Inline-Markup [▶ 317])

Hyperlink-Ziel

Beschreibung	Das Hyperlink-Ziel besteht aus einem expliziten Markup-Start („.. “), einem Unterstrich, dem Referenznamen, einem Doppelpunkt, Leerzeichen und einem Linkblock:
	`.. _reference-name: link-block`
	Eine Phrasenreferenz im Hyperlink-Ziel kann optional in Backquotes eingeschlossen werden:
	`.. _`reference name`: link-block`
	`.. _reference name: link-block`
	(Siehe auch: Explizite Markup-Blöcke [▶ 316], Referenznamen [▶ 319])

Prinzip	`+-----+-----------------+`
	`|".. "|"_"name":" link  |`
	`+-----+block            |`
	`      |                 |`
	`       +-----------------+`

Eigenschaften	• Externe Hyperlink-Ziele haben einen absoluten URI oder eine E-Mail-Adresse in ihrem Linkblock.
	• Der URI eines externen Hyperlinks kann entweder in derselben Zeile wie das explizite Markup beginnen oder in einem eingerückten Textblock unmittelbar danach, ohne dass Leerzeilen dazwischenkommen.
	• Wenn sich mehrere Zeilen im Linkblock befinden, werden diese verkettet. Zeilenumbrüche im Linkblock werden entfernt. Die folgenden externen Verweisziele sind somit äquivalent:
	`.. _one-liner: https://infosys.beckhoff.de/`
	`.. _starts-on-this-line: http://`
	`infosys.beckhoff.de/`
	`.. _entirely-below:`
	`https://infosys.`
	`beckhoff.de/`
	Wenn der URI eines externen Hyperlink-Ziels als letztes Zeichen einen Unterstrich enthält, muss dieses „escaped“ werden, um nicht mit einem indirekten Verweisziel verwechselt zu werden (siehe Escaping-Mechanismus [▶ 319]).

Beispiel

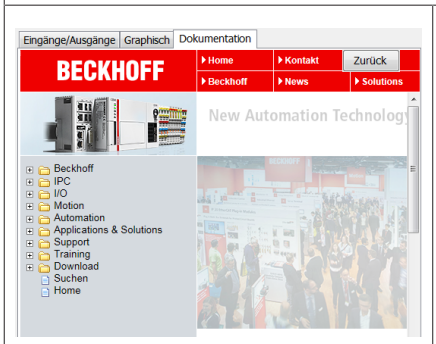
Durch Anklicken der Hyperlink-Referenz wird die Webseite direkt im Bibliotheksverwalter aufgerufen bzw. ein entsprechendes E-Mail-Programm geöffnet.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_ExternalHyperlinks)

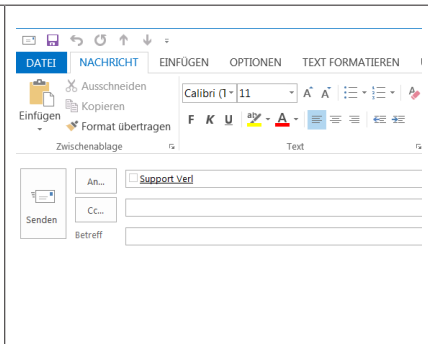
```
(*
See the Beckhoff_ home page for more information.

Please contact the `Beckhoff Support`_ for technical assistance.

.. _Beckhoff: http://www.beckhoff.de
.. _Beckhoff Support: support@beckhoff.com

For more information see the `PLC Lib Tc2_System`_ and `PLC Lib Tc2_Standard`_.

.. _PLC Lib Tc2_System: https://download.beckhoff.com/download/document/automation/twincat3/
TwinCAT_3_PLC_Lib_Tc2_System_EN.pdf
.. _PLC Lib Tc2_Standard: https://download.beckhoff.com/download/document/automation/twincat3/
TwinCAT_3_PLC_Lib__Tc2_Standard_EN.pdf
*)
```

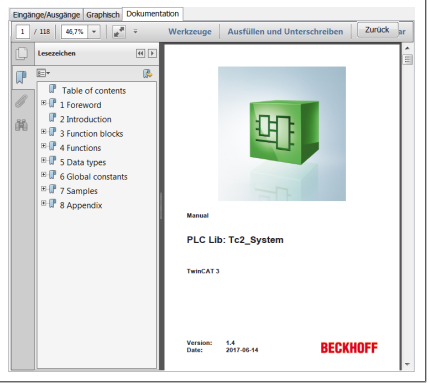
See the Beckhoff home page for more information.

Please contact the Beckhoff Support for technical assistance.

For more information see PLC Lib Tc2_System and PLC Lib Tc2_Standard.







Es ist auch möglich, URIs direkt in Verweisreferenz einzubinden.

```
(*
External hyperlinks, like `Beckhoff <http://www.beckhoff.de/>`_.
*)
```

External hyperlinks, like Beckhoff.

Siehe auch: Eingebettete URIs und Aliases [▶ 345]

13.6.2.8.2.1 Alleinstehende Hyperlinks

Ein URI (Uniform Resource Identifier) innerhalb eines Textblocks wird als allgemeiner externer Hyperlink behandelt, wobei der URI selbst als Text des Links dargestellt wird.

Start- und Endzeichen	Keine Start- oder Endzeichen.
Eigenschaften	• Absolute URIs und eigenständige E-Mail-Adressen werden erkannt.

Beispiel

Das folgende Beispiel zeigt zwei alleinstehende Hyperlinks. Der URI selbst entspricht dem Linktext.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_ExternalHyperlinks)

```
(*
See http://www.beckhoff.de for info.

Contact support@beckhoff.com
*)
```

See http://www.beckhoff.de for info.

Contact support@beckhoff.com

13.6.2.8.2.2 Eingebettete URIs und Aliases

Eigenschaften

- Dem eingeklammerten URI muss ein Leerzeichen vorangestellt werden.
- Der eingeklammerte URI muss der letzte Text vor dem Ende der Zeichenkette sein.

- Mit einem einzigen abschließenden Unterstrich wird die Hyperlink-Referenz „benannt“ und der gleiche Ziel-URI kann erneut referenziert werden.
- Bei zwei nachgestellten Unterstrichen sind Hyperlink-Referenz und Hyperlink-Ziel anonym und das Ziel kann nicht erneut referenziert werden. Dies sind „einmalige“ Hyperlinks.

Zum Beispiel:

```
See the `Beckhoff Online Information System <http://infosys.beckhoff.de/
>`__ or the `Beckhoff home page <http://beckhoff.de/>`__.
```

Dies entspricht:

```
See the `Beckhoff Online Information System`__ or the `Beckhoff home page`__.

__ http://infosys.beckhoff.de/
__ http://beckhoff.de/
```

See the Beckhoff Online Information System or the Beckhoff home page.

Der Referenztext kann auch weggelassen werden, in diesem Fall wird der URI zur Verwendung als Referenztext dupliziert:

```
See the `<http://infosys.beckhoff.de/>`__ or the `<http://beckhoff.de/>`__.
```

See the http://infosys.beckhoff.de/ or the http://beckhoff.de/.

Das eingebettete URI-Konstrukt bietet eine einfache Erstellung und Pflege von Hyperlinks auf Kosten der allgemeinen Lesbarkeit. Inline-URIs, besonders lange, unterbrechen zwangsläufig den natürlichen Textfluss.

Beispiel

(Im Beispielprojekt: B_DocuElements\Hyperlinks\FB_Libdoc_ExternalHyperlinks)

Eine Hyperlink-Referenz kann direkt einen Ziel-URI in spitzen Klammern („<...>“) einbetten.

```
See the `Beckhoff home page <http://www.beckhoff.de>`_ for info.

This `link <Beckhoff home page_>`_ is an alias to the link above.
```

Dies entspricht:

```
See the `Beckhoff home page`_ for info.

This link_ is an alias to the link above.

.. _Beckhoff home page: http://www.beckhoff.de
.. _link: `Beckhoff home page`_
```

See the Beckhoff home page for info.

This link is an alias to the link above.

13.6.2.8.3 Indirekte Hyperlinks

Bei indirekten Hyperlinks enthält das Hyperlink-Ziel selbst wieder eine Hyperlink-Referenz. Indirekte Hyperlinks ermöglichen so die Verlinkung von expliziten Hyperlink-Zielen.

Hyperlink-Referenz

Beschreibung	Die Hyperlink-Referenz besteht aus einem Referenznamen gefolgt von einem Unterstrich:
	`reference-name_`
	Phrasenreferenzen müssen in Backquotes angegeben werden:
	`` `reference name`_ ``
	(Siehe auch: Referenznamen [▶ 319])

Start- und Endzeichen	• Kein Startzeichen, Endzeichen = „_“ • Startzeichen = „`“, Endzeichen = „`_“ (Phrasenreferenzen) (Siehe auch: Inline-Markup [▶ 317])

Hyperlink-Ziel

Beschreibung	Das Hyperlink-Ziel besteht aus einem expliziten Markup-Start („.. “), einem Unterstrich, dem Referenznamen, einem Doppelpunkt, Leerzeichen und einem Linkblock:
	`.. _reference-name: link-block`
	Eine Phrasenreferenz im Hyperlink-Ziel kann optional in Backquotes eingeschlossen werden:
	`.. _`reference name`: link-block`
	`.. _reference name: link-block`
	(Siehe auch: Explizite Markup-Blöcke [▶ 316], Referenznamen [▶ 319])

Prinzip	`+-----+----------------+`
	`|".. "|"_"name":" link  |`
	`+-----+block            |`
	`  |                |`
	`  +----------------+`

Eigenschaften	• Indirekte Hyperlink-Ziele haben eine Hyperlink-Referenz in ihrem Linkblock. • Wie bei externen Hyperlink-Zielen kann der Linkblock eines indirekten Hyperlink-Ziels in der gleichen Zeile wie der explizite Markup-Block beginnen oder in der nächsten Zeile. Beispielsweise sind die folgenden indirekten Hyperlink-Ziele äquivalent: `.. _one-liner: `A HYPERLINK`_` `.. _entirely-below:` `  `a hyperlink`_` `.. _split: `A` ` Hyperlink`_` Wenn der Referenzname Doppelpunkte enthält: • muss die Phrase im Linkblock des Hyperlink-Ziels in Backquotes eingeschlossen sein `` `Beckhoff Support:`_ `` `* worldwide support` `* design, programming and commissioning of complex automation systems` `* training program for Beckhoff system components` `.. _`Beckhoff Support:`: support@beckhoff.com` • oder der/die Doppelpunkt(e) muss/müssen mit einem Backslash versehen werden: `` `Beckhoff Support:`_ `` `* worldwide support` `* design, programming and commissioning of complex automation systems` `* training program for Beckhoff system components` `.. _Beckhoff Support\:: support@beckhoff.com`

Bespiele

(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_IndirectHyperlinks)

Indirekte Verweise auf ein internes Verweisziel

Im folgenden Beispiel verweist das Hyperlink-Ziel `.. _one` indirekt auf das Ziel `.. _two` und das Ziel `.. _two` verweist indirekt auf Ziel `.. _three`, ein internes Verweisziel. Eigentlich beziehen sich alle drei Ziele auf dasselbe (auf denselben Absatz):

```
(*
This hyperlink points to target one_ and indirect to target three.

.. _one: two_
.. _two: three_
.. _three:

The hyperlink targets above point to this paragraph.
*)
```

This hyperlink points to target one and indirect to target three.

The hyperlink targets above point to this paragraph.

Indirekte Verweise auf ein externes Verweisziel

Im folgenden Beispiel verweist das Ziel `.. _Beckhoff` indirekt auf das Ziel `.. _Beckhoff Information System`, ein externes Verweisziel.

```
(*
The `Beckhoff Information System`_ is a reference source for Beckhoff_ products

.. _Beckhoff: `Beckhoff Information System`_

.. _Beckhoff Information System: https://infosys.beckhoff.de/
*)
```

The Beckhoff Information System is a reference source for Beckhoff products

Es ist auch möglich, einen Alias direkt in das Hyperlink-Ziel einzufügen (siehe Embedded URIs and Aliases [▶ 346]).

13.6.2.8.4 Inline-Hyperlinks

Inline-Hyperlinks entsprechen internen Hyperlinks, das Hyperlink-Ziel befindet sich jedoch inline im Text.

Hyperlink-Referenz

Beschreibung	Die Hyperlink-Referenz besteht aus einem Referenznamen gefolgt von einem Unterstrich:
	`reference-name_`
	Phrasenreferenzen müssen in Backquotes angegeben werden:
	`` `reference name`_ ``
	(Siehe auch: Referenznamen [▶ 319])

Start- und Endzeichen	• Kein Startzeichen, Endzeichen = „_“ • Startzeichen = „`“, Endzeichen = „`_“ (Phrasenreferenzen) (Siehe auch: Inline-Markup [▶ 317])

Hyperlink-Ziel

Beschreibung	Das Hyperlink-Ziel besteht aus einem Unterstrich gefolgt vom Referenznamen.
	`_reference-name`
	Phrasenreferenzen müssen in Backquotes angegeben werden.
	`_`reference name``
	Interne Inline-Ziele dürfen nicht anonym sein.

Start- und Endzeichen	• Starzeichen = „_“, kein Endzeichen • Startzeichen = „_`”, Endzeichen = „`” (Siehe auch: Inline-Markup [▶ 317])

Beispiel

Der folgende Absatz enthält beispielsweise eine Hyperlink-Referenz und ein inline Hyperlink-Ziel namens „FB_Sample“. Durch Anklicken der Hyperlink-Referenz wird der Satz mit dem Hyperlink-Ziel angezeigt.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_InlineHyperlinks)

```
(*
The function block _`FB_Sample` is used to...

For more information see the description of the `FB_Sample`_.
*)
```

The function block FB_Sample is used to...

For more information see the description of the FB_Sample.

13.6.2.8.5 Anonyme Hyperlinks

Bezeichnungen von Hyperlinks sollten im Allgemeinen möglichst ausführlich und sprechend sein. Das Duplizieren eines langen Referenznamens im Hyperlink-Ziel kann jedoch aufwändig und fehleranfällig sein.

Bei anonymen Hyperlinks enthält das Hyperlink-Ziel keinen Referenznamen, d. h. der Name der Hyperlink-Referenz wird nicht verwendet, um die Referenz mit ihrem Ziel abzugleichen. Stattdessen ist die Reihenfolge der anonymen Hyperlink-Ziele innerhalb des Kommentars signifikant: Die erste anonyme Hyperlink-Referenz wird auf das erste anonyme Hyperlink-Ziel verlinkt usw. Die Anzahl der anonymen Hyperlink-Referenzen im Kommentar muss dabei mit der Anzahl der anonymen Hyperlink-Ziele übereinstimmen.

Anonyme Hyperlinks können zu missverständlichen und unlesbaren Kommentaren führen. Aus Gründen der Lesbarkeit wird empfohlen, die Hyperlink-Ziele möglichst nahe an den Hyperlink-Referenzen zu platzieren.

Beachten Sie beim Bearbeiten von Kommentaren mit anonymen Hyperlinks, dass insbesondere beim Hinzufügen, Entfernen und Neuanordnen von Hyperlink-Referenzen auch die Reihenfolge der entsprechenden Hyperlinke-Ziele angepasst werden muss.

Hyperlink-Referenz

Beschreibung	Die Hyperlink-Referenz besteht aus dem Referenznamen gefolgt von zwei Unterstrichen:
	`anonymous-hyperlink-reference-name__.`
	Phrasenreferenzen müssen in Backquotes angegeben werden.
	`` `anonymous hyperlink reference name`__. ``

Start- und Endzeichen	• Kein Starzeichen, Endzeichen = „__“ • Startzeichen = „`“, Endzeichen = „`__“ (Phrasenreferenzen) (Siehe auch: Inline-Markup [▶ 317])

Hyperlink-Ziel

Beschreibung	Das Hyperlink-Ziel besteht aus einem expliziten Markup-Start („.. “), zwei Unterstrichen, einem Doppelpunkt, Leerzeichen und einem Linkblock. Es gibt keinen Referenznamen.
	`.. __: anonymous-hyperlink-target-link-block`
	Alternativ können anonyme Hyperlink aus zwei Unterstrichen, einem Leerzeichen und einem Linkblock bestehen:
	`__ anonymous-hyperlink-target-link-block`
	(Siehe auch: Explizite Markup-Blöcke [▶ 316])

Prinzip	`+-----+------------------+`
	`|".. "|"__"name":" link  |`
	`+-----+block             |`
	`      |                  |`
	`      +------------------+`

Beispiel

Im folgenden Beispiel verweist die erste anonyme Hyperlink-Referenz auf das Beckhoff Online-Informationssystem und die zweite anonyme Hyperlink-Referenz auf die Beckhoff Homepage. Wenn die Reihenfolge der anonymen Hyperlink-Ziele verändert wird, verändert sich auch die Zuweisung der Verweise. (Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_AnonymousHyperlinks)

```
(*
See the `Beckhoff Online Information System`__ or the `Beckhoff home page`__.

.. __: http://infosys.beckhoff.de/

.. __: http://beckhoff.de/
*)
```

See the Beckhoff Online Information System or the Beckhoff home page.

13.6.2.8.6 Fußnoten

Fußnoten ermöglichen es, Anmerkungen, Bemerkungen oder Quellenangaben zu einer Textstelle aus dem Text auszulagern und so den Text lesbarer zu gestalten.

Fußnoten bestehen wie Hyperlinks aus zwei Teilen: Fußnotenreferenz (Quelle) und Fußnote (Ziel). Sie verwenden numerische Bezeichnungen als Referenznamen.

Typen von Fußnoten

Fußnoten können manuell, automatisch oder manuell und automatisch gemischt nummeriert werden:

- Manuell nummerierte Fußnoten [▶ 350]
- Automatisch nummerierte Fußnoten [▶ 352]
- Benannte automatisch nummerierte Fußnoten [▶ 354]
- Manuell und automatisch nummerierte Fußnoten [▶ 357]

Außerdem besteht die Möglichkeit der automatischen Generierung von Symbolen für die Fußnoten:

- Automatische Generierung von Symbolen [▶ 356]

13.6.2.8.6.1 Manuell nummerierte Fußnoten

Bei manuell nummerierten Fußnoten wird jeder Fußnote eine Bezeichnung zugewiesen.

Fußnotenreferenz

Beschreibung	Die Fußnotenreferenz besteht aus einem Fußnotenlabel in eckigen Klammern gefolgt von einem abschließenden Unterstrich. Das Fußnotenlabel ist eine beliebige ganze Dezimalzahl, die aus einer oder mehreren Ziffern besteht.
Start- und Endzeichen	Startzeichen = „[“, Endzeichen = „]_“ (Siehe auch: Inline-Markup [▶ 317])

Fußnote

Beschreibung	Die Fußnote (Fußnotenbeschriftung) besteht aus einem expliziten Markup-Start („.. “), dem von eckigen Klammern umschlossenen Fußnotenlabel und einem Leerzeichen gefolgt von eingerückten Körperelementen (Fußnoteninhalt). (Siehe auch: Explizite Markup-Blöcke [▶ 316])

Prinzip	`+-----+----------------------+`
	`|".. "|"["label"]" footnote  |`
	`+-----+                      |`
	`  |(body elements)       |`
	`  +----------------------+`

Eigenschaften	• Zwischen der Fußnotenbeschriftung und dem Fußnoteninhalt kann, aber muss keine Leerzeile sein.
	• Der Fußnoteninhalt muss eingerückt sein (mindestens ein Leerzeichen).
	• Fußnoten können überall im Kommentar positioniert werden, nicht nur am Ende.
	Zum Beispiel:
	`Footnote references, like [1]_.`
	
	`.. [1] Body elements go here.`
	oder
	`Footnote references, like [1]_`
	
	`.. [1]`
	` Body elements go here.`
	
	Footnote references, like [1].
	
	[1] Body elements go here.
	
	oder
	`Footnote references, like [1]_.`
	
	`.. [1]`
	` - Body elements go here.`
	` - Second body element.`
	
	Footnote references, like [1].
	
	[1] ■ Body elements go here.
	■ Second body element.

Beispiel

Durch Anklicken der Fußnotenreferenz [5]_ wird der Fußnoteninhalt angezeigt.
(Im Beispielprojekt [▶ 305]:
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_ManuallyNumberedFootnotes)

```
(*
**Numerical footnote**

Footnote references, like [5]_. Note that footnotes may get rearranged, e.g., to the bottom of the "
```

```
page".

--------

.. [5] A numerical footnote. Note there's no colon after the ``]``.
*)
```

Numerical footnote

Footnote references, like [5]. Note that footnotes may get rearranged, e.g., to the bottom of the “page”.

[5] A numerical footnote. Note there’s no colon after the] .

13.6.2.8.6.2 Automatisch nummerierte Fußnoten

Bei automatisch nummerierten Fußnoten wird jeder Fußnote automatisch eine Bezeichnung zugewiesen.

Fußnotenreferenz

Beschreibung	Die Fußnotenreferenz besteht aus einem Fußnotenlabel in eckigen Klammern gefolgt von einem abschließenden Unterstrich. Das Fußnotenlabel ist ein einzelnes „#“.
Start- und Endzeichen	Startzeichen = „[“, Endzeichen = „]_“ (Siehe auch: Inline-Markup [▶ 317])

Fußnote

Beschreibung	Jede Fußnote (Fußnotenbeschriftung) besteht aus einem expliziten Markup-Start („.. “), dem von eckigen Klammern umschlossenen Fußnotenlabel und einem Leerzeichen gefolgt von eingerückten Körperelementen (Fußnoteninhalt). (Siehe auch: Explizite Markup-Blöcke [▶ 316])

Prinzip	`+-----+-------------------+`
	`|".. "|"["#"]" footnote  |`
	`+-----+                   |`
	`      |(body elements)    |`
	`      +-------------------+`

Eigenschaften	• Die erste Fußnote zur Anforderung einer automatischen Nummerierung erhält das Label „1“, die zweite das Label „2“ usw. (sofern keine manuell nummerierten Fußnoten vorhanden sind). Eine Fußnote, die automatisch ein Label „1“ erhalten hat, erzeugt ein implizites Hyperlink-Ziel mit dem Namen „1“, so als ob das Label explizit angegeben wäre.
	• Die Nummerierung wird durch die Reihenfolge der Fußnoten bestimmt, nicht durch die Reihenfolge der Fußnotenreferenzen. Bei automatisch nummerierten Fußnoten ([#]_) müssen die Fußnoten und Fußnotenreferenzen in der gleichen relativen Reihenfolge angeordnet sein, müssen aber nicht zusammenhängen.
	Zum Beispiel:
	`[#]_ is a reference to footnote 1, and [#]_ is a reference to footnote 2.`
	`.. [#] This is footnote 1.`
	`.. [#] This is footnote 2.`
	`.. [#] This is footnote 3.`
	`[#]_ is a reference to footnote 3.`
	[1] is a reference to footnote 1, and [2] is a reference to footnote 2.
	[1] This is footnote 1.
	[2] This is footnote 2.
	[3] This is footnote 3.
	[3] is a reference to footnote 3.
	Besondere Vorsicht ist geboten, wenn Fußnoten selbst autonummerierte Fußnotenverweise enthalten oder wenn mehrere Verweise in unmittelbarer Nähe erfolgen. Fußnoten und Verweise werden in der Reihenfolge notiert, in der sie im Kommentar vorkommen, was nicht unbedingt mit der Reihenfolge übereinstimmt, in der eine Person sie lesen würde.
	(Siehe auch: Benannte automatisch nummerierte Fußnoten [▶ 354])

Beispiel

Das folgende Beispiel zeigt automatisch nummerierte Fußnoten ohne und mit zusätzlichem Fußnotenlabel. (Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_AutomaticallyNumberedFootnotes)

```
(*
**Autonumbered footnotes**

Autonumbered footnotes are possible, like using [#]_ and [#]_.

--------

.. [#] This is the first one.

.. [#] This is the second one.
*)
```

Autonumbered footnotes

Autonumbered footnotes are possible, like using [1] and [2].

[1] This is the first one.
[2] This is the second one.

13.6.2.8.6.3 Benannte automatisch nummerierte Fußnoten

Bei benannten automatisch nummerierten Fußnoten wird jeder Fußnote automatisch eine Bezeichnung zugewiesen und gleichzeitig explizit ein Label angegeben.

Fußnotenreferenz

Beschreibung	Die Fußnotenreferenz besteht aus einem Fußnotenlabel in eckigen Klammern gefolgt von einem abschließenden Unterstrich. Das Fußnotenlabel ist ein einzelnes „#“ gefolgt von einem beliebigen Label.
Start- und Endzeichen	Startzeichen = „[“, Endzeichen = „]_“ (Siehe auch: Inline-Markup [▶ 317])

Fußnote

Beschreibung	Jede Fußnote (Fußnotenbeschriftung) besteht aus einem expliziten Markup-Start („.. “), dem von eckigen Klammern umschlossenen Fußnotenlabel und einem Leerzeichen gefolgt von eingerückten Körperelementen (Fußnoteninhalt). (Siehe auch: Explizite Markup-Blöcke [▶ 316])

Prinzip	`+-----+----------------------+`
	`|".. "|"["#label"]" footnote  |`
	`+-----+                      |`
	`      |(body elements)       |`
	`      +----------------------+`

Eigenschaften	• Auf der Fußnote selbst wird ein Hyperlink-Ziel erzeugt, dessen Name dem angegebenen Label (ohne „#“) entspricht.
	• Durch das explizite Label kann eine automatisch nummerierte Fußnote eindeutig zugeordnet und mehrfach als Fußnotenreferenz oder Hyperlinkreferenz bezeichnet werden.
	Das Beispiel zeigt eine automatisch nummerierte Fußnote, auf die sowohl eine Fußnotenreferenz (`[#note]_` bzw. `[#label]_`) als auch eine Hyperlink-Referenz (`note_` bzw. `label_`) verweist.
	`If [#note]_ is the first footnote, it will show up as "[1]".`
	`We can refer to it again as [#note]_.  We can also refer to it as note_ (an ordin`
	`ary internal hyperlink reference).`
	
	`If [#label]_ is the second footnote, it will show up as "[2]".`
	`We can refer to it again as [#label]_.  We can also refer to it as label_ (an ord`
	`inary internal hyperlink reference).`
	
	`.. [#note] This is the first footnote labeled "note".`
	`.. [#label] This is the second footnote labeled "label".`
	
	If [1] is the first footnote, it will show up as “[1]”. We can refer to it again as [1]. We can also refer to it as note (an ordinary internal hyperlink reference).
	
	If [2] is the second footnote, it will show up as “[2]”. We can refer to it again as [2]. We can also refer to it as label (an ordinary internal hyperlink reference).
	
	[1] *(1, 2)* This is the first footnote labeled “note”.
	[2] *(1, 2)* This is the second footnote labeled “label”.
	
	(Siehe auch: Automatisch nummerierte Fußnoten [▶ 352])

Beispiel

Das folgende Beispiel zeigt automatisch nummerierte Fußnoten mit explizitem Fußnotenlabel.

(Im Beispielprojekt [▶ 305]:
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_NamedAutomaticallyNumberedFootnotes)

```
(*
**Autonumbered labeled footnotes**

They may be assigned 'autonumber labels' - for instance, [#first]_ and [#second]_.

--------

.. [#first] a.k.a. first_

.. [#second] a.k.a. second_
*)
```

Autonumbered labeled footnotes

They may be assigned 'autonumber labels' - for instance, [1] and [2].

---

[1] a.k.a. first
[2] a.k.a. second

13.6.2.8.6.4 Automatische Generierung von Symbolen für Fußnoten

Bei der automatischen Generierung von Symbolen für Fußnoten wird jeder Fußnote ein Symbol als Bezeichnung zugewiesen.

Fußnotenreferenz

Beschreibung	Die Fußnotenreferenz besteht aus einem Fußnotenlabel in eckigen Klammern gefolgt von einem abschließenden Unterstrich. Das Fußnotenlabel ist ein einzelnes Sternchen „*“.
Start- und Endzeichen	Startzeichen = „[“, Endzeichen = „]_“ (Siehe auch: Inline-Markup [▶ 317])

Fußnote

Beschreibung	Jede Fußnote (Fußnotenbeschriftung) besteht aus einem expliziten Markup-Start („.. “), dem von eckigen Klammern umschlossenen Fußnotenlabel und einem Leerzeichen gefolgt von eingerückten Körperelementen (Fußnoteninhalt). (Siehe auch Explizite Markup-Blöcke [▶ 316])

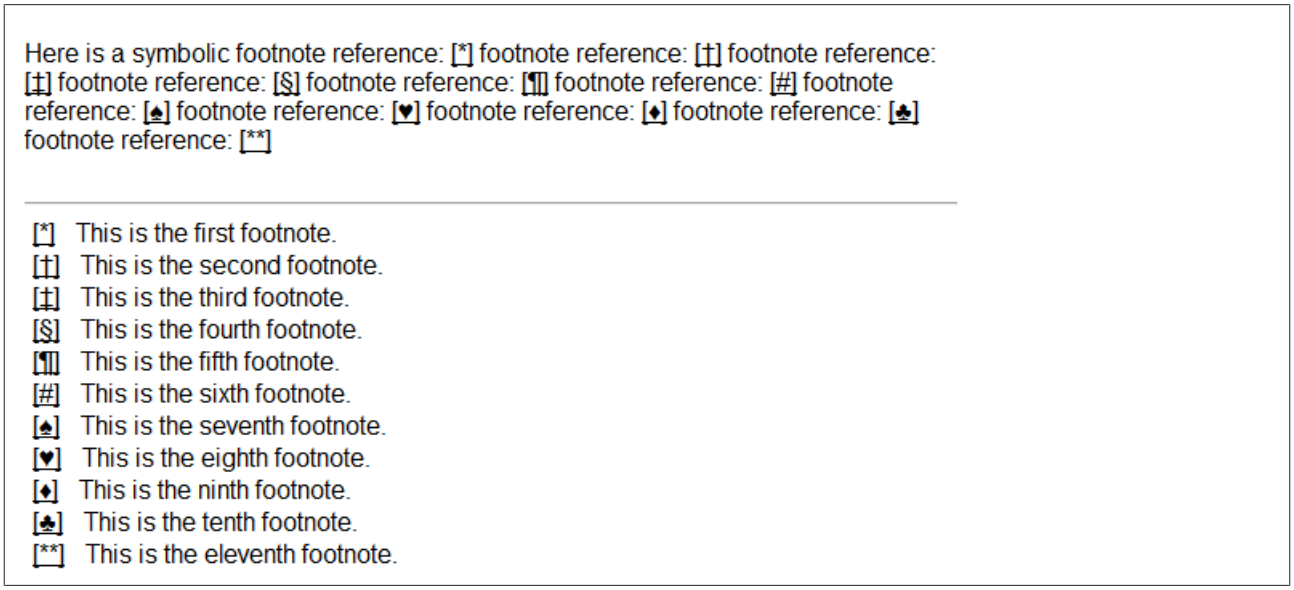
Prinzip	`+-----+--------------------+`
	`|".. "|"["*"]" footnote  |`
	`+-----+                    |`
	`|(body elements)    |`
	`+--------------------+`

Eigenschaften	• Die Anzahl der Fußnotenreferenzen muss der Anzahl der Fußnoten entsprechen. • Eine Symbolfußnote darf nicht mehrfach referenziert werden. • Eine Transformation fügt Symbole als Beschriftungen in die entsprechende Fußnoten und Fußnotenreferenzen ein. Für die Fußnoten werden folgende Symbole verwendet: ◦ Sternchen/Stern („*“) ◦ Dolch („†“) ◦ Doppeldolch („‡“) ◦ Paragraf („§“) ◦ Absatzzeichen („¶“) ◦ Nummernzeichen („#“) ◦ Pik („♠“) ◦ Herz („♥“) ◦ Karo („♦“) ◦ Kreuz („♣“) • Wenn mehr als zehn Symbole benötigt werden, wird die gleiche Reihenfolge wiederverwendet, verdoppelt und dann verdreifacht usw. („**“ usw.).

Beispiel

Das folgende Beispiel zeigt die automatische Generierung von Symbolen für Fußnoten und deren Darstellung im Bibliotheksverwalter.
(Im Beispielprojekt [▶ 305]:
B_DocuElements\Hyperlinks\Footnotes\FB_Libdoc_AutomaticallySymbolFootnotes)

```
(*
Here is a symbolic
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_ footnote reference: [*]_
footnote reference: [*]_

.. [*] This is the first footnote.
.. [*] This is the second footnote.
.. [*] This is the third footnote.
.. [*] This is the fourth footnote.
.. [*] This is the fifth footnote.
.. [*] This is the sixth footnote.
.. [*] This is the seventh footnote.
.. [*] This is the eighth footnote.
.. [*] This is the ninth footnote.
.. [*] This is the tenth footnote.
.. [*] This is the eleventh footnote.
*)
```



13.6.2.8.6.5 Manuell und automatisch nummerierte Fußnoten

Innerhalb eines Kommentars können sowohl manuelle als auch automatische nummerierte Fußnoten verwendet werden, wobei die Nummerierung der manuell erzeugten Fußnoten Priorität hat. Nur unbenutzte Fußnotenlabel werden automatisch nummerierten Fußnoten zugeordnet.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements
\Hyperlinks\Footnotes\FB_Libdoc_ManuallyAndAutomaticallyNumberedFootnotes)

```
(*
[2]_ will be "2" (manually numbered footnote),

[3]_ will be "3" (another manually numbered footnote),

[#]_ will be "4" (anonymous auto-numbered footnote), and

[#label]_ will be "1" (labeled auto-numbered).

--------
```

```
.. [2] This footnote is labeled manually, so its number is fixed.

.. [3] This footnote is also labeled manually, so its number is also fixed.

.. [#label] This autonumber-labeled footnote will be labeled "1".
   It is the first auto-numbered footnote and no other footnote
   with label "1" exists.  The order of the footnotes is used to
   determine numbering, not the order of the footnote references!

.. [#] This footnote will be labeled "4".  It is the second
   auto-numbered footnote, but footnote labels "1", "2", "3" are already used.
*)
```

[2] will be “2” (manually numbered footnote),

[3] will be “3” (another manually numbered footnote),

[4] will be “4” (anonymous auto-numbered footnote), and

[1] will be “1” (labeled auto-numbered).

---

[2] This footnote is labeled manually, so its number is fixed.
[3] This footnote is also labeled manually, so its number is also fixed.
[1] This autonumber-labeled footnote will be labeled “1”. It is the first auto-numbered footnote and no other footnote with label “1” exists. The order of the footnotes is used to determine numbering, not the order of the footnote references!
[4] This footnote will be labeled “4”. It is the second auto-numbered footnote, but footnote labels “1”, “2”, “3” are already used.

Siehe auch:

- Manuell nummerierte Fußnoten [▶ 350]
- Automatisch nummerierte Fußnoten [▶ 352]

13.6.2.8.7 Zitate

Zitate sind identisch zu Fußnoten, außer dass sie alphanumerische Bezeichnungen wie `[note]` oder `[GVR2001]` verwenden und im Bibliotheksverwalter tabellenartig dargestellt werden.

Zitate bestehen wie Hyperlinks aus zwei Teilen: Zitatreferenz [▶ 358] (Quelle) und Zitat [▶ 359] (Ziel).

Zitatreferenz

Beschreibung	Jede Zitatreferenz besteht aus einem Zitatlabel in eckigen Klammern gefolgt von einem Unterstrich. Zitatlabel sind einfache Referenznamen (einzelne Wörter, bestehend aus alphanumerischen Zeichen, keine Leerzeichen).
Start- und Endzeichen	Startzeichen = „[“, Endzeichen = „]_“ (Siehe auch: Inline-Markup [▶ 317])

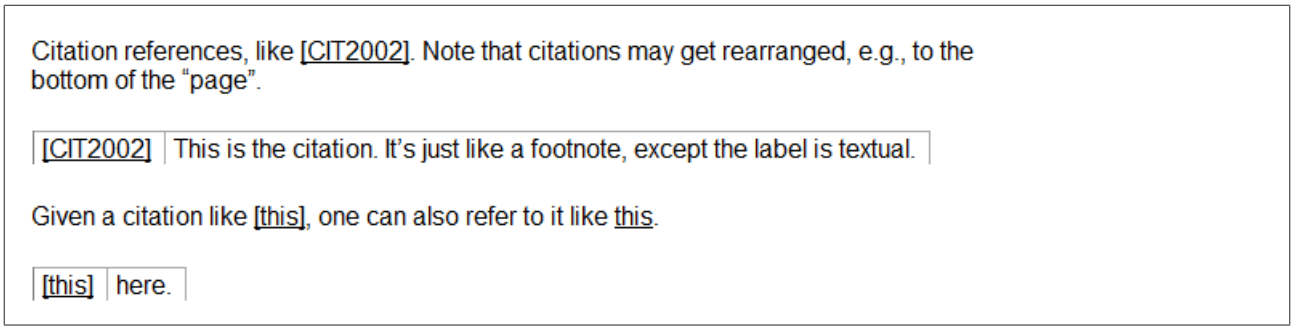
Zitat (Ziel)

Beschreibung	Jedes Zitat (Zitatbeschriftung) besteht aus einem expliziten Markup-Start („.. “), dem von eckigen Klammern umschlossenen Zitatlabel und einem Leerzeichen gefolgt von eingerückten Körperelementen (Zitatinhalt). (Siehe auch: Explizite Markup-Blöcke [▶ 316])
Prinzip	`.. [CIT]`
Eigenschaften	• Zitate werden im Gegensatz zu Fußnoten im Bibliotheksverwalter tabellenartig dargestellt. • Zwischen der Zitatbeschriftung und dem Zitatinhalt kann, aber muss keine Leerzeile vorhanden sein. • Der Zitatinhalt muss eingerückt sein (mindestens ein Leerzeichen). • Zitate können überall im Kommentar positioniert werden, nicht nur am Ende.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_Citation)

```
(*
Citation references, like [CIT2002]_. Note that citations may get rearranged, e.g., to the bottom of
 the "page".

.. [CIT2002] This is the citation.  It's just like a footnote,
   except the label is textual.

Given a citation like [this]_, one can also refer to it like this_.

.. [this] here.
*)
```



13.6.2.8.8 Link auf ein anderes Objekt

Es ist möglich, einen Link auf die Dokumentation eines anderen Bibliotheksobjekts zu setzen, welches sich ebenfalls in dieser Bibliothek befindet. Durch Anklicken dieses Links wird die Dokumentation des verlinkten Objekts bzw. Bausteins angezeigt.

Beschreibung	Die Referenz besteht aus :ref: gefolgt von einem Referenznamen, welcher in Backquotes angegeben wird: `:ref:`Objectname``

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Hyperlinks\FB_Libdoc_LinkToAnotherObject)

Im folgenden Beispielcode sind zwei Links enthalten, die jeweils auf einen anderen Funktionsbaustein verweisen. Durch Anklicken dieses Links wird die Dokumentation des verlinkten Bausteins im Bibliotheksverwalter angezeigt.

```
(*
| It is also possible to create a reference to the documentation of another object which is also
part of this library.
| For example, by clicking on this link :ref:`FB_Libdoc_InlineHyperlinks`, the documentation of this
FB will be shown.
```

```
| Or you can also create a link to :ref:`FB_Libdoc_CodeBlock` which is part of another folder.

*)
```

13.6.2.9 Substitution

Substitutionen ermöglichen es, beliebig komplexe Inline-Strukturen in den Text einzubinden und gleichzeitig die Details aus dem Textfluss herauszuhalten.

Sie bestehen aus zwei Teilen: Subsitutionsreferenz [▶ 360] und Subsitutionsdefinition [▶ 360]. Das Verarbeitungssystem ersetzt die Substitutionsreferenzen durch den Inhalt der entsprechenden Substitutionsdefinitionen.

Substitutionsreferenz

Beschreibung	Die Substitutionsreferenz besteht aus einem von vertikalen Balken eingeklammerten Referenztext. Eine Substitutionsreferenz kann gleichzeitig eine Hyperlink-Referenz durch Anhängen eines „_“ (benannt) oder „__“ (anonym) sein.
Start- und Endzeichen	Startzeichen = „|“, Endzeichen = „|“ (optional gefolgt von „_“ oder „__“) (Siehe auch: Inline-Markup [▶ 317])
Eigenschaften	• Substitutionsreferenzen werden inline durch den Inhalt der Substitutionsdefinition ersetzt. • Ein Referenztext darf nicht mit einem Leerzeichen beginnen oder enden

Substitutionsdefinition

Beschreibung	Die Substitutionsdefinition besteht aus einem expliziten Markup-Start („.. “) gefolgt von dem in vertikalen Balken umschlossenen Referenztext, einem Leerzeichen und dem Definitionsblock. Der Definitionsblock enthält eine eingebettete inline-kompatible Direktive, wie z. B. „image“ oder „replace“. (Siehe auch: Explizite Markup-Blöcke [▶ 316])

Prinzip	`+-----+-----------------------------------------+`
	`|".. "|"|"reference text"|" directive type"::" data  |`
	`+-----+directive block                                |`
	`      |                                                |`
	`      +-----------------------------------------+`

Eigenschaften	• Der Inhalt der Substitutionsdefinition ersetzt inline die Substitutionsreferenz. • Ein Referenztext darf nicht mit einem Leerzeichen beginnen oder enden.

Anwendungsfälle

Nachfolgend werden einige Anwendungsfälle für den Substitutionsmechanismus beschrieben:

- Ersatztext [▶ 360]
- Bilder [▶ 361]

Ersatztext

Der Substitutionsmechanismus kann für die einfache Textsubstitution verwendet werden. Dies kann sinnvoll sein, wenn der Ersetzungstext mehrmals im Kommentar wiederholt wird, insbesondere wenn er später geändert werden muss.

Direktiventyp: `replace`

Beispiele

(Im Beispielprojekt [▶ 305]: B_DocuElements\Substitution\FB_Libdoc_Substitution_ReplacementText)

```
(*
|RST|_ is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax.
Among others it is useful for library documentation. If the |RST| option is activated,
comments written in |RST|
 will be shown in the Documentation tab of the library manager in an attractive format.
|RST| is a little annoying to type over and over and spelling out the
bicapitalized word |RST| every time isn't really necessary for |RST| source readability.

.. |RST| replace:: reStructuredText
.. _RST: https://infosys.beckhoff.de/
*)
```

reStructuredText is an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax. Among others it is useful for library documentation. If the reStructuredText option is activated, comments written in reStructuredText will be shown in the Documentation tab of the library manager in an attractive format. reStructuredText is a little annoying to type over and over and spelling out the bicapitalized word reStructuredText every time isn't really necessary for reStructuredText source readability.

Beachten Sie den abschließenden Unterstrich bei der ersten Verwendung der Substitutionsreferenz. Dies zeigt einen Verweis auf das entsprechende Hyperlink-Ziel an. Inline-Markup werden in der Substitutionsdefinition nicht verarbeitet.

```
(*
This is a simple |substitution reference|. It will be replaced by
the processing system.

This is a |substitution and hyperlink reference|_. In
addition to being replaced, the replacement text or element will
refer to the "substitution and hyperlink reference" target.

.. |substitution reference| replace:: example of substitution
.. |substitution and hyperlink reference| replace:: combination of substitution and hyperlink
reference
.. _substitution and hyperlink reference: https://beckhoff.de/
*)
```

This is a simple example of substitution. It will be replaced by the processing system.

This is a combination of substitution and hyperlink reference. In addition to being replaced, the replacement text or element will refer to the "substitution and hyperlink reference" target.

Bilder

(Im Beispielprojekt [▶ 305]: B_DocuElements\Substitution\FB_Libdoc_Substitution_Images)

Der Substitutionsmechanismus kann auch verwendet werden, um Bilder in den Kommentartext einzubinden.

Direktiventyp: `image`

(Siehe auch: Bilder [▶ 364])

Beispiele

Substitution in Absätzen

Der Kommentartext `|Beckhoff|_` wird durch das Logo ergänzt. Der Unterstrich, der der Substitutionsreferenz folgt, zeigt einen Verweis auf ein Hyperlink-Ziel an. Durch einen Klick auf das Bild wird die Beckhoff Homepage im Bibliotheksverwalter geöffnet.

```
(*
Images are a common use for substitution references: |Beckhoff|_.

.. |Beckhoff| image:: C:\Tc3LibDocImages\SampleLib1\logo.gif
 :height: 16
```

```
 :width: 64
.. _Beckhoff: http://www.beckhoff.de/
*)
```

Images are a common use for substitution references: BECKHOFF.

```
(*
The |safety| symbol indicates a hazardous situation which,
if not avoided, could result in minor or moderate injury.

.. |safety| image:: C:\Tc3LibDocImageslogo\SampleLib1\safetysymbol.png
*)
```

The ⚠ symbol indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

Substitution in Listen oder Tabellen

```
(*
* |Run mode| Run mode
* |Stop mode| Stop mode
* |Config mode| Config mode

============= ======
Symbol        Status
============= ======
|Run mode|    Run mode
|Stop mode|   Stop mode
|Config mode| Config mode
============= ======

.. |Run mode| image:: C:\Tc3LibDocImages\SampleLib1\tc3rtrunmode.png
.. |Stop mode| image:: C:\Tc3LibDocImages\SampleLib1\tc3rtstopmode.png
.. |Config mode| image:: C:\Tc3LibDocImages\SampleLib1\tc3rtconfigmode.png
*)
```

- Run mode
- Stop mode
- Config mode

Symbol	Status
	Run mode
	Stop mode
	Config mode

13.6.2.10 Hinweiselemente

In reStructuredText werden spezifische und generische Hinweise unterschieden. Hinweise werden im Bibliotheksverwalter als Offset-Block schattiert mit einem Titel (Signalwort) dargestellt.

13.6.2.10.1 Spezifische Hinweise

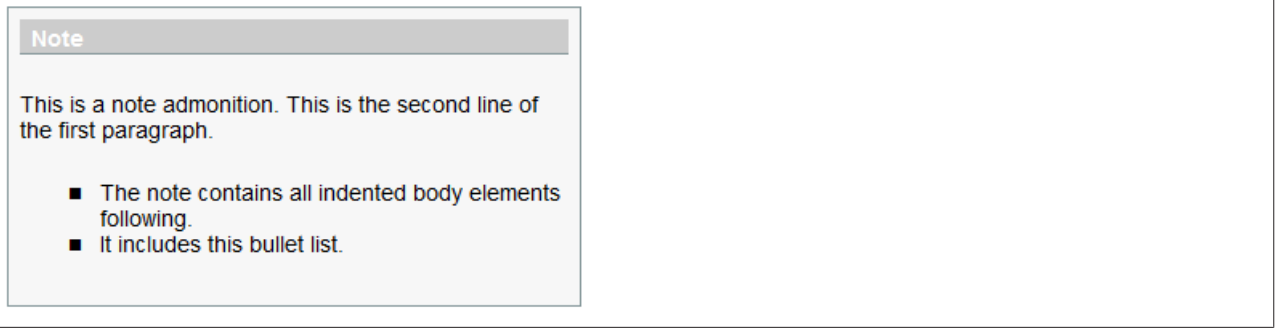
Spezifische Hinweise (u. a. Sicherheits- und Warnhinweise) können mit den Direktiven `attention`, `caution`, `danger`, `error`, `hint`, `important`, `note`, `tip` und `warning` erzeugt werden. Der Titel des Hinweisblocks wird durch die Direktive angegeben und entspricht dem Typ des Hinweises.

Beschreibung	Die Direktivenmarkierung besteht aus einem expliziten Markup-Start („.. “) gefolgt vom Typ der Direktive (z. B. `note`) und zwei Doppelpunkten. (Siehe auch: Direktiven [▶ 316])
Prinzip	`.. note::`
Eigenschaften	• Spezifische Hinweise können im Kommentar wie gewöhnliche Körperelemente eingesetzt werden und beliebige Körperelemente enthalten. • Folgende Hinweistypen sind umgesetzt: ◦ Achtung (Attention) ◦ Vorsicht (Caution) ◦ Gefahr (Danger) ◦ Fehler (Error) ◦ Hinweis (Hint) ◦ Wichtig (Important) ◦ Notiz (Note) ◦ Tipp (Tip) ◦ Warnung (Warning) • Jeder Text, der unmittelbar auf die Direktivenmarkierung auf der gleichen Zeile und/ oder eingerückt auf der folgenden Zeilen folgt, wird als Anweisungsblock (Direktiveninhalt) interpretiert. • Zwischen der Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Note elements\FB_Libdoc_SpecificNotes)

```
(*
.. note:: This is a note admonition.
   This is the second line of the first paragraph.

   - The note contains all indented body elements
     following.
   - It includes this bullet list.
*)
```



13.6.2.10.2 Generische Hinweise

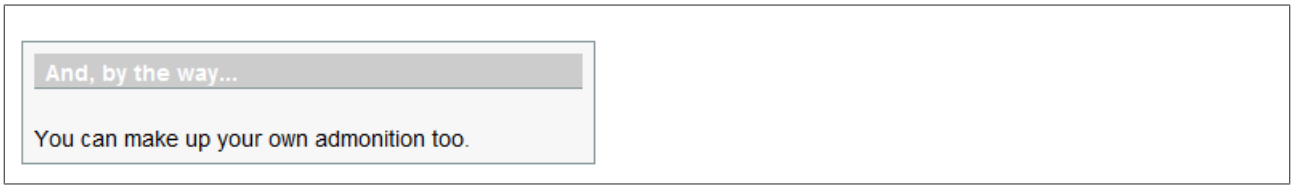
Generische Hinweise können mit der Direktive `admonition` erzeugt werden. Bei generischen Hinweisen ist der Titel und somit der Hinweistyp frei wählbar.

Beschreibung	Die Direktivenmarkierung besteht aus einem expliziten Markup-Start („.. “) gefolgt vom Typ der Direktive (`admonition`) und zwei Doppelpunkten. (Siehe auch: Direktiven [▶ 316])
Prinzip	`.. admonition::`
Eigenschaften	• Das Direktivenargument entspricht dem Hinweistitel • Zwischen der Direktivenmarkierung und dem Anweisungsblock (Direktiveninhalt) muss eine Leerzeile eingefügt werden. • Zwischen der Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig.

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Note elements\FB_Libdoc_GenericNotes)

```
(*
.. admonition:: And, by the way...

   You can make up your own admonition too.
*)
```



13.6.2.11 Bilder

Bilder können mit der Direktive `image` in den Kommentar eingebunden werden.

Beschreibung	Die Direktivenmarkierung besteht aus einem expliziten Markup-Start („.. “) gefolgt vom Typ der Direktive (`image`) und zwei Doppelpunkten. (Siehe auch: Direktiven [▶ 316])
Prinzip	`.. image::`
Eigenschaften	• Der Speicherpfad für die Bildquelldatei wird im Argument der Direktive angegeben. Wie bei Hyperlink-Zielen kann der Speicherpfad in derselben Zeile wie der explizite Markup-Start oder in einem eingerückten Textblock unmittelbar danach beginnen (ohne dazwischenliegende Leerzeilen). Der Speicherpfad kann absolut oder relativ angegeben werden und darf keine Leerzeichen enthalten. (Siehe: Absoluter oder relativer Speicherpfad als Direktivenargument [▶ 366]) • Das Bild wird entsprechend der Fensterbreite des Bibliotheksverwalters größtmöglich (=100%) angezeigt. Wird die Fensterbreite des Bibliotheksverwalters verändert, bleibt die Proportionalität des Bildes erhalten. • Zwischen der Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig.

Optionen

Optional kann der Direktivenblock eine flache Feldliste mit Bildoptionen enthalten. Folgende Optionen werden erkannt:

height: length	Höhe des Bildes. Wird verwendet, um das Bild vertikal zu skalieren. Die Breite des Bildes wird nach Möglichkeit proportional dazu angepasst. • Angabe in px (mit oder ohne Leerzeichen): ◦ Die Höhe des angezeigten Bildes entspricht immer der angegebenen Höhe des Bildes in px unabhängig von der Höhe und Breite des Bibliotheksverwalter-Fensters. Wenn die Höhe des Bibliotheksverwalter-Fensters kleiner als die angegebene Bildhöhe ist, können Sie durch Scrollen innerhalb des Bibliotheksverwalters den restlichen Teil des Bildes anzeigen. ◦ Die Bildbreite ist maximal so groß wie die Fensterbreite des Bibliotheksverwalters: Wenn das Bibliotheksverwalter-Fenster breit genug ist, wird das Bild proportional dargestellt. Ansonsten wird die Darstellung verzerrt. ◦ Die Proportionalität des Bildes ist somit abhängig von der Fensterbreite des Bibliotheksverwalters vorhanden.
width: length or percentage of the current line width	Breite des Bildes. Wird verwendet, um das Bild horizontal zu skalieren. Die Höhe des Bildes wird nach Möglichkeit proportional dazu angepasst. • Angabe in px (mit oder ohne Leerzeichen): ◦ Die Breite des angezeigten Bildes entspricht nach Möglichkeit der angegebenen Breite des Bildes in px, ist aber maximal so groß wie die Fensterbreite des Bibliotheksverwalters. D. h. falls die Breite des Bibliotheksverwalter-Fensters kleiner als die angegebene Bildbreite ist, ist das Bild nur so breit wie das Bibliotheksverwalter-Fenster. ◦ Die Höhe des Bildes wird proportional zur angegebenen Bildbreite angepasst. ◦ Die Proportionalität des Bildes ist somit abhängig von der Fensterbreite des Bibliotheksverwalters vorhanden. • Angabe in % (mit oder ohne Leerzeichen): ◦ Die Breite des angezeigten Bildes nimmt den angegebenen prozentualen Anteil der Fensterbreite des Bibliotheksverwalters ein. ◦ Die Höhe des Bildes wird proportional zur Bildbreite angepasst. ◦ Die Proportionalität des Bildes ist somit vorhanden.
align: „left“ or „right“	Ausrichtung des Bildes. Die Werte „left“ und „right“ steuern die horizontale Ausrichtung eines Bildes, sodass das Bild schweben und der Text es umfließen kann.
target: text (URI or reference name)	Macht das Bild zu einem Verweis („clickable“). Das Argument der Option kann ein URI (relativ oder absolut) oder ein Referenzname mit Unterstrich-Suffix (z. B. name_) sein.
scale: integer percentage	Einheitlicher Skalierungsfaktor des Bildes. Wird verwendet, um das Bild proportional zu skalieren. • Angabe in % (mit oder ohne Leerzeichen): ◦ Die Breite und die Höhe des angezeigten Bildes entsprechen immer dem angegebenen prozentualen Anteil der ursprünglichen Bildgröße (d. h. der Größe der Grafik, die im Dateisystem referenziert wird) unabhängig von der Breite und Höhe des Bibliotheksverwalter-Fensters. Wenn die Breite bzw. die Höhe des Bibliotheksverwalter-Fensters kleiner als die angegebene Bildbreite bzw. -höhe ist, können Sie durch Scrollen innerhalb des Bibliotheksverwalters den restlichen Teil des Bildes anzeigen. ◦ Die Proportionalität des Bildes ist somit vorhanden.

Wenn ein Bild ohne height-, width- oder scale-Option eingebunden wird, entspricht die Breite des angezeigten Bildes der Fensterbreite des Bibliotheksverwalters. Die Höhe des Bildes wird proportional zur Bildbreite angepasst. Das Bild wird somit größtmöglich dargestellt. Die Proportionalität des Bildes ist vorhanden.

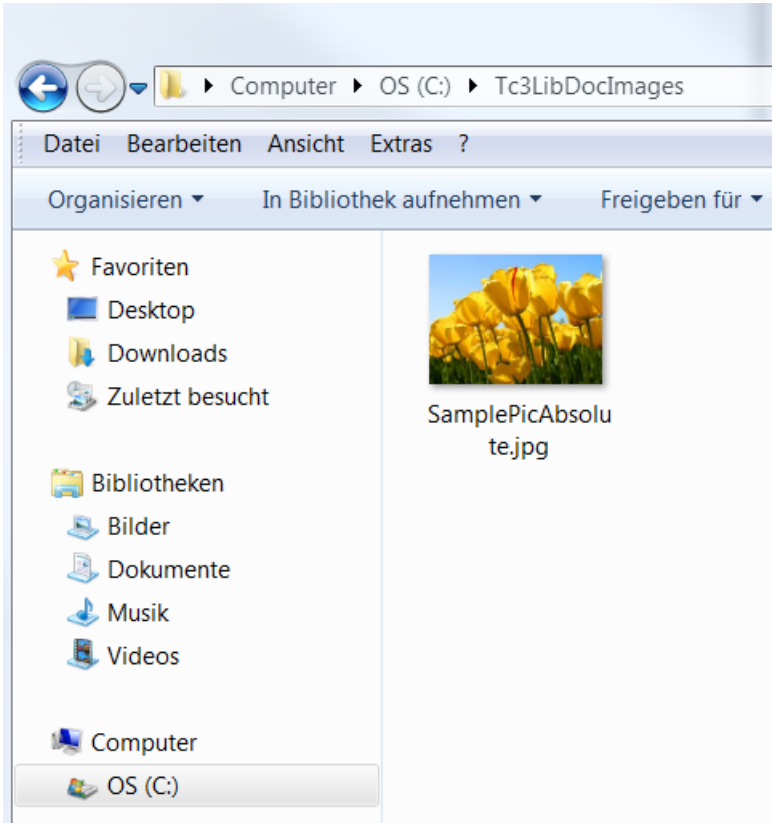
Absoluter oder relativer Speicherpfad als Direktivenargument

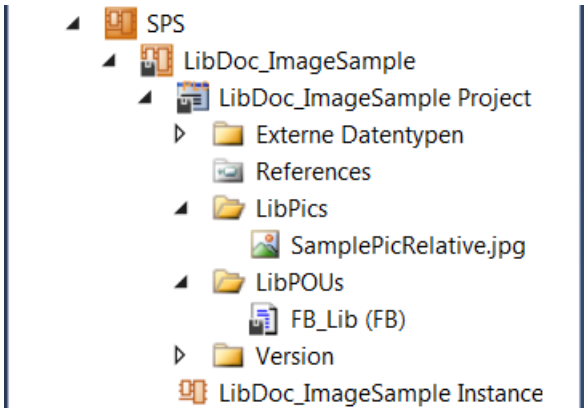
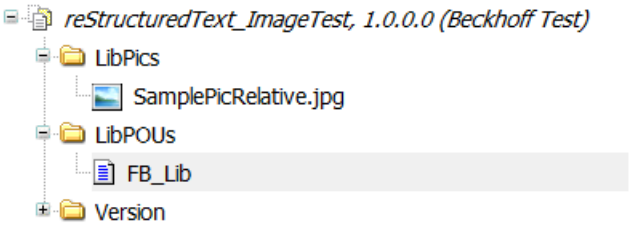
Im Argument der image-Direktive können Sie sowohl absolute als auch relative Speicherpfade angeben, um Bilder zu referenzieren.

Einen absoluten Speicherpfad geben Sie an, wenn das Bild in einem beliebigen Ordner Ihres Dateisystems gespeichert ist. Einen relativen Speicherpfad geben Sie an, wenn das Bild in das Bibliotheksprojekt eingebunden ist.

Weitergabe der Bibliotheksdatei

Wenn Sie die Bilder in das Bibliotheksprojekt einbinden und relative Speicherpfade verwenden und das Bibliothekprojekt dann als Datei weitergeben, werden die benötigten Bildinformationen direkt als Teil des Bibliotheksprojekts mitgeliefert. Wenn Sie absolute Speicherpfade verwenden, müssen die Bilder separat weitergegeben werden und bei demjenigen, der die Bibliothek verwendet, an der im Speicherpfad angegebenen Stelle im Dateisystem vorhanden sein. Letzteres ist eine statische Vorgehensweise, bei der die Organisation und Pflege der Bildinformationen im Dateisystem zusätzlichen Aufwand bedeutet.

Absoluter Speicherpfad	z. B. *C:\Tc3LibDocImages\SamplePicAbsolute.jpg* 

Relativer Speicherpfad Voraussetzung: TwinCAT 3.1.4022.22	z. B. *../../LibPics/SamplePicRelative.jpg* **Bild in das SPS-Projekt einbinden** 1. Legen Sie im SPS-Projektbaum für eine einheitliche Ablage und übersichtliche Projektstruktur einen Ordner für Bilder an (z. B. LibPics). 2. Wählen Sie im Kontextmenü des Ordners den Befehl **Hinzufügen > Vorhandenes Element**. 3. Wählen Sie im Dateisystem das Bild aus (z. B. SamplePicRelative) und bestätigen Sie den Dialog. ⇨ Das Bild wird dem SPS-Projekt hinzugefügt und kann bei der Dokumentation eines Bibliotheksobjektes in einem reStructuredText-Kommentar referenziert werden. (Siehe: Beispiele [▶ 368]) ⇨ Beim Speichern des SPS-Projekts als Bibliothek wird das Bild mit in die Bibliotheksdatei gespeichert (*.library/*.compiled-library). Wenn die Bibliothek in einem SPS-Projekt referenziert wird, wird das Bild mit im Bibliotheksverwalter anzeigt. Per Doppelklick auf das Bild im Bibliotheksverwalter kann das Bild geöffnet werden. Projektmappen-Explorer:  Bibliotheksverwalter: 

Beispiele

(Im Beispielprojekt [▶ 305]: B_DocuElements\Images\FB_Libdoc_Images)

Absoluter oder relativer Speicherpfad als Direktivenargument

```
(*
Absolute path

.. image:: C:\Tc3LibDocImages\SamplePicAbsolute.jpg
   :width: 70px

Relative path

.. image:: ../../LibPics/SamplePicRelative.jpg
   :width: 70px
*)
```

Absolute path

Relative path

Bild linksbündig

```
(*
.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
   :width: 20%

Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
*)
```

Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.
Description of the function block. Description of the function block.

Schwebendes Bild linksbündig

```
(*
.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
   :width: 20%
   :align: left

Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
*)
```

Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block.

Schwebendes Bild rechtsbündig

```
(*
.. image:: C:\Tc3LibDocImages\SampleLib1\img11.jpg
   :width: 20%
   :align: right

Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
Description of the function block. Description of the function block. Description of the function bl
ock.
*)
```

Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block. Description of the function block.

Inline-Bilder können mit einer `image`-Direktive in einer Substitutionsdefinition definiert werden (siehe Substitution [▶ 360]).

13.6.2.12 Codeblock

Die Direktive `code` ermöglicht es, Codebereiche im Kommentar darzustellen. Diese werden im Bibliotheksverwalter grau schattiert und in Konstantschrift mit farblich hervorgehobener Syntax angezeigt.

Beschreibung	Die Direktivenmarkierung besteht aus einem expliziten Markup-Start („.. “) gefolgt vom Typ der Direktive (`code`) und zwei Doppelpunkten. (Siehe auch: Direktiven [▶ 316])
Prinzip	`.. code::`
Eigenschaften	• Zwischen der Direktive und einem vorangehenden Textkörperelement (z. B. einem Absatz mit Text) ist eine Leerzeile notwendig.

Optionen

Optional kann ein Direktivenblock eine flache Liste mit Code-Optionen enthalten. Folgende Option wird erkannt:

Number-lines : [start line number]	Jeder Zeile wird eine Zeilennummer vorangestellt. Das optionale Argument ist die Nummer der ersten Zeile. Standardwert ist 1. Die Syntax wird in diesem Fall nicht farblich hervorgehoben.

Beispiele

(Im Beispielprojekt [▶ 305]: B_DocuElements\Code Block\FB_Libdoc_CodeBlock)

Codeblock mit farblich hervorgehobener Syntax

```
(*
.. code::

    // Attempts to return the value of a boolean property.

    FUNCTION GetBooleanProperty : BOOL
    VAR_INPUT
        sKey: STRING;
    END_VAR

    // This structure defines a special profile.

    TYPE ST_Profile :
    STRUCT
```

```
        nId     : INT := -1;
        sBuffer : STRING(255) := 'Hello';
    END_STRUCT
    END_TYPE
*)
```

```
// Attempts to return the value of a boolean property.

FUNCTION GetBooleanProperty : BOOL
VAR_INPUT
    sKey: STRING;
END_VAR

// This structure defines a special profile.

TYPE ST_Profile :
STRUCT
    nId     : INT := -1;
    sBuffer : STRING(255) := 'Hello';
END_STRUCT
END_TYPE
```

Codeblock mit Zeilennummerierung

```
(*
.. code::
   :number-lines: 1

    // Attempts to return the value of a boolean property.

    FUNCTION GetBooleanProperty : BOOL
    VAR_INPUT
        sKey: STRING;
    END_VAR

    // This structure defines a special profile.

    TYPE ST_Profile :
    STRUCT
        nId     : INT := -1;
        sBuffer : STRING(255) := 'Hello';
    END_STRUCT
    END_TYPE
*)
```

```
 1  // Attempts to return the value of a boolean property.
 2
 3  FUNCTION GetBooleanProperty : BOOL
 4  VAR_INPUT
 5      sKey: STRING;
 6  END_VAR
 7
 8  // This structure defines a special profile.
 9
10  TYPE ST_Profile :
11  STRUCT
12      nId     : INT := -1;
13      sBuffer : STRING(255) := 'Hello';
14  END_STRUCT
15  END_TYPE
```

13.6.2.13 Interne Kommentare

Interne Kommentare werden bei der Ausgabe entfernt und nicht im Bibliotheksverwalter angezeigt.

Beschreibung	Ein interner Kommentar besteht aus einem expliziten Markup-Start („.. “) gefolgt von beliebig eingerücktem Text. (Siehe auch: Explizite Markup-Blöcke [▶ 316])

Prinzip	`+-----+----------------+`
	`|".. "|comment |`
	`+-----+block |`
	`| |`
	`+----------------+`

Eigenschaften	• Der Kommentartext kann in der gleichen Zeile wie der explizite Markup-Start oder eingerückt in der nachfolgenden Zeile stehen.
	`.. This is a comment`
	
	`..`
	`This is a comment`

Beispiel

(Im Beispielprojekt [▶ 305]: B_DocuElements\Comments\FB_Libdoc_InternalComments)

```
(*
This is the first paragraph.

.. Comments begin with two dots and one space. Anything can
   follow, except for the syntax of footnotes/citations,
   hyperlink targets, directives, or substitution definitions.

Since comments are not shown, this paragraph follows the previous paragraph directly.
*)
```

This is the first paragraph.

Since comments are not shown, this paragraph follows the previous paragraph directly.

13.6.2.14 Schriftstil

Inline-Markup ermöglichen es, Wörter oder Phrasen fett, kursiv oder in Konstantschrift im Bibliotheksverwalter darzustellen.
(Im Beispielprojekt [▶ 305]: B_DocuElements\Font style\FB_Libdoc_FontStyle)

Hervorgehobener Text (kursiv)

Text, der von einzelnen Sternzeichen eingeschlossen ist, wird kursiv hervorgehoben.

Start- und Endzeichen	Startzeichen = Endzeichen = „*“
Beispiel	`This is *emphasized text*.` This is *emphasized text*.

Stark hervorgehobener Text (fett)

Text, der von Doppelsternen eingeschlossen ist, wird fett hervorgehoben.

Star- und Endzeichen	Startzeichen = Endzeichen = „**“.
Beispiel	`This is **strong text**.` This is **strong text**.

Inline-Literale (Konstantschrift)

Text, der von doppelten Anführungszeichen eingeschlossenen ist, wird in Konstantschrift (Maschinenschrift, Monospaced-Schrift) dargestellt.

Inline-Literale können für kurze Codebereiche oder Inline-Code verwendet werden.

Start- und Endzeichen	Startzeichen = Endzeichen = „``“
Beispiel	`This text is an example of ``inline literals``.` This text is an example of `inline literals`.

13.7 Weitere Befehle und Dialoge

13.7.1 Befehl Bibliothek hinzufügen

Funktion: Der Befehl öffnet den Dialog **Bibliothek hinzufügen**. In diesem Dialog können Sie Bibliotheksreferenzen in Form eines Platzhalters zum Bibliotheksverwalter hinzufügen und so in Ihre Anwendung einbinden.

Aufruf: Kontextmenü des Objekts **References** im SPS-Projektbaum

Voraussetzung: Die Bibliothek ist auf dem lokalen System im Bibliotheksrepository [▶ 285] installiert.

Platzhalter vs. Bibliothek

Durch die Verwendung von Bibliotheksplatzhaltern ist die Anpassung von verwendeten Bibliotheksversionen mit äußerst geringem Aufwand verbunden, sodass sich der Engineering-Prozess von Projekten und Bibliotheken sehr flexibel gestaltet. Wenn eine Bibliotheksreferenz in ein Projekt oder in ein Bibliotheksprojekt eingebunden wird, sollte daher möglichst ein Platzhalter anstelle einer Bibliothek eingesetzt werden. Das bedeutet, dass innerhalb des SPS-Projekts keine konkrete Bibliothek, sondern ein Platzhalter eingebunden wird. Dem Platzhalter wird dann eine „reelle“ Bibliothek zugewiesen.

Weitere Informationen finden Sie unter Bibliotheksplatzhalter [▶ 293].

Aufgrund der Vorteile eines Platzhalters wird die Verwendung vom Befehl Bibliothek hinzufügen [▶ 373] gegenüber dem Befehl Bibliothek ohne Platzhalterauflösung hinzufügen [▶ 374] empfohlen.

Dialog Bibliothek hinzufügen

Über den Befehl **Bibliothek hinzufügen** öffnet sich ein Dialog, über den eine Bibliotheksreferenz standardmäßig als Platzhalter mit dem dazugehörigen Platzhalternamen ins Projekt eingebunden wird. Voraussetzung hierfür ist, dass bei der Bibliothekserstellung [▶ 282] ein Platzhaltername vergeben wurde – explizit oder implizit (Falls kein expliziter Platzhaltername bei der Bibliothekserstellung eingetragen wird, wird standardmäßig der Bibliothekstitel als Platzhaltername eingetragen). Falls für die einzubindende Bibliothek kein standardmäßiger Platzhaltername vorhanden sein sollte – was beispielsweise für ältere Bibliotheken der Fall sein kann, für die kein expliziter Platzhaltername vergeben wurde – wird die Bibliotheksreferenz als Bibliothek eingebunden.

Die über diesen Dialog hinzugefügte Bibliotheksreferenz wird als Platzhalter ins Projekt eingebunden, wobei der Platzhalter standardmäßig als „immer neueste Version“ („*“), also ohne feste Version, aufgelöst wird. Die Versionsauflösung kann über den Platzhalter-Dialog [▶ 294] oder über das Eigenschaftenfenster [▶ 377] angepasst werden.

In der Zeile oberhalb der Bibliotheksliste können Sie nach Bibliotheksnamen oder Bibliotheksbausteinen suchen, indem Sie eine entsprechende Zeichenfolge eintippen. Per Doppelklick auf ein Suchergebnis oder per Fokussieren eines Suchergebnisses + [OK] wird der zu dem Suchergebnis gehörende Platzhalter ins Projekt bzw. in den Bibliotheksverwalter [▶ 289] eingebunden.	
Bibliothek	Liste aller im Bibliotheksrepository installierten Bibliotheken. Sie können die Art der Auflistung über die Schaltflächen oben rechts im Dialog ändern: • nach Bibliothekskategorien („Kategorienansicht“): Bei der Sortierung nach Bibliothekskategorien erscheinen die Kategorien als Knoten. Ein Klick auf einen Knoten öffnet die Liste der zugehörigen Bibliotheken bzw. Unterkategorien. • alphabetisch nach Bibliothekstiteln („Listenansicht“) Sie können den gewünschten Platzhalter per Doppelklick oder per Fokussieren + [OK] ins Projekt bzw. in den Bibliotheksverwalter [▶ 289] einbinden.
Firma	Hersteller der Bibliothek

Siehe auch:

- Bibliotheksplatzhalter [▶ 293]
- Bibliotheksrepository [▶ 285]
- TwinCAT 3 User Interface\ Referenz Benutzeroberfläche\ Kontextmenü SPS-Projekt
  - Befehl Projektbibliotheken installieren-
  - Befehl Projektbibliotheken installieren (Unbekannte Versionen)
  - Befehl Projektbibliotheksordner aktualisieren

Sehen Sie dazu auch

- Befehl Bibliothek ohne Platzhalterauflösung hinzufügen [▶ 374]
- Bibliothekserstellung [▶ 282]
- Bibliotheksverwalter [▶ 289]
- Befehl Eigenschaften [▶ 377]
- Platzhalter [▶ 294]

13.7.2 Befehl Bibliothek ohne Platzhalterauflösung hinzufügen

Funktion: Der Befehl öffnet den Dialog **Bibliothek suchen**. In diesem Dialog können Sie Bibliotheksreferenzen in Form einer Bibliothek zum Bibliotheksverwalter hinzufügen und so in Ihre Anwendung einbinden.

Aufruf: Kontextmenü des Objekts **References** im SPS-Projektbaum

Voraussetzung: Die Bibliothek ist auf dem lokalen System im Bibliotheksrepository [▶ 285] installiert.

Platzhalter vs. Bibliothek

Durch die Verwendung von Bibliotheksplatzhaltern ist die Anpassung von verwendeten Bibliotheksversionen mit äußerst geringem Aufwand verbunden, sodass sich der Engineering-Prozess von Projekten und Bibliotheken sehr flexibel gestaltet. Wenn eine Bibliotheksreferenz in ein Projekt oder in ein Bibliotheksprojekt eingebunden wird, sollte daher möglichst ein Platzhalter anstelle einer Bibliothek eingesetzt werden. Das bedeutet, dass innerhalb des SPS-Projekts keine konkrete Bibliothek, sondern ein Platzhalter eingebunden wird. Dem Platzhalter wird dann eine „reelle“ Bibliothek zugewiesen.

Weitere Informationen finden Sie unter Bibliotheksplatzhalter [▶ 293].

Aufgrund der Vorteile eines Platzhalters wird die Verwendung vom Befehl Bibliothek hinzufügen [▶ 373] gegenüber dem Befehl Bibliothek ohne Platzhalterauflösung hinzufügen [▶ 374] empfohlen.

Siehe auch:

- Befehl Bibliothek hinzufügen [▶ 373]

Dialog Bibliothek suchen

Über den Befehl **Bibliothek ohne Platzhalterauflösung hinzufügen** öffnet sich ein Dialog, über den eine Bibliotheksreferenz als Bibliothek und nicht als Platzhalter ins Projekt eingebunden wird.

In der Zeile oberhalb der Bibliotheksliste können Sie nach Bibliotheksnamen oder Bibliotheksbausteinen suchen, indem Sie eine entsprechende Zeichenfolge eintippen. Per Doppelklick auf ein Suchergebnis oder per Fokussieren eines Suchergebnisses + [OK] wird die zu dem Suchergebnis gehörende Bibliothek ins Projekt bzw. in den Bibliotheksverwalter [▶ 289] eingebunden.	
Firma	Filterung der Liste nach Hersteller.
Gruppieren nach Kategorie	☑ : Darstellung der Bibliotheken in einer Baumstruktur in Kategorien gruppiert. Bei der Sortierung nach Bibliothekskategorien erscheinen die Kategorien als Knoten. Ein Klick auf einen Knoten öffnet die Liste der zugehörigen Bibliotheken bzw. Unterkategorien. ☐ : Darstellung der Bibliotheken alphabetisch in einer flachen Struktur.
Alle Versionen anzeigen	☑ : Anzeige aller Versionen der Bibliotheken. Versionsangabe „*“ bedeutet aktuellste im Repository verfügbare Version. ☐ : Anzeige nur der aktuellsten Versionen der Bibliotheken. In dieser Darstellung ist eine Mehrfachauswahl von Bibliotheken möglich: Halten Sie die Taste **[Umschalt]** gedrückt, während Sie die Einträge auswählen.
Details	Öffnet den Dialog Details [▶ 376].
Bibliotheksrepository	Öffnet den Dialog Bibliotheksrepository [▶ 285]. Wenn Sie eine Bibliothek einfügen möchten, die noch nicht auf dem lokalen System installiert ist, können Sie über diese Schaltfläche direkt das Bibliotheksrepository öffnen, um die Installation durchzuführen.

Siehe auch:

- Bibliotheksplatzhalter [▶ 293]
- Bibliotheksrepository [▶ 285]
- TwinCAT 3 User Interface\ Referenz Benutzeroberfläche\ Kontextmenü SPS-Projekt
  - Befehl Projektbibliotheken installieren-
  - Befehl Projektbibliotheken installieren (Unbekannte Versionen)
  - Befehl Projektbibliotheksordner aktualisieren

Sehen Sie dazu auch

- Bibliothekserstellung [▶ 282]
- Bibliotheksverwalter [▶ 289]
- Befehl Eigenschaften [▶ 377]
- Platzhalter [▶ 294]
- Bibliothekserstellung [▶ 282]
- Platzhalter [▶ 294]
- Befehl Eigenschaften [▶ 377]

13.7.3 Befehl Bibliothek erneut laden

Funktion: Der Befehl versucht, die selektierte Bibliothek erneut zu laden.

Aufruf:

- Kontextmenü der Bibliothek im SPS-Projektbaum
- Schaltfläche im Bibliotheksverwalter [▶ 289]

Voraussetzung: Das Laden einer Bibliothek beim Öffnen eines Projekts ist fehlgeschlagen. Die Bibliothek, deren Laden fehlgeschlagen ist, ist im SPS-Projektbaum bzw. im Editor des Bibliotheksverwalters selektiert.

Wenn eine Bibliothek aus irgendeinem Grund nicht im definierten Repository-Pfad bereitsteht, wenn Sie ein Projekt öffnen, gibt TwinCAT eine entsprechende Fehlermeldung aus. Nachdem Sie den Fehler behoben haben und die Bibliothek wieder ordnungsgemäß verfügbar ist, können Sie mit dem Befehl diese Bibliothek nachladen, ohne dass Sie das Projekt verlassen müssen.

13.7.4 Befehl Bibliothek entfernen

Funktion: Der Befehl entfernt die selektierte Bibliothek aus dem Bibliotheksverwalter.

Aufruf:

- Kontextmenü der Bibliothek im SPS-Projektbaum
- Schaltfläche im Bibliotheksverwalter [▶ 289] (Symbol: )

Voraussetzung: Die Bibliothek ist im Bibliotheksverwalter vorhanden.

Siehe auch:

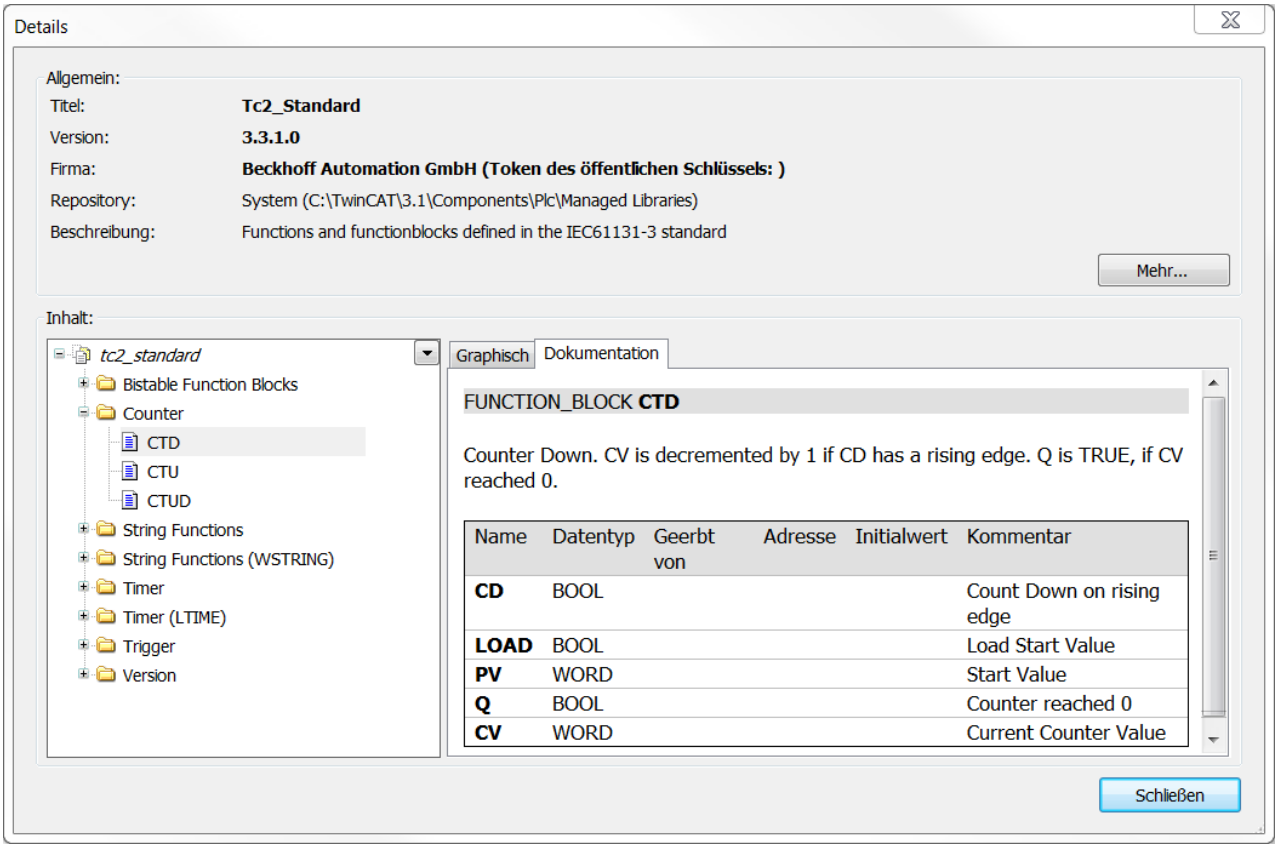
- TwinCAT 3 User Interface\ Referenz Benutzeroberfläche\ Kontextmenü SPS-Projekt
  - Befehl Projektbibliotheken installieren-
  - Befehl Projektbibliotheken installieren (Unbekannte Versionen)
  - Befehl Projektbibliotheksordner aktualisieren

13.7.5 Befehl Details

Funktion: Der Befehl öffnet für die ausgewählte Version einer Bibliothek den Dialog **Details** mit Informationen zu der Bibliothek.

Aufruf:

- Kontextmenü der Bibliothek im SPS-Projektbaum
- Schaltfläche im Bibliotheksrepository [▶ 285]
- Schaltfläche im Bibliotheksverwalter [▶ 289] (Symbol: )



Über die Schaltfläche **Mehr…** erhalten Sie außerdem folgende Informationen:

- Größe: Angabe in Bytes
- Erzeugt: Erstellungsdatum
- Geändert: Datum der letzten Änderung
- Letzter Zugriff: Datum
- Attribute
- Eigenschaften

13.7.6 Befehl Abhängigkeiten

Funktion: Der Befehl öffnet für die ausgewählte Version einer Bibliothek den Dialog **Abhängigkeiten**, der die Abhängigkeiten der selektierten Bibliothek anzeigt.

Aufruf:

- Kontextmenü der Bibliothek im SPS-Projektbaum
- Schaltfläche im Bibliotheksrepository [▶ 285]

Wenn innerhalb der selektierten Bibliothek andere Bibliotheken eingebunden bzw. referenziert sind, werden diese im Dialog aufgelistet. Für jede Bibliotheksreferenz werden Titel, Version und Firma angezeigt. Referenzen, die über Platzhalter funktionieren, sind mit der Syntax #<Platzhaltername> dargestellt.

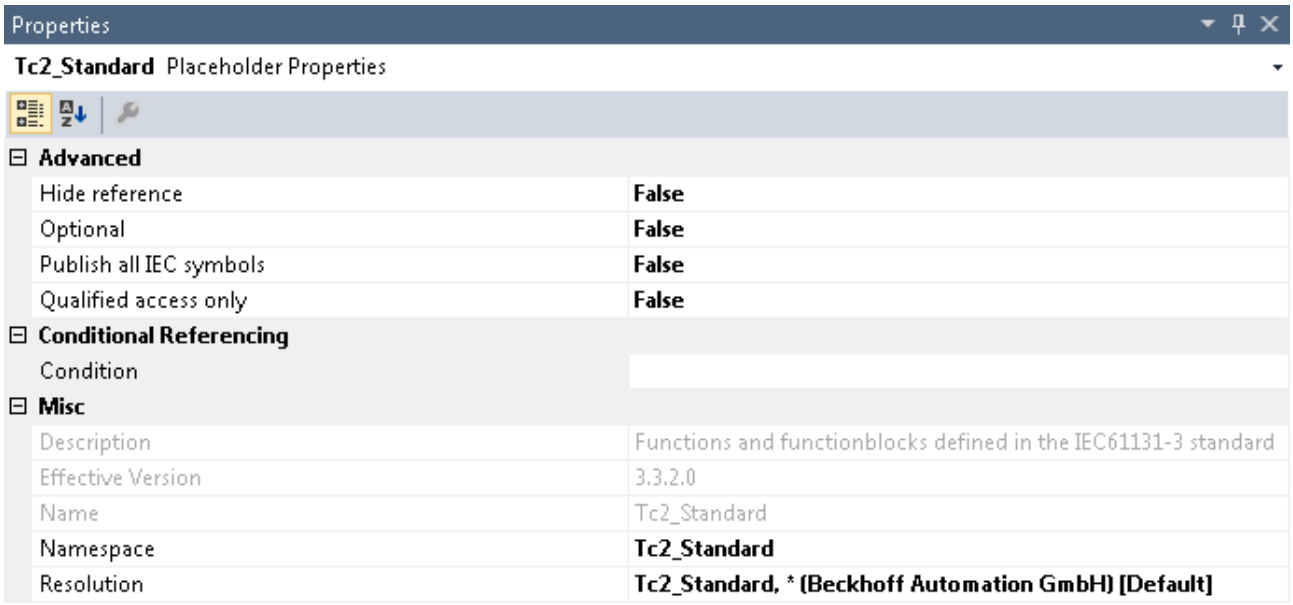
13.7.7 Befehl Eigenschaften

Funktion: Der Befehl aktiviert die Ansicht **Eigenschaften** für die selektierte Bibliothek zum Konfigurieren der Einstellungen.

Aufruf:

- Kontextmenü der Bibliothek im SPS-Projektbaum
- Bei bereits geöffneter **Eigenschaftenfenster**: Bibliothek im SPS-Projektbaum selektieren

Ansicht Eigenschaften



Erweitert

Die folgenden Einstellungen sind von Interesse, sobald die Bibliothek in einer anderen Bibliothek eingebunden (referenziert) wird. Standardmäßig sind sie deaktiviert:	
Hide reference	TRUE: Wenn das aktuelle Projekt als Bibliothek in einem anderen Projekt referenziert wird, wird diese Bibliotheksreferenz im Abhängigkeitsbaum und damit im Bibliotheksverwalter nicht angezeigt. Somit können Sie „versteckte“ Bibliotheken einfügen. Im Fall von Übersetzungsfehlern, die auf Bibliotheksfehler zurückgehen, kann das Finden der Ursache dadurch möglicherweise schwierig sein. FALSE: Wenn das aktuelle Projekt als Bibliothek in einem anderen Projekt referenziert wird, wird die Bibliotheksreferenz im Abhängigkeitsbaum und damit im Bibliotheksverwalter angezeigt.
Optional	TRUE: Die selektierte Bibliothek wird als optional behandelt. Beim Laden eines Projektes, das die Bibliothek referenziert, wird kein Fehler ausgegeben, auch wenn die Bibliothek im Bibliotheksrepository **nicht** vorhanden ist.
Publish all IEC symbols	TRUE: Wenn das aktuelle Projekt später als Bibliothek in eine andere eingebunden wird, werden die IEC-Symbole der ausgewählten Bibliotheksreferenz so in dieses Projekt veröffentlicht, als wenn die ausgewählte Bibliotheksreferenz direkt in dem Projekt eingebunden wäre. Sie können die Bausteine der Bibliothek eindeutig ansprechen, wenn Sie den entsprechenden Namensraum-Pfad verwenden. Dieser setzt sich aus dem Namensraum der „Vater“-Bibliothek und dem eigenen Namensraum zusammen und wird dem Bausteinnamen vorangestellt. Die Option sollten Sie nur dann aktivieren, wenn Sie eine sogenannte „Container-Bibliothek“ erstellen wollen. Eine Container-Bibliothek ist eine Bibliothek, die keine eigenen Bausteine definiert, sondern nur andere Bibliotheken referenziert, um eine Art Bibliotheksbündel zu erstellen. Dies kann sinnvoll sein, wenn Sie mehrere Bibliotheken gleichzeitig in ein Projekt einbinden müssen. In diesem Fall ist es allerdings wünschenswert, dass Sie die einzelnen Bibliotheken des Pakets toplevel im Bibliotheksverwalter des Projekts platzieren, um die Zugriffspfade für die Bibliotheksbausteine zu verkürzen. Eben dies können Sie durch Aktiveren der vorliegenden Option vorausblickend erzielen. FALSE: Wenn das aktuelle Projekt als Bibliothek in einem anderen Projekt referenziert wird, können die Bausteine der in den Eigenschaften angezeigten Bibliotheksreferenz eindeutig angesprochen werden, indem Sie den entsprechenden Namensraum verwenden. Der Namensraum setzt sich aus dem Namensraum des aktuellen Bibliotheksprojekts und dem Namensraum der angewählten Bibliotheksreferenz zusammen.
Qualified access only	TRUE: Sie können auf die Bausteine und Variablen der Bibliothek im Projekt nur mit vorangestelltem Namensraum-Präfix zugegriffen.

Bedingtes Referenzieren

Condition	Hier können Sie Einträge hinzufügen, die mit den im SPS-Projekt gesetzten Compilerdefinitionen verglichen werden. Wenn mindestens einer der Einträge mit einer der Definitionen übereinstimmt, wird die Bibliothek aktiv referenziert. Wenn kein Eintrag mit einer der Definitionen übereinstimmt, wird die Bibliothek deaktiviert und ausgegraut dargestellt. Falls Sie mehrere Einträge vornehmen wollen, können Sie diese mit einem Komma voneinander trennen. Falls kein Eintrag vorhanden ist, hat diese Einstellung keine Auswirkung. Beispiel: def1,def2,def3

Sonstiges

Hier können Sie (neu) definieren, welche Version der Bibliothek im Projekt verwendet wird, falls es sich um keinen Bibliotheksplatzhalter handelt, oder Sie können die Platzhalterauflösung bearbeiten. Verwenden Sie möglichst keine festen Bibliotheken, sondern referenzieren Sie die Bibliotheken über Platzhalter.	
Beschreibung	Bibliotheksbeschreibung
Effektive Version	Effektive Version der Bibliothek, auf den der Platzhalter referenziert. Dieses Feld wird angezeigt, wenn in dem Feld **Auflösung** die Einstellung „immer neueste“/„*“ konfiguriert ist. Dadurch wird dem Anwender trotz der allgemeinen „immer neuesten“ Auflösung die tatsächlich verwendete Bibliotheksversion angezeigt.
Name	Bibliotheksname
Namensraum	Anzeige des aktuellen Namensraums. Standardmäßig identisch mit dem Bibliotheksnamen, außer Sie haben beim Erstellen der Bibliothek explizit einen anderen Standardnamensraum in den Projektinformationen definiert. Sie können den Namensraum für das lokale Projekt hier in der Ansicht „Eigenschaften“ ändern.
Auflösung	Wenn Sie einen Bibliotheksplatzhalter im Bibliotheksverwalter ausgewählt haben, enthält dieses Feld den Namen und die Version der Bibliothek, die den Platzhalter ersetzen soll. An dieser Stelle wird der Platzhalter somit als eine „reelle“ Bibliothek und dabei entweder als eine bestimmte Version der Bibliothek (z.B. „1.0.0.0“) oder als „immer neueste“/„*“ aufgelöst. Wünschen Sie eine bestimmte Version der Bibiothek, wählen Sie diese aus der Liste und TwinCAT verwendet genau diese Version. Wünschen Sie die „immer neueste“/„*“ Version der Bibliothek, wählen Sie dies aus der Liste und TwinCAT verwendet immer die neueste im Bibliotheksrepository gefundene Version der Bibliothek. Dadurch können sich die tatsächlich verwendeten Bibliotheksmodule ändern, wenn eine neuere Version der Bibliothek verfügbar ist.

Siehe auch:

- Bibliotheken verwenden [▶ 278]
- Platzhalter [▶ 294]

13.7.8 Befehl Effektive Version verwenden

Funktion: Der Befehl setzt, je nachdem ob das Objekt **References** oder eine einzelne Bibliothek im SPS-Projektbaum selektiert ist, alle im Projekt verfügbaren Platzhalter und Bibliotheken oder die ausgewählte Bibliothek bzw. den ausgewählten Platzhalter auf eine effektive Version.

Aufruf:

- Kontextmenü des Objekts **References** im SPS-Projektbaum, um alle im Projekt verfügbaren Platzhalter und Bibliotheken auf eine effektive Version zu setzen
- Kontextmenü der Bibliothek im SPS-Projektbaum, um die ausgewählte Bibliothek bzw. den ausgewählten Platzhalter auf eine effektive Version zu setzen

Wenn Sie den Befehl auf alle im Projekt verfügbaren Platzhalter und Bibliotheken anwenden, werden auch die nicht in den Referenzen aufgelisteten intern verwendeten Bibliotheken auf eine effektive Version gesetzt.

Beispiel: Wenn eine Bibliothek beispielsweise zuvor als „immer neueste“/„*“ verwendet wurde und dabei die Bibliotheksversion 3.3.10.0 zum Einsatz kam, wird die Bibliothek über den Befehl **Effektive Version verwenden** nicht mehr als „immer neueste“, sondern als feste Version 3.3.10.0 referenziert.

13.7.9 Befehl Immer neueste Version verwenden

Funktion: Der Befehl setzt, je nachdem ob das Objekt **References** oder eine einzelne Bibliothek im SPS-Projektbaum selektiert ist, alle im Projekt verfügbaren Platzhalter und Bibliotheken oder die ausgewählte Bibliothek bzw. den ausgewählten Platzhalter auf die „immer neueste“ Version („*“). Dies bedeutet, dass TwinCAT immer die neueste im Bibliotheksrepository gefundene Version der zugehörigen Bibliothek verwendet.

Aufruf:

- Kontextmenü des Objekts **References** im SPS-Projektbaum, um alle im Projekt verfügbaren Platzhalter und Bibliotheken auf die „immer neueste“ Version zu setzen
- Kontextmenü der Bibliothek im SPS-Projektbaum, um die ausgewählte Bibliothek bzw. den ausgewählten Platzhalter auf die „immer neueste“ Version zu setzen

Wenn Sie den Befehl auf alle im Projekt verfügbaren Platzhalter und Bibliotheken anwenden, werden auch die nicht in den Referenzen aufgelisteten intern verwendeten Bibliotheken auf die immer neueste Version (*) gesetzt.

Beispiel: In einem Projekt wird die Bibliothek „Lib1“ als „immer neueste“ („*“) verwendet. In dem Bibliotheksrepository, das Entwicklungsrechner A verwendet, ist die Bibliothek in den Versionen 3.0.0.0 und 3.1.2.0 installiert. In dem Bibliotheksrepository von Entwicklungsrechner B ist die Bibliothek in der Version 3.0.1.0 installiert.
Wenn das beschriebene Projekt auf Entwicklungsrechner A geöffnet ist, wird die Version 3.1.2.0 von „Lib1“ referenziert. Wird dasselbe Projekt auf Entwicklungsrechner B genutzt, wird die Bibliotheksversion 3.0.1.0 verwendet.

14 Multitask-Datenzugriffs-Synchronisation in der SPS

Wenn von mehreren Tasks auf dieselben Daten zugegriffen wird, kann es je nach Task-/Echtzeitkonfiguration vorkommen, dass die Tasks gleichzeitig auf dieselben Daten zugreifen. Wenn die Daten dabei von mindestens einer der Tasks geschrieben werden, können die Daten während oder nach einer Änderung einen inkonsistenten Zustand haben. Um dies zu verhindern, müssen alle konkurrierenden Zugriffe synchronisiert werden, sodass zu einem Zeitpunkt nur von höchstens einer Task auf die gemeinsam genutzten Daten zugegriffen werden kann.

Zu diesen konkurrierenden Zugriffen aus mehreren Tasks, bei denen eine Synchronisation nötig ist, gehören beispielsweise die folgenden Fälle:

- Direkter Zugriff auf globale oder andere nicht-temporäre Variablen, beispielsweise mittels Operatoren
- Indirekter Zugriff auf globale oder andere nicht-temporäre Variablen, beispielsweise innerhalb von Funktionen, Methoden oder anderen POU-Aufrufen (z. B. besonders häufig, wenn eine Funktionsbausteininstanz global instanziiert ist)

Kurz: Wenn von mehreren Tasks auf dieselben Daten zugegriffen wird und bei mindestens einem dieser Zugriffe die Daten geschrieben werden, müssen alle lesenden und schreibenden Zugriffe synchronisiert werden. Dies gilt unabhängig davon, ob die Tasks auf einem oder mehreren CPU Kernen laufen.

⚠ WARNUNG

Inkonsistenzen und weitere Gefahren durch ungesicherten Datenzugriff

Werden konkurrierende Zugriffe nicht synchronisiert, so besteht die Gefahr eines inkonsistenten oder ungültigen Datensatzes. Je nachdem wie die Daten im weiteren Programmverlauf genutzt werden, kann dies ein Fehlverhalten des Programms, eine ungewünschte Achsbewegung oder auch den plötzlichen Programmstillstand zur Folge haben. Abhängig von der gesteuerten Anlage können Schäden an Anlage und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

Um ein Gefühl für die Notwendigkeit der Zugriffs-Synchronisation zu erhalten, finden Sie bei den Beispielprogrammen zu den MUTEX-Verfahren jeweils Funktionstests mit entsprechender Erläuterung.

Synchronisationsmöglichkeiten

Zur Synchronisation der Zugriffe stehen u. a. die folgenden Möglichkeiten zur Verfügung:

- Mutex-Verfahren (TestAndSet, FB_IecCriticalSection) zum Absichern von kritischen Bereichen [▶ 383]
  - Die Anzahl der kritischen Bereiche ist immer möglichst klein zu halten.
  - Die Länge der kritischen Bereiche ist kurz zu halten.
  - Bei vergleichsweise kurzen kritischen Bereichen empfiehlt sich meist die Verwendung von FB_IecCriticalSection [▶ 388].
  - Bei vergleichsweise langen kritischen Bereichen empfiehlt sich meist die Verwendung von TestAndSet [▶ 387].
- Austausch von Daten über das SPS-Prozessabbild [▶ 390]
  - Diese Variante ist nur möglich und empfiehlt sich, wenn nur von einer Task schreibend auf dieselben Daten zugegriffen wird.
  - Aufgrund von notwendigen internen Kopieraktionen ist die mögliche Datenmenge eingeschränkt. Weitere Informationen hierzu entnehmen Sie bitte der Beschreibung „Austausch von Daten über das SPS-Prozessabbild“.
- Austausch von Daten über synchronisierte Puffer [▶ 391]
  - Diese Variante ist nur möglich, wenn nur von einer Task schreibend auf dieselben Daten zugegriffen wird.
  - Hierbei handelt es sich um eine anwendereigene Implementierung, für die es unterschiedliche Möglichkeiten gibt.
  - Der Zugriff auf die einzelnen Puffer muss, beispielsweise mit TestAndSet(), abgesichert werden.

- Aufgrund von durchgeführten Kopieraktionen ist die mögliche Datenmenge eingeschränkt. Weitere Informationen hierzu entnehmen Sie bitte der Beschreibung „Austausch von Daten übersynchronisierte Puffer“.

Synchronisation auch bei atomarem Zugriff

Die Notwendigkeit zur Synchronisation gilt normalerweise selbst dann, wenn ein einzelner Zugriff auf eine Variable (z. B. Integer schreiben) als atomar, d. h. ununterbrechbar, bezeichnet werden könnte.

Weil die Eigenschaft des atomaren Zugriffs u. a. von der eingesetzten Prozessorarchitektur abhängig ist, sollte der Einfachheit halber sowie zur Sicherheit jeder Zugriff als nicht-atomar betrachtet werden.

Zu beachten ist auch, dass sich selbst vermeintlich sichere Zugriffe bei näherer Betrachtung fast immer als unsicher herausstellen. Dies wird im Folgenden mithilfe von zwei Szenarien exemplarisch dargestellt:

- I.d.R. ist mehr als ein atomarer Zugriff für den gewünschten Funktionsausdruck erforderlich (z. B. Lesen, Ändern, Schreiben). Wenn ein solch mehrteiliger Funktionsausdruck in mehreren Tasks vorhanden ist, kann eine gleichzeitige Ausführung auf mehreren Tasks zu einem anderen Gesamtergebnis führen als eine sequenzielle Abarbeitung der Ausdrücke.
  - Beispiel: Eine globale Zählvariable (initialisiert mit 0) soll jeweils (aus zwei Tasks heraus) um 1 inkrementiert werden (`nGlobal := nGlobal + 1;`). Wird die Inkrementierung gleichzeitig ausgeführt, wird beide Male `0 + 1 = 1` gerechnet und der resultierende Wert der globalen Variablen ist 1, obwohl der Wert 2 beabsichtigt wäre.
- Mehrere Lesezugriffe zu unterschiedlichen Zeitpunkten innerhalb einer Taskabarbeitung können unterschiedliche Variablenwerte liefern.
  - Beispiel: Eine globale Variable wird aus einer Task heraus geschrieben. Ihr Wert war zuvor 50 und wird nun gleich 0 geschrieben (`nGlobal := 0;`). In einem anderen Taskkontext wird der Wert der globalen Variablen abgefragt (`IF nGlobal>10 AND nGlobal<20 THEN`). Die Abfrage beinhaltet zwei Lesezugriffe. Findet zwischen diesen Lesezugriffen der obige Schreibzugriff aus der anderen Task statt, so ist die Bedingung erfüllt, obwohl die globale Variable zu keinem Zeitpunkt einen Wert zwischen 10 und 20 hatte.

Weitere Hinweise

- Systemoperatoren, wie z. B. ADD, SIZEOF oder __NEW, können von mehreren Tasks gleichzeitig aufgerufen werden. Diese Aussage bezieht sich nur auf den verwendeten Operator. Wenn bei den Aufrufen auf Daten zugegriffen wird, die wiederum von mehreren Tasks aus genutzt werden, müssen diese Datenzugriffe entsprechend synchronisiert werden.
- Eine Funktionsbausteininstanz, die intern ADS nutzt, darf nicht in unterschiedlichen Tasks verwendet werden. Wenn die Instanz trotzdem aus einer anderen Task aufgerufen wird, wird der Fehler ADSERR_DEVICE_INVALIDCONTEXT=0x709=1801 ausgegeben.
- Aufgrund der Ausführung auf dem Stack ist die Verwendung der STRING-Funktionen (Tc2_Standard-Bibliothek) in unterschiedlichen Tasks in TwinCAT 3 unkritisch (in TwinCAT 2 sind die STRING-Funktionen nicht standardmäßig sicher bei Taskwechsel).
- Beachten Sie auch die Möglichkeiten der Compiler-Erweiterung (Dokumentation TE1200 TC3 PLC Static Analysis, die Regeln zum Thema „gleichzeitige/konkurrierende Zugriffe“ zur Verfügung stellt. Dies ist als Hilfsmittel zu sehen, mit dem potentieller Synchronisationsbedarf aufgedeckt werden kann.

14.1 Mutex-Verfahren (TestAndSet, FB_IecCriticalSection) zum Absichern von kritischen Bereichen

Bei der Verwendung von Mutex-Verfahren bzw. bei der Implementierung eines gegenseitigen Ausschlusses werden die Bereiche, in denen konkurrierende Zugriffe stattfinden, als kritische Bereiche (engl. Critical Sections) bezeichnet. Diese Bereiche können mithilfe der Funktion TestAndSet() [▶ 387] oder des Funktionsbausteins FB_IecCriticalSection [▶ 388] (beide aus der SPS-Bibliothek Tc2_System) synchronisiert werden, sodass die Bereiche unter gegenseitigen Ausschluss gestellt werden und zu einem Zeitpunkt jeweils nur eine Task auf die gemeinsam genutzten Daten zugreifen kann.

Das Betreten eines kritischen Bereichs kann von einer oder mehrerer Bedingungen abhängen. Zudem können verschiedene kritische Bereiche von jeweils unterschiedlichen Bedingungen abhängen.

Beispiele

- Wenn der Bereich 1a lesend und der Bereich 1b schreibend auf die Daten „Data1“ zugreift, müssen die Bereiche 1a und 1b miteinander synchronisiert werden. Wenn die Bereiche 2a, 2b und 2c jeweils lesend und schreibend auf die Daten „Data2“ zugreifen, müssen die Bereiche 2a, 2b und 2c miteinander synchronisiert werden.
  Die Bereiche 1x und 2x sind jeweils nur von einer Bedingung abhängig (Bedingung: „Data1“ bzw. „Data2“ nicht gesperrt). Zudem sind sie in diesem Fall nicht übergreifend voneinander abhängig, da in den Bereichen jeweils auf unterschiedliche Daten zugriffen wird. D. h. auch wenn Bereich 1b die Daten „Data1“ sperrt, kann Bereich 2a gleichzeitig die Daten „Data2“ sperren und auf diese zugreifen.
  Somit müssen die Datenressourcen zwischen den folgenden kritischen Bereichen synchronisiert werden:
  - die Verwendung der Ressource „Data1“ zwischen den Bereichen 1a und 1b
  - die Verwendung der Ressource „Data2“ zwischen den Bereichen 2a, 2b und 2c
- Wenn die Bereiche 1x und 2x zusätzlich jeweils auf die Daten „DataA“ zugreifen (dabei findet mindestens ein schreibender Zugriff statt), ändert sich die Situation.
  Dann sind alle Bereiche 1x und 2x von zwei Bedingungen abhängig: Zum Betreten von Bereich 1x müssen die Daten „Data1“ sowie „DataA“ und zum Betreten von Bereich 2x müssen die Daten „Data2“ sowie „DataA“ freigegeben sein.
  Zudem sind die Bereiche 1x und 2x aufgrund der gemeinsamen Nutzung der Daten „DataA“ nun übergreifend voneinander abhängig und dürfen nicht mehr gleichzeitig ausgeführt werden.
  Somit müssen die Datenressourcen zwischen den folgenden kritischen Bereichen synchronisiert werden:
  - die Verwendung der Ressource „Data1“ zwischen den Bereichen 1a und 1b
  - die Verwendung der Ressource „Data2“ zwischen den Bereichen 2a, 2b und 2c
  - die Verwendung der Ressource „DataA“ zwischen allen Bereichen (1a, 1b, 2a, 2b, 2c)

Bei der Funktion TestAndSet() repräsentiert jeweils ein Flag (Boolesche Variable) eine Bedingung, von der das Betreten des kritischen Bereichs abhängt (z. B. bLockData1). Bei FB_IecCriticalSection wird jeweils eine Instanz des Funktionsbausteins als Sperrbedingung verwendet.

Blockierung

- TestAndSet(): Mit der Funktion kann die Belegung eines kritischen Bereiches markiert und geprüft werden. Die Funktion blockiert jedoch nicht und es besteht die Möglichkeit, dass der Bereich in einem Zyklus nicht durchlaufen werden kann.
- FB_IecCriticalSection: Wenn eine weitere Task durch Aufruf der Methode Enter() einen bereits belegten kritischen Abschnitt betreten will, wird sie durch den TwinCAT Scheduler blockiert. **Die Task wird angehalten, bis der Bereich wieder freigegeben ist.**

⚠ VORSICHT

Zykluszeitüberschreitung durch angehaltene Task

Die Dauer der Taskblockierung kann je nach (Auslastung der) Zykluszeit zur Zykluszeitüberschreitung der Task führen.

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

Daraus ergibt sich der Vorteil der Funktion TestAndSet() gegenüber dem Funktionsbaustein FB_IecCriticalSection, dass eine Task in keinem Fall blockiert wird. Der Nachteil ist wiederum, dass für den Fall, dass die Funktion TestAndSet() keinen Zugriff gewährt, eine alternative Implementierung vorhanden sein muss. Dies könnte beispielsweise über eine Zustandsmaschine realisiert werden, um den Zugriff im nächsten Zyklus erneut zu versuchen.

Verklemmung (engl. deadlock)

Beachten Sie, dass es bei einer ungünstigen Verwendung des Funktionsbausteins FB_IecCriticalSection aufgrund der möglichen Blockierung von Tasks zu Verklemmungen kommen kann. An einer Verklemmung sind immer mindestens zwei Tasks beteiligt. Sie entsteht dann, wenn die Tasks während ihrer Blockierung gegenseitig auf die Freigabe einer weiteren Ressource warten, die die jeweils andere Task bereits gesperrt hat.

⚠ VORSICHT

Dauerhafter Taskstillstand durch Verklemmung

Wenn eine so beschriebene Verklemmung einmal eingetreten ist, lässt sich diese nicht mehr programmatisch beseitigen. Es kommt zum dauerhaften Stillstand der beteiligten Tasks.

Beispiel:

- Task 1 und 2 benötigen beide Zugriff auf die Daten „DataA“ und „DataB“.
- Task 1 sperrt zunächst die Ressource „DataA“ und Task 2 sperrt gleichzeitig zunächst die Ressource „DataB“.
- Anschließend möchte Task 1 die Daten „DataB“ sperren. Da diese Ressource bereits von Task 2 gesperrt ist, ist dies nicht möglich ist und Task 1 wird blockiert.
- Task 2 wiederum möchte zusätzlich zu „DataB“ noch die Daten „DataA“ sperren. Da diese bereits von Task 1 gesperrt sind, ist auch diese Sperrung nicht möglich und auch Task 2 wird blockiert.
- Somit sind beide Tasks blockiert. Sie warten jeweils auf die Freigabe von Daten, die die jeweils andere Task sperrt, aber aufgrund ihrer eigenen Blockierung nicht freigeben kann. Folglich warten die beiden Tasks unendlich lange aufeinander.

Auch bei einer falschen Synchronisation muss nicht zwangsläufig eine Verklemmungssituation eintreten. Es kommt nur dann zu einer Verklemmung, wenn sich die Abläufe zufällig in einer ungünstigen Reihenfolge ereignen.

Vermeidung von Verklemmungen

Generell können Sie Verklemmungen vermeiden, wenn jede Task immer nur eine Ressource zu einer Zeit sperren möchte.

Ebenso können Sie Verklemmungen dadurch vermeiden, dass Ressourcen immer nur in einer bestimmten Reihenfolge angefordert und gesperrt werden.

Beispiel:

- Das Problem bei dem zuvor genannten Beispiel ist, dass Task 1 und Task 2 die Ressourcen „DataA“ und „DataB“ in einer unterschiedlichen Reihenfolge anfordern und sperren.
- Wenn beide Tasks die Ressourcen hingegen immer in der gleichen Reihenfolge sperren, wird die Verklemmungsproblematik vermieden. Das heißt: Wenn jede Task, die in einem kritischen Bereich Zugriff auf die Daten „DataA“ und „DataB“ benötigt, immer zuerst „DataA“ anfordert und, wenn möglich, sperrt und erst bei erfolgreicher Sperrung von „DataA“ die Ressource „DataB“ anfordert und, wenn möglich, sperrt, kann es nicht zu der oben beschriebenen Verklemmung kommen, bei der sich die Tasks die benötigten Ressourcen in unterschiedlichen Reihenfolgen „wegschnappen“.

Beispielprogramm: Zugriffs-Synchronisation mittels TestAndSet()

Dieses Beispiel zeigt, wie aus verschiedenen Taskkontexten sicher auf gemeinsame Daten zugegriffen werden kann. Die Daten sind in einer Strukturinstanz zusammengefasst. Diese beinhaltet zusätzlich eine boolesche Variable bLocked als Test-Flag.

Bevor Sie auf Daten dieser globalen Strukturinstanz lesend oder schreibend zugreifen, müssen sie den Zugriff auf diese erfragen, indem Sie die Funktion TestAndSet() [▶ 387] mit dem entsprechenden Test-Flag aufrufen. Wenn der Zugriff nicht gestattet ist, können Sie in diesem Zyklus nicht auf die Daten zugreifen. In diesem Fall muss applikativ eine Alternativbehandlung vorgesehen werden. Bei Bedarf wird der Zugriff im nächsten Zyklus erneut angefragt. Wenn der Zugriff gestattet ist, entsprechend TestAndSet() erfolgreich aufgerufen wurde, können Sie die Daten lesen oder verändern. Sobald Sie die Bearbeitung abgeschlossen haben, geben Sie den Zugriff wieder frei, indem Sie das Test-Flag mit FALSE belegen.

Funktionstest

Starten Sie das Beispielprogramm. Innerhalb MAIN1 und MAIN2 befindet sich jeweils eine Zählervariable (nLocalBlockedCounter), die bei fehlgeschlagenem TestAndSet()-Aufruf hochzählt. Wenn diese 0 ist, konnte der Datenzugriff auf die globalen Variablen bei jedem Versuch erfolgreich ausgeführt werden.

Um ein Gefühl für die Notwendigkeit der Zugriffs-Synchronisation zu erhalten, können Sie die zwei Tasks von unterschiedlichen CPU-Cores ausführen lassen. Wenn Sie nun das Beispielprogramm starten, dann sehen Sie, wie die Zählervariablen unregelmäßig hochzählen und somit anzeigen, dass der Datenzugriff ab und zu gesperrt war.

Die Zugriffs-Synchronisation ist jedoch nicht nur bei Multi-Core-Verwendung notwendig, sondern auch wenn die zwei Tasks auf einer CPU laufen. Durch gegenseitige Taskunterbrechungen kann es hier ebenso zu kritischen Inkonsistenzen bei ungesichertem Datenzugriff kommen.

Download: https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644032779.zip

Beispielprogramm: Zugriffs-Synchronisation mittels FB_IecCriticalSection

Windows CE

Die Funktionalität vom FB_IecCriticalSection wird unter Windows-CE-Betriebssystemen ab TwinCAT v3.1.4022.29 unterstützt.

Das Beispiel zeigt die Verwendung von Critical Sections in der SPS anhand von Geldtransfers bei Geldkonten. Ein Konto wird durch einen Funktionsbaustein FB_Account repräsentiert. Vier Konten sind beteiligt. Alle vier Funktionsbausteininstanzen sind in einer globalen Variablenliste deklariert, um den Zugriff (hier: Geldtransfer) aus verschiedenen Taskkontexten zu ermöglichen.

Jedes Konto hat zu Beginn einen Kontostand von 1000. Der Kontostand von jedem Konto kann mit den Methoden Get() und Set() ausgelesen und neu gesetzt werden. Die folgenden Geldtransfers sind in vier unterschiedlichen Taskkontexten implementiert.

Task 1: A->B 500

Task 2: B->C 250, B->D 250

Task 3: C->A 250

Task 4: D->A 250

Nach Abschluss dieser Geldtransfers sollte jedes Konto wieder über einen Betrag von 1000 verfügen.

Es muss sichergestellt werden, dass der Zugriff auf ein Konto niemals aus zwei Taskkontexten zugleich geschieht.

Im FB_Account ist der Funktionsbaustein FB_IecCriticalSection [▶ 388] verwendet. Die Methode FB_Account.Lock() führt ein Enter() der Critical Section aus und die Methode FB_Account.Unlock() führt ein Leave() der Critical Section aus. Bevor auf den Kontostand eines Kontos zugegriffen werden darf, muss die Methode Lock() erfolgreich ausgeführt werden. Der Zugriff auf dieses Konto ist dann für andere gesperrt.

Um eine Verklemmung (engl. deadlock) zu vermeiden, wird eine Sperrreihenfolge festgelegt. Diese soll folgendermaßen definiert sein:

Sperrreihenfolge: A vor B vor C vor D

Damit ergibt sich als Entsperrreihenfolge: D vor C vor B vor A

Implementierung vom Geldtransfer innerhalb der Task 3:

```
(* Task 3: C->A 250*)
IF GVL.fbDepotA.Lock() THEN
    IF GVL.fbDepotC.Lock () THEN
        GVL.fbDepotC.Set (GVL.fbDepotC.Get() - 250);
        GVL.fbDepotA.Set (GVL.fbDepotA.Get() + 250);
        GVL.fbDepotC.Unlock();
    END_IF
    GVL.fbDepotA.Unlock();
END_IF
```

Funktionstest

Starten Sie das Beispielprogramm. Innerhalb Main1 können Sie die Summe aller Kontostände sehen, welche im SPS-Online-Ansicht immer 4000 betragen muss.

Um ein Gefühl für die Notwendigkeit der Verwendung von Critical Sections in diesem Beispiel zu erhalten, können Sie die vier Tasks von unterschiedlichen CPUs ausführen lassen und die globale Variable bIgnoreLock auf TRUE setzen. Wenn Sie nun das Beispielprogramm starten, dann sehen Sie, wie die Kontostände fehlerhafte Werte annehmen und die Summe aller Kontostände ebenfalls eine Fehlfunktion der Geldtransfers belegt.

Die Zugriffs-Synchronisation ist jedoch nicht nur bei Multi-Core-Verwendung notwendig, sondern auch wenn die vier Tasks auf einem CPU-Core laufen. Durch gegenseitige Taskunterbrechungen kann es hier ebenso zu kritischen Inkonsistenzen bei ungesichertem Datenzugriff kommen.

Download: https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643978251.zip

14.1.1 TestAndSet

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

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

Ein-/Ausgänge

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

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

Beispiel

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

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

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

    (* ... *)

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

    (* ... *)
END_IF
```

NEGATIV-Beispiel

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

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

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

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

    (* ... *)

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

Mit dem Funktionsbaustein FB_IecCriticalSection [▶ 388] wird die Verwendung von kritischen Bereichen als alternatives Mutex-Verfahren angeboten.

Voraussetzungen

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

14.1.2 FB_IecCriticalSection

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

Zykluszeitüberschreitung durch angehaltene Task

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

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

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

Windows CE

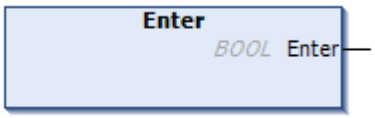
Die Funktionalität wird unter Windows-CE-Betriebssystemen ab TwinCAT v3.1.4022.29 unterstützt. (Bei einer älteren TwinCAT Version geben die Methoden FALSE aus.)

Alternative

Kritische Bereiche können auch mit der Funktion TestAndSet() [▶ 387] realisiert werden. Mit der Funktion kann die Belegung eines kritischen Bereiches markiert und geprüft werden. Die Funktion blockiert jedoch nicht und es besteht die Möglichkeit, dass der Bereich in einem Zyklus nicht durchlaufen werden kann.

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

Methode Enter()



Die Methode markiert den Anfang eines kritischen Bereiches.

Mögliche Rückgabewerte:

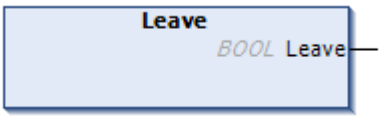
TRUE:

- Der kritische Bereich darf betreten werden.

FALSE:

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

Methode Leave()



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

Mögliche Rückgabewerte:

TRUE:

- Der Bereich wurde erfolgreich verlassen.

FALSE:

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

Beispiel für die Verwendung des Bausteins:

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

```
VAR_GLOBAL
    fbCrititcalSection : FB_IecCriticalSection;
END_VAR

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

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

Voraussetzungen

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

14.2 Austausch von Daten über das SPS-Prozessabbild

Von dem SPS-Prozessabbild werden typischerweise Daten mit dem IO-Prozessabbild ausgetauscht. Auf die gleiche Art und Weise ist es ebenso möglich, Daten innerhalb einer SPS zwischen zwei Taskkontexten auszutauschen.

Eine prinzipielle Bedingung für die Anwendbarkeit dieser Methode ist, dass nur von einer Task schreibend auf dieselben Daten zugegriffen werden soll. Die Daten werden dann so deklariert, dass sie Teil des Ausgangs-Prozessabbilds dieser Task sind.

Beispiel

- Aus dem Taskkontext 1 soll schreibend und aus dem Taskkontext 2 lesend auf die Daten „DataA“ zugegriffen werden.
- Zusätzlich soll aus dem Taskkontext 2 schreibend und aus dem Taskkontext 1 lesend auf die Daten „DataB“ zugegriffen werden.
- Umsetzung:
  - Sie definieren „DataA“ für den Taskkontext 1 im Ausgangs-Prozessabbild und zudem für den Taskkontext 2 im dortigen Eingangs-Prozessabbild.
  - Entsprechend definieren Sie „DataB“ für den Taskkontext 2 im Ausgangs-Prozessabbild und zudem für den Taskkontext 1 im dortigen Eingangs-Prozessabbild.
  - Die Daten „DataA“ und „DataB“ können Sie direkt an ihrem Verwendungsort deklarieren, anstatt sie global zu deklarieren.
  - In der Prozessabbilddarstellung unterhalb der SPS-Instanz im TwinCAT XAE verknüpfen Sie jeweils „DataA“ miteinander sowie auch „DataB“ miteinander.

Es dürfen nur jene Daten im Ausgangs-Prozessabbild angefügt werden, welche im anderen Taskkontext benötigt werden, um sicherzustellen, dass das Prozessabbild nicht unnötig groß ist. Das asynchrone Mapping führt in jedem Zyklus Kopieraktionen aus, um die Daten zwischen den SPS-Prozessabbildern und dem zusätzlichen temporären Puffer auszutauschen. Weil Kopieraktionen mit großen Datenblöcken viel Rechenzeit benötigen - wie dies auch beim Aufruf einer MEMCPY-Funktion der Fall ist - schränkt dies die mögliche Datenmenge meist auf deutlich weniger als 1 MB ein. Das Mapping wird von der jeweiligen Task selbst ausgeführt, sodass ein entsprechend aufwändiges Mapping die Tasklaufzeit für die eigentliche Programmabarbeitung reduziert.

Eine zweite Bedingung für die Anwendbarkeit dieser Methode ist, dass es sich bei den Daten nicht um Funktionsbausteininstanzen oder Zeiger jeglicher Art (POINTER TO, Interfacepointer/Interfacevariablen, Referenzen, ...) handeln darf. Es ist also hierbei nicht möglich, Methoden von ein und derselben Funktionsbausteininstanz aus zwei unterschiedlichen Taskkontexten aufzurufen.

Bei Bedarf können die erforderlichen Daten zu einer Struktur zusammengefasst werden.

Beispielprogramm: Synchronisation mittels Austausch von Daten über das SPS-Prozessabbild

Das Beispiel zeigt den Datenaustausch zwischen einer langsamen Task und einer schnellen Task innerhalb einer SPS-Instanz.

Exemplarisch für beliebige Datenblöcke (keine Funktionsbausteininstanzen oder Zeiger) wird eine Integer-Variable zur jeweils anderen Task übertragen. Zusätzlich ist dargestellt, wie Sie mittels einer booleschen Variablen eine Funktionalität im anderen Taskkontext triggern können.

Download: https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643979915.zip

14.3 Austausch von Daten über synchronisierte Puffer

Bei dieser Synchronisationsmöglichkeit handelt es sich um eine anwendereigene Implementierung. Als solche wird nicht auf spezielle bereitgestellte Funktionsbausteine einer Bibliothek zurückgegriffen. Die Komplexität des Programmcodes ist größer und hängt zudem von der gewählten Umsetzung ab.

Eine prinzipielle Bedingung für die Anwendbarkeit dieser Methode ist, dass nur von einer Task schreibend auf dieselben Daten zugegriffen werden soll.

Mittels zusätzlicher temporärer Datenpuffer wird der schreibende und der lesende Zugriff auf die Daten synchronisiert. Zur Synchronisation dieser Datenpuffer ist wiederum ein MUTEX-Verfahren [▶ 383] notwendig. Gegenüber der alleinigen direkten Verwendung von TestAndSet() hat diese Methode jedoch den Vorteil, dass der Zugriff auf einen dieser zusätzlichen Datenpuffer immer gewährt wird und keine Alternativbehandlung notwendig ist.

Eine zweite Bedingung für die Anwendbarkeit dieser Methode ist, dass es sich bei den Daten nicht um Funktionsbausteininstanzen oder Zeiger jeglicher Art (POINTER TO, Interfacepointer/Interfacevariablen, Referenzen, ...) handeln darf. Es ist also hierbei nicht möglich, Methoden von ein und derselben Funktionsbausteininstanz aus zwei unterschiedlichen Taskkontexten aufzurufen.

Es dürfen nur Daten zum Datenaustausch definiert werden, welche im anderen Taskkontext benötigt werden, um sicherzustellen, dass die Datenmenge nicht unnötig groß ist. Die Implementierung führt Kopieraktionen aus, um die Daten zwischen den Puffern auszutauschen. Weil Kopieraktionen mit großen Datenblöcken viel Rechenzeit benötigen (Aufrufe der MEMCPY-Funktion), schränkt dies die mögliche Datenmenge meist auf deutlich weniger als 1 MB ein.

Die für diese Synchronisationsmöglichkeit notwendigen Methoden werden vom Anwender implementiert. Diese spezifische Implementierung kann unterschiedlich ausgeführt werden. Teilweise werden auch Begriffe wie 3-Wege-Puffer oder 3-Puffer-Prinzip für die Art der Implementierung verwendet.

Beispielprogramm: Synchronisation mittels Austausch von Daten über synchronisierte Puffer

Das Beispiel zeigt den Datenaustausch zwischen einer langsamen Task und einer schnellen Task innerhalb einer SPS-Instanz.

Exemplarisch für beliebige Datenblöcke (keine Funktionsbausteininstanzen oder Zeiger) werden zwei Strukturen ST_DataA und ST_DataB zur jeweils anderen Task übertragen. Aus dem Taskkontext 1 soll schreibend und aus dem Taskkontext 2 lesend auf die Daten „DataA“ zugegriffen werden. Zusätzlich soll aus dem Taskkontext 2 schreibend und aus dem Taskkontext 1 lesend auf die Daten „DataB“ zugegriffen werden. „DataA“ und „DataB“ werden für den Taskkontext 1 und Taskkontext 2 jeweils als Strukturinstanz definiert.

Zur Synchronisation dieser Datenpuffer gibt es im Beispiel den Funktionsbaustein FB_DataSync mit den jeweiligen Erweiterungen FB_MyDataASync und FB_MyDataBSync. Instanzen dieser Funktionsbausteine werden in einer globalen Variablenliste deklariert und im Programmablauf verwendet. Hier gilt weiterhin, dass nur ein Taskkontext schreibend – Methode Write() – und ein anderer lesend – Methode Read() – auf diese Funktionsbausteininstanzen zugreifen darf. Zur Sicherheit schlagen die Methoden bei Nichtbeachtung fehl (Rückgabewert FALSE).

Wie eingangs erwähnt ist die Implementierung komplexer als bei anderen SPS-Beispielen. In diesem Beispiel steckt die eigentliche Umsetzung des Datenaustausches im Funktionsbaustein FB_DataSync, der den Zugriff auf mehrere Puffer verwaltet. Wenn Sie die Funktionalität mit einer eigenen neuen Datenstruktur testen möchten, können Sie FB_DataSync unverändert lassen. Analog zu FB_MyDataASync definieren Sie sich einen neuen Funktionsbaustein, der von FB_DataSync ableitet. Weiterhin analog instanziieren Sie Ihre eigene Datenstruktur dort zweimal und fügen die Methoden Read()/Write() an, welche jeweils die Zuweisung einer Referenz Ihrer neuen Datenstruktur verlangen und intern die Methoden ReadData()/WriteData() der Basisklasse aufrufen.

Zur Umsetzung von FB_DataSync in diesem Beispiel sei gesagt, dass zwei interne Puffer zur temporären Datenablage verwendet werden. Der schreibende Taskkontext kopiert seine Daten in diese Puffer hinein und der lesende Taskkontext kopiert seine Daten aus ihnen heraus. Der korrekte Zugriff auf jeweils einen der

zwei internen Puffer wird mit der Funktion TestAndSet() sichergestellt. Somit findet kein gleichzeitiger Zugriff auf dieselben Daten statt, obwohl außen im Applikationscode zeitgleich Daten geschrieben und gelesen werden können.

Download: https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643976587.zip

15 Visualisierung erstellen

Im Unterschied zu TwinCAT 2 sind die Visualisierungsclients nichts anderes als Interpreter von Zeichenkommandos. Unabhängig von seinem Typ (PLC HMI [▶ 635], PLC HMI Web [▶ 640]) bekommt jeder Client die gleichen Anweisungen, sodass für jeden Typ die gleiche Visualisierung resultiert. Als Konsequenz muss die zugehörige Visualisierungsapplikation immer auf die SPS heruntergeladen werden. Dies kann für kleine Steuerungen ein Problem darstellen.

Es werden drei verschiedene Varianten der Visualisierung im SPS-Projekt unterschieden:

- Integrierte Visualisierung [▶ 634] – Visualisierung läuft in der Entwicklungsumgebung von TwinCAT auf dem Programmiersystem
- PLC HMI [▶ 635] – Visualisierung läuft ohne Entwicklungsumgebung auf dem Steuerungsrechner oder einem dritten Rechner
- PLC HMI Web [▶ 640] – Visualisierung läuft in einem Browser

Trotz der drei verschiedenen Varianten ist nur ein Engineering notwendig, wodurch das Look-and-feel in allen Visualisierungen identisch ist. PLC HMI und PLC HMI Web können ganz einfach durch Hinzufügen des TargetVisualization [▶ 638]- und WebVisualization [▶ 641]-Objekts freigeschaltet werden. Die Verfügbarkeiten der einzelnen Visualisierungselemente [▶ 424] und Funktionalitäten in den verschiedenen Varianten sind im Abschnitt "Verfügbarkeiten [▶ 643]" zu finden.

Eine Visualisierung kann aus mehreren Visualisierungsseiten bestehen. Jede dieser Seiten wird in einem Visualisierungsobjekt [▶ 422] verwaltet. In den Einstellungen eines Visualisierungsobjekts können zum Beispiel die Größenanpassung und die geplante Verwendung [▶ 422] ("Visualisierung", "Numpad/Keypad", "Dialog") geändert werden. Beim Hinzufügen der ersten Visualisierungsseite werden automatisch die Visualisierungsbibliotheken [▶ 420] und ein Visualisierungsmanager [▶ 405], der allgemeine Einstellungen für alle Visualisierungsseiten des SPS-Projekts verwaltet, hinzugefügt.

Eine Visualisierungsseite wird in einem Visualisierungseditor [▶ 394] entwickelt. Dieser Editor wird ergänzt durch einen Werkzeugkasten [▶ 402], der die verfügbaren Visualisierungselemente bereitstellt, und einem Eigenschaftenfenster [▶ 403] für die Konfiguration der eingefügten Elemente. Die Visualisierungselemente werden durch die Visualisierungsbibliotheken gemäß dem aktuell eingestellten Profil [▶ 421] geliefert. Die Visualisierungselemente können auf einfache Weise angeordnet und gruppiert [▶ 394] werden.

Im Gegensatz zu TwinCAT 2 werden alle Visualisierungselemente wie auch die Visualisierungsobjekte selbst als IEC61131-3 Funktionsbausteine implementiert. Sie können in Bibliotheken abgespeichert und somit auch in anderen Projekten verfügbar gemacht werden. Zudem können die in den Visualisierungsobjekten verwalteten Visualisierungsseiten in anderen Visualisierungsseiten instanziiert, das heißt referenziert [▶ 644], werden. Ergänzt wird dieses Konzept durch Platzhalter [▶ 644] in Form von Ein- und Ausgängen, über die die referenzierten Seiten parametriert werden können. Die Platzhalter können in einem Schnittstelleneditor bearbeitet werden. Auf diese Weise besteht die Möglichkeit, Standardseiten oder -dialoge mit einmaligem Aufwand zu entwickeln und sie dann beliebig oft verschieden parametriert wiederzuverwenden.

Die Visualisierungselemente können über Projektvariablen animiert werden, die direkt oder über Ausdrücke, das heißt Kombinationen der Variablen mit Operatoren und Konstanten, in den entsprechenden Einstellungen eingetragen werden können. Das erlaubt zum Beispiel eine Skalierung von Variablenwerten, so dass sie für die Verwendung in der Visualisierung geeignet sind. Beispielhaft besteht so für Elemente vom Typ Rechteck die Möglichkeit, eine absolute Bewegung zu programmieren.

Zusätzlich kann eine Benutzerverwaltung [▶ 412] und eine Sprachumschaltung [▶ 648], die durch Verwendung von Textlisten möglich ist, implementiert werden. Als Zeichenkodierungsvarianten stehen in der Visualisierung ANSI und Unicode [▶ 407] zur Verfügung. Innerhalb einer Visualisierung sind zudem beliebige Ausdrücke, sogar Funktionsaufrufe, zulässig.

TwinCAT-2-Visualisierungen können importiert werden.

15.1 Visualisierungseditor

Eine Visualisierung kann mithilfe des Visualisierungseditors erstellt und konfiguriert werden. Er wird durch einen Doppelklick auf ein Visualisierungsobjekt [▶ 422] im SPS-Projekt geöffnet und verfügt über drei weitere Editoren:

- Schnittstellen-Editor [▶ 395] zum Definieren von Platzhaltern
- Tastaturkonfigurationseditor [▶ 400] für das Verknüpfen von Aktionen mit Tasten bzw. Tastenkombinationen
- Elementliste [▶ 402], die eine Auflistung aller Elemente der im Visualisierungseditor geöffneten Visualisierungsseite enthält

Sie können entweder über den Menüpunkt "Visualization" oder über die Pfeile am oberen Rand des Visualisierungseditors ein- und ausgeblendet werden.

Der Visualisierungseditor wird zusätzlich funktional erweitert durch den Werkzeugkasten [▶ 402], der die verfügbaren Visualisierungselemente [▶ 424] für das Einfügen im Visualisierungseditor bereitstellt, und den Eigenschaftenfenster [▶ 403], in dem die Eigenschaften des gerade im Visualisierungseditor markierten Elements angezeigt und bearbeitet werden können. Sie können über den Menüpunkt "Ansicht" geöffnet werden.

Visualisierungselement selektieren

Ein im Visualisierungseditor selektiertes Visualisierungselemente [▶ 424] zeigt kleine schwarze Quadrate in seiner Rahmenlinie, die mit einem weiteren Mausklick ausgewählt und verschoben werden können. Auf diese Weise können Position und Größe des Elements verändert werden.

Um ein Element auszuwählen, ziehen Sie den Mauszeiger über das Element. Wenn das Cursor-Symbol eine Hand darstellt. Drücken Sie die linke Maustaste. Eine Mehrfach-Selektion ist möglich. Dazu werden jeweils mit gedrückter Umschalt-Taste die einzelnen Elemente ausgewählt oder mit gedrückter linker Maustaste ein Rechteck um die Elemente gezogen. Die Elemente können auch in der Elementliste [▶ 402] selektiert werden. Dann können auch einzelne Unterelemente einer Gruppe ausgewählt werden.

Tastaturbedienung

Wenn ein Element ausgewählt ist, kann die Selektion mithilfe der Tabulator-Taste auf das nächste Element in Hinzufügereihenfolge verschoben werden und mit Umschalt-Taste + Tabulator-Taste auf das entsprechend vorherige.

Mit der Leertaste wird in einem gerade selektierten Element ein Texteingabefeld geöffnet. Nach der Eingabe einer Zeichenfolge wird diese mit der Eingabe-Taste für das Element übernommen.

Die Eigenschaften eines ausgewählten Elements werden unmittelbar im Eigenschaften-Fenster angezeigt und können dort bearbeitet werden. Sind mehrere Elemente ausgewählt, werden Änderungen an den Eigenschaften aller Elemente angewendet.

Position, Größe, Ausrichtung, Reihenfolge

Nach dem Einfügen eines Visualisierungselements [▶ 424] können folgende Eigenschaften unmittelbar im Editorfenster verändert werden, wobei die Größe und Position auch im Eigenschaftenfenster [▶ 403] bearbeitet werden können.

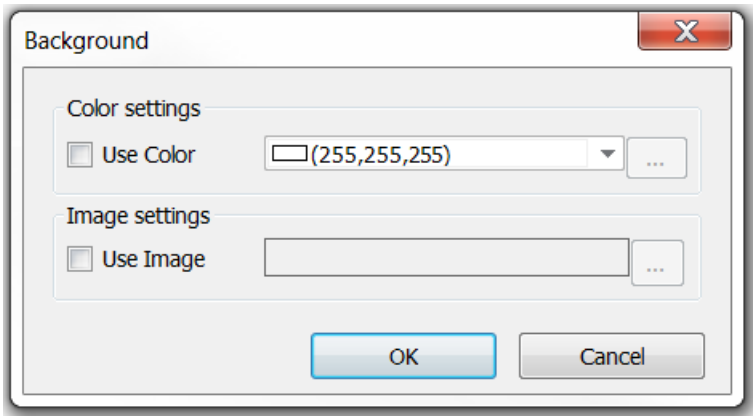
Position	Entweder Sie selektieren das Element mit einem Mausklick und ziehen es sofort, ohne die Maustaste loszulassen, an die gewünschte Position oder Sie klicken erneut auf ein bereits selektiertes Element und bewegen es dann ebenfalls mit gedrückter Maustaste oder mithilfe der Pfeiltasten. Eine weitere Möglichkeit besteht darin, die Positionswerte in den Elementeigenschaften [▶ 403] zu editieren.
Größe	Selektieren Sie das Element und klicken dann mit der Maus auf eines der kleinen Quadrate in seiner Rahmenlinie. Das Cursor-Symbol – gekreuzte Doppelpfeile oder ein vertikaler oder horizontaler Doppelpfeil – zeigt jeweils an, in welche Richtung die Größe des Quadrats verändert werden kann.
Ausrichtung	Um zwei oder mehrere Elemente aneinander auszurichten, selektieren Sie zunächst die entsprechenden Elemente. Öffnen Sie dann entweder über den Menüpunkt "Vizualization" oder über einen Rechtsklick im Visualisierungseditor das Untermenü "Alignment", um die Ausrichtung zu ändern.
Reihenfolge	Um ein oder mehrere Visualisierungselemente auf der Visualisierung in den Hintergrund, den Vordergrund oder dazwischen zu positionieren, selektieren Sie zunächst das oder die entsprechenden Elemente. Öffnen Sie dann entweder über den Menüpunkt "Visualization" oder über einen Rechtsklick im Visualisierungseditor das Untermenü "Order", um die Reihenfolge zu ändern.

Gruppierung von Visualisierungselementen

Um mehrere Visualisierungselemente zu einer Gruppe zusammenzufassen, selektieren Sie zunächst die entsprechenden Elemente. Wählen Sie dann entweder über den Menüpunkt "Visualization" oder über einen Rechtsklick im Visualisierungseditor "Gruppe" aus, um die Elemente zu gruppieren. Unterelemente einer Gruppe können nach der Gruppierung nur über die Elementliste [▶ 402] selektiert werden.

Hintergrundbild

Auf einer Visualisierungsseite besteht die Möglichkeit, eine Hintergrundfarbe und/ oder ein Hintergrundbild festzulegen. Über einen Rechtsklick im Visualisierungseditor kann ein Kontextmenü geöffnet werden, über das der Eintrag "Hintergrund" auswählbar ist. Dieser Eintrag öffnet den folgenden Dialog:



Mithilfe jeweils einer Checkbox kann eine Hintergrundfarbe und ein Hintergrundbild aktiviert werden. Ein Bild muss zuvor in einer Bildersammlung [▶ 153] zum SPS-Projekt hinzugefügt worden sein, damit es als Hintergrundbild eingestellt werden kann.

15.1.1 Schnittstellen-Editor

Der Schnittstellen-Editor wird verwendet, um Schnittstellen in Visualisierungen zu definieren. Die Schnittstellen sind dazu bestimmt, in einem Frame auf einer anderen Visualisierungsseite referenziert zu werden. Der Editor gehört zum Visualisierungseditor und erscheint im oberen Teil des Editors als separates Registerblatt neben dem Editor für die Tastaturkonfiguration [▶ 400] und der Elementliste [▶ 402].

Deklaration einer Schnittstelle

Da eine Visualisierung als Funktionsbaustein behandelt wird, erscheint der Schnittstelleneditor als normaler Deklarationseditor im oberen Teil des Visualisierungseditors. Hier können Eingabevariablen, also Frameparameter, deklariert werden.

Deklaration der Parameter

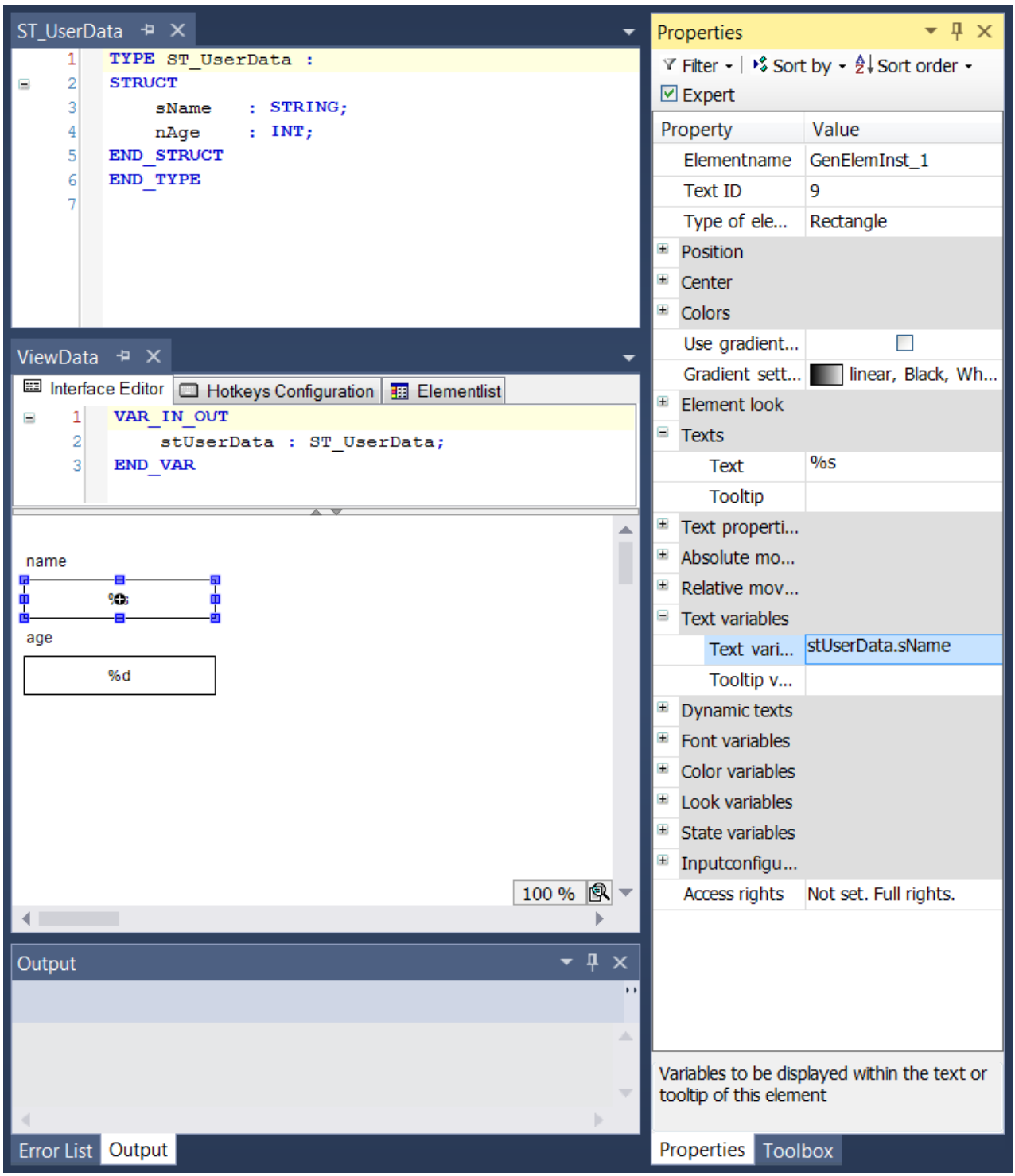
- VAR_INPUT
- VAR_INPUT mit Attribut 'parameterstringof': Eine Variable mit diesem Attribut erhält als Wert den Namen der Variable, die beim Aufruf in einem Frame Element auf eine zu definierende Eingangsvariablen gelegt ist.

```
VAR_INPUT
{attribute 'parameterstringof' := 'bInput'}
sVarName : STRING;
bInput : BOOL;
END_VAR
```

- VAR_IN_OUT: Wird eine Datenstruktur übertragen, muss diese als VAR_IN_OUT definiert werden. Ohne Pragma werden die Werte der Parameter kopiert, wenn die Visualisierung geöffnet wird. Eingaben werden auf die kopierte Datenstruktur geschrieben und mit Schließen der Visualisierung werden die Werte in die Parameter zurückgeschrieben.
- VAR_IN_OUT mit Attribut VAR_IN_OUT_AS_POINTER: Hier ist es erlaubt, Objekte als Referenzen an Visualisierungen zu übergeben. Nützlich kann das sein, wenn Informationen an eine Visualisierung übergeben werden sollen, ohne dass diese kopiert werden, oder wenn Informationen von außen aktualisiert werden sollen, während der Dialog geöffnet ist. Dieses Attribut ist ausschließlich bei Visualisierungen, die als Dialog eingestellt und verwendet werden, zulässig.

Beispiel

Die Visualisierung ViewData verwendet die Schnittstelle stUserData des Typs ST_UserData. Das Visualisierungselement GenElemInst_1 der Visualisierung ViewData verwendet User und zeigt den Variablenwert von stUserData.sName an, so wie es in der Ansicht Eigenschaften von GenElemInst_1 programmiert ist.



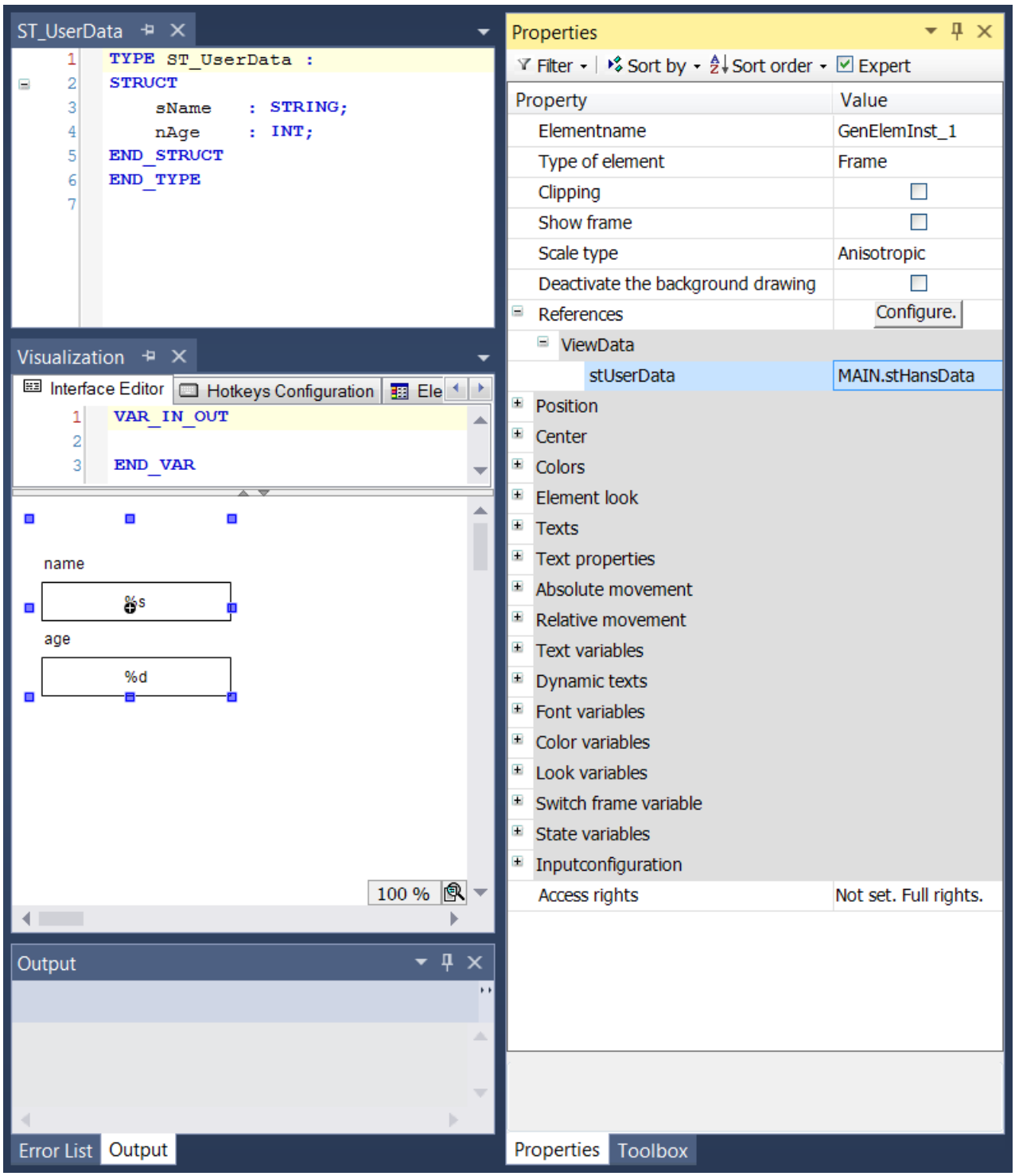
Programmierung des Frames

Um ein Frame-Element zu programmieren, gehen Sie wie folgt vor:

1. Fügen Sie ein Frame-Element in Ihrer Hauptvisualisierung hinzu.
2. Setzen Sie die Referenz auf eine Visualisierung mit Interface, die auch in einer Bibliothek gespeichert sein kann.
3. Weisen Sie jedem Frameparameter eine Variable zu.

Beispiel

Das Visualisierungselement Frame referenziert die Visualisierung ViewData. In der Ansicht Eigenschaften wird diese Zuweisung mit allen Frame-Parametern aufgeführt. Mit Klick in das Wertefeld einer solchen Variablen wird eine Variable des gleichen Typs zugewiesen. Hier wird also MAIN.stHansData zugewiesen.



Aktualisierung der Frameparameter

Wenn die Schnittstelle einer in einem Frame eingefügten Visualisierung geändert wird, wird beim Speichern oder Kompilieren des Projekts oder beim Öffnen eines betroffenen Objekts der im Beispiel dargestellte Dialog automatisch geöffnet. Sie können dann in diesem Dialog die Frame-Parameter anpassen. In dem Fall, dass die Schnittstelle einer aus einer Bibliothek stammenden Visualisierung geändert wurde, müssen die Parameter in Ihrem Projekt auch aktualisiert werden.

Parameter	In dieser Spalte können Sie die geänderten Parameter in Baumansicht sehen. Übergeordnet werden zuerst die Visualisierung und dann der entsprechende Frame mit der Referenz auf die geänderte Visualisierung aufgelistet. Darunter sind die aktuell gültigen Parameter und dann die vorigen aufgelistet.
Typ	In dieser Spalte ist der Typ der Parameter angezeigt.
Wert	Das Wertefeld enthält die Variable, die dem Parameter zugewiesen wurde. Beachten Sie die Färbung des Feldes: • Beige: Dieser Parameter wurde automatisch aus der bisherigen Konfiguration übernommen. • Grau: Diesem neuen Parameter ist noch kein Wert zugewiesen. • Grün: Diesem neuen Parameter ist bereits ein Wert zugewiesen. • Weiß: Hier muss nichts konfiguriert werden.

Werte bearbeiten

Um Werte, die sich unter Aktuelle befinden, zu bearbeiten, selektieren Sie das Wertefeld per Mausklick. Nach einem weiteren Mausklick in das Feld oder nach Drücken der Leertaste, öffnet sich ein Zeileneditor. Sie können nach dem Selektieren auch einfach anfangen zu tippen. Weisen Sie eine andere Variable zu, indem Sie deren Namen eintippen oder selektieren Sie eine mithilfe des Eingabeassistenten.

Ein vorhandener Werteintrag kann in ein anderes Feld kopiert werden. Selektieren Sie dafür den Eintrag, der kopiert werden soll, verwenden Sie "Kopieren" und selektieren Sie das Feld, in das der Eintrag eingefügt werden soll und verwenden Sie "Einfügen".

Bestätigen Sie die Einstellung im Dialog mit "OK". Der Dialog schließt und die neue Parameterzuweisungen können in der Ansicht Eigenschaften, Kategorie Referenzen beim zugehörigen Frame-Element angezeigt werden.

Gebrauch von Instanznamen in der Visualisierung

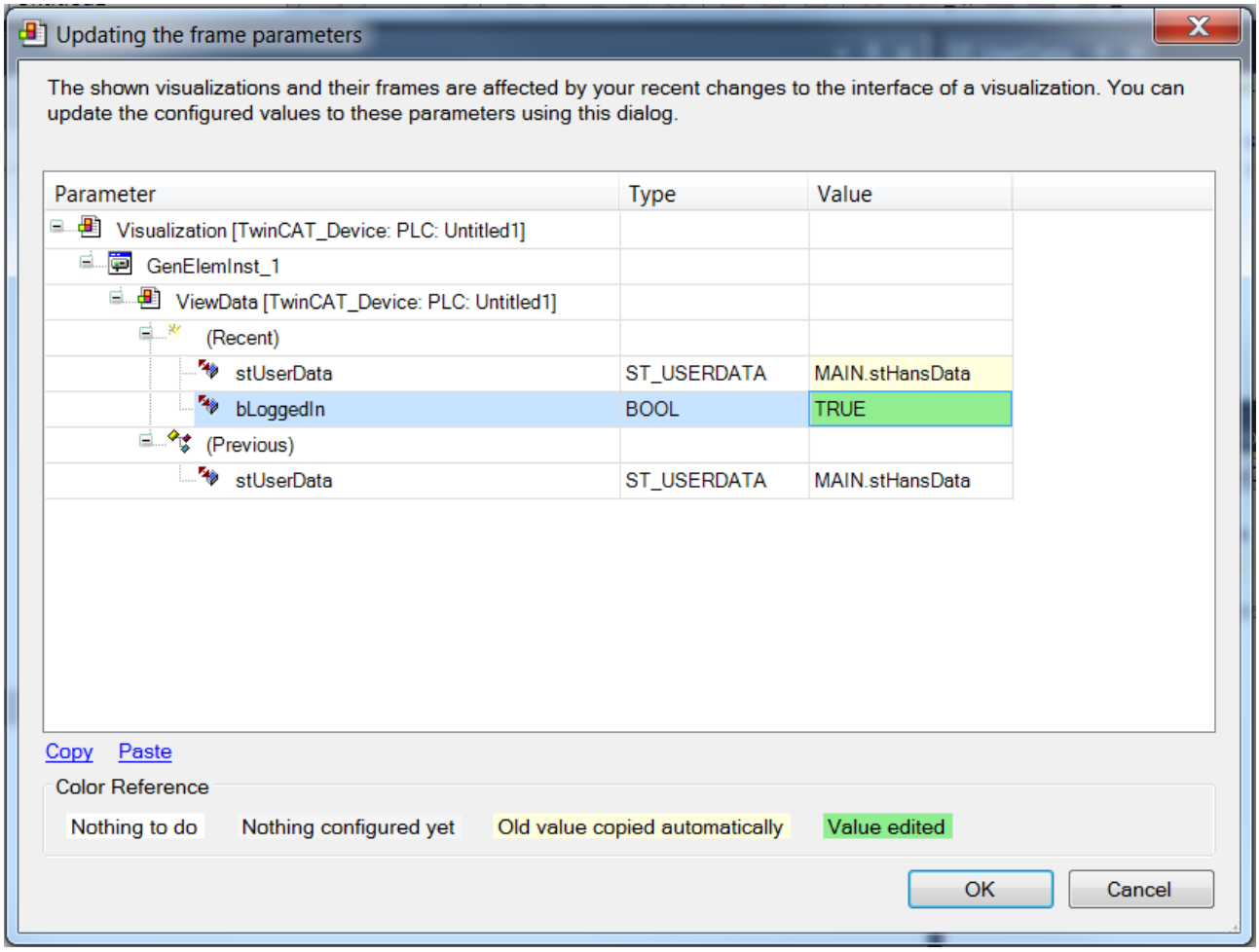
Falls Sie innerhalb des referenzierten Visualisierungsbausteins vom Instanznamen einer übergebenen Variablen Gebrauch machen möchten, müssen Sie mit dem Pragma-Attribut 'parameterstringof [▶ 856]' arbeiten, welcher einen entsprechenden String zur Verfügung stellt.

Beispiel

Die Schnittstelle der Visualisierung ViewData ist um die Variable bLoggedIn erweitert worden:

```
VAR_IN_OUT
stUserData : ST_UserData;
bLoggedIn : BOOL;
END_VAR
```

Beim Kompilieren oder Speichern des Projektes, erscheint der folgende Dialog für die Konfiguration des neuen Parameters:



Er enthält einen Vergleich zwischen den aktuellen und den vorigen Parametern. bLoggedIn wurde der neue Wert TRUE zugewiesen. Nach dem Schließen des Dialogs mit OK, können die neuen Parameter auch in den Eigenschaften des Frame-Elements nachvollzogen werden.

Pragmas

Es besteht die Möglichkeit, folgende zwei Parameter im Schnittstellen-Editor zu verwenden.

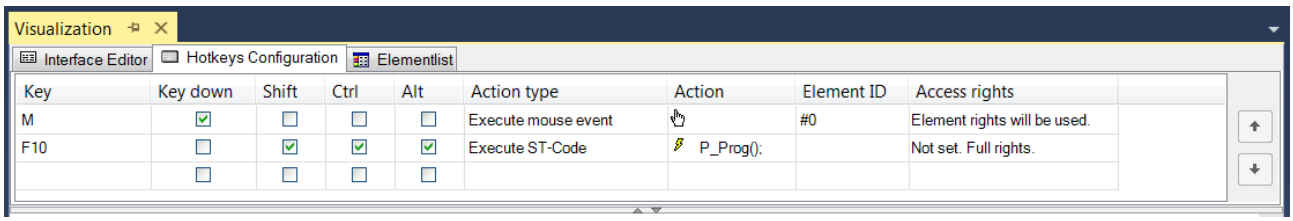
- Attribut "VAR_IN_OUT_AS_POINTER"
- Attribut 'parameterstringof'

15.1.2 Tastaturkonfiguration

Zusätzlich zu den Standard-Tastaturkürzeln [▶ 653] können in diesem Teil des Visualisierungseditors spezielle Tastenkürzel für die Bedienung einer Visualisierungsseite im Onlinebetrieb definiert werden. Das heißt, eine Aktion kann einer bestimmten Taste /Tastenkombination zugeordnet werden. Diese Tastenkonfiguration gilt genau für die vorliegende Visualisierungsseite. Tastenkonfigurationen, die auf allen Visualisierungsseiten des SPS-Projektes verwendbar sein sollen, sollten im Standardtastaturkürzel [▶ 411] definiert werden.

In diesem Zusammenhang ist auch die "Tastaturkürzel" Eigenschaft eines Visualisierungselements zu beachten, die in den Eingabekonfigurationen [▶ 460] eingestellt werden kann. Eine solche elementspezifische Tastenkonfiguration wird ebenfalls im Tastaturkonfigurations-Editor verwaltet. Sie kann an beiden Stellen bearbeitet werden, wobei sie jeweils auch im anderen Editor aktualisiert wird.

Der Tastaturkonfigurations-Editor ist als separates Registerblatt neben dem Schnittstellen-Editor [▶ 395] und der Elementliste [▶ 402] im oberen Teil des Visualisierungseditors zu finden.

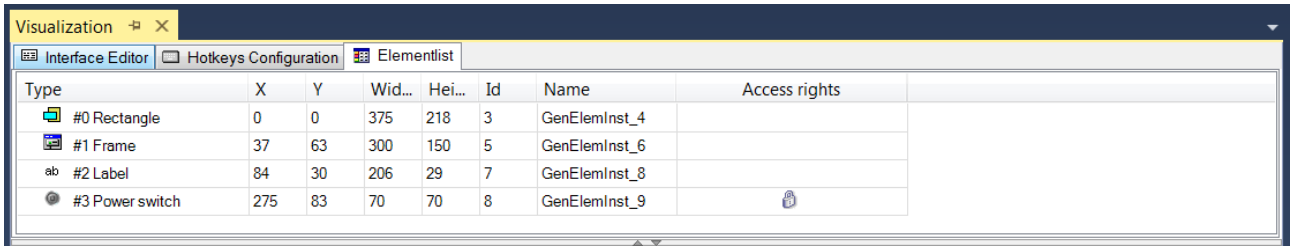


In jeder Zeile des Tabelleneditors kann eine Taste oder eine Tastenkombination mit einer Aktion verknüpft werden. Siehe die folgenden Spalten:

Taste	Name der Taste. Der Tastenname kann manuell oder über eine Auswahlliste eingegeben werden, die sich bei einem Doppelklick auf die Zelle öffnet. Die Liste enthält alle Tasten, die über die Gerätebeschreibung definiert sind.
Taste drücken	Wenn diese Option aktiviert ist, wird die Aktion ausgeführt, sobald die Taste gedrückt wird. Ansonsten wird sie ausgeführt, wenn die Taste wieder losgelassen wird. Wenn die Aktion sowohl beim Drücken (KeyDown) als auch beim Loslassen (KeyUp) der Taste ausgeführt werden soll, müssen zwei entsprechende Definitionen für die Taste in der Tabelle vorliegen.
Umschalten, Strg, Alt	Wenn diese Option aktiviert ist, muss die Umschalt- bzw. Strg- bzw. Alt-Taste zusammen mit der Taste gedrückt werden, damit die Aktion ausgeführt wird.
Aktionstyp	Der Aktionstyp kann über eine Auswahlliste definiert werden, die bei einem Doppelklick auf die Zelle geöffnet wird. Die Aktionstypen entsprechen denen, die in der Eingabekonfiguration [▶ 425] eines Visualisierungselements verfügbar sind.
Aktion	Genaue Konfiguration der auszuführenden Aktion. Sie hängt vom Aktionstypen ab und entspricht der Mausaktion, wie sie in der Eingabekonfiguration [▶ 425] eines Visualisierungselements verwendet werden kann.
Element ID	ID des Visualisierungselements, dem die Taste über die "Tastaturkürzel [▶ 460]" Eigenschaft zugeordnet ist. Eindeutige Kennung innerhalb der aktuellen Visualisierung. Wenn eine Taste mit mehreren Aktionen verknüpft ist, werden diese in der gleichen Reihenfolge ausgeführt, wie sie von oben nach unten hier in der Konfigurationstabelle stehen. Diese Anordnung kann geändert werden, indem eine Tastendefinition ausgewählt wird (Mausklick in die Tabellenzeile) und über die Pfeiltasten rechts von der Tabelle nach oben oder unten verschoben wird. Zu beachten ist die folgende Aufrufreihenfolge beim Abarbeiten von Tastenaktionen: 1. Eventhandler der Applikation, falls aktiviert (optional) z.B. Ereignisse und Eingabeaktionen bemerken 2. Tastaturkonfiguration [▶ 411] im Visualisierungsmanager, die für alle Visualisierungen der Applikation gilt 3. Definition der Standard-Tastaturbedienung [▶ 653] im Onlinebetrieb 4. Spezielle Tastaturkonfigurationen der einzelnen Visualisierungen, die hier beschrieben werden, wobei die Hauptvisualisierungen vor den in Frames referenzierten Visualisierungen beachtet werden.
Zugriffsrechte	Bei dieser Einstellung kann eine Auswahl der Benutzergruppen definiert werden, die das entsprechende Tastaturkürzel nutzen darf. Mit einem Mausklick kann ein Dialog geöffnet werden, über den die vorhandenen Gruppen ausgewählt werden können. Es existieren die folgenden beiden Statusmeldungen: • Nicht gesetzt. Volle Berechtigungen. Die Standardmeldung ist gesetzt, wenn das Tastaturkürzel für alle Gruppen bedienbar ist. • Rechte sind gesetzt: Begrenzte Berechtigungen. Die Meldung ist gesetzt, wenn das Tastaturkürzel von mindestens einer Gruppe nicht genutzt werden darf. • Elementrechte werden verwendet Falls das Tastaturkürzel einem Visualisierungselement über die "Tastaturkürzel [▶ 460]" Eigenschaft zugeordnet ist, werden automatisch die entsprechenden Rechte des Elements übernommen.

15.1.3 Elementliste

Dieser Teil des Visualisierungseditors zeigt eine Liste aller Elemente der aktuell im Editor geöffneten Visualisierungsseite.



Die Elemente werden von oben nach unten gemäß ihrer Position auf der Z-Achse der Visualisierung aufgelistet, das am weitesten im Hintergrund liegende in der ersten Zeile. Die folgenden Werte werden angezeigt, können jedoch hier nicht editiert werden:

Typ	Elementtyp und Symbol, wie sie im Werkzeugkasten [▶ 402] verwendet werden, sowie die Elementnummer, die sich zunächst aus der Reihenfolge des Einfügens ergibt.
X, Y	Position der oberen linken Ecke des Elements (0, 0 = obere linke Ecke des Visualisierungsbereichs)
Breite, Höhe	Maße des Elements
ID	Intern zugewiesene Elementkennung
Name	Elementname, wie in den Elementeigenschaften [▶ 403] definiert
Zugriffsrechte	Ist das Verhalten eines Elements für einige Benutzergruppen eingeschränkt, dann wird dies mit dem Vorhängeschlosssymbol gekennzeichnet.

Element(e) können durch Selektieren der entsprechenden Tabellenzeile(n) ausgewählt werden, wobei die Auswahl immer mit der im Hauptfenster des Visualisierungseditors synchronisiert wird. Zu beachten ist jedoch, dass Unterelemente einer Gruppe nur hier in der Elementliste ausgewählt werden können.

Die Position eines ausgewählten Elements auf der Z-Achse, d.h. auf der Hintergrund-Vordergrund Achse, kann nur im Visualisierungseditor [▶ 394] selbst geändert werden.

Die Befehle, um Bearbeitungsaktionen in der Elementliste rückgängig zu machen ([Strg] + [z]) oder Elemente zu entfernen (Entf), können auch in der Elementliste genutzt werden.

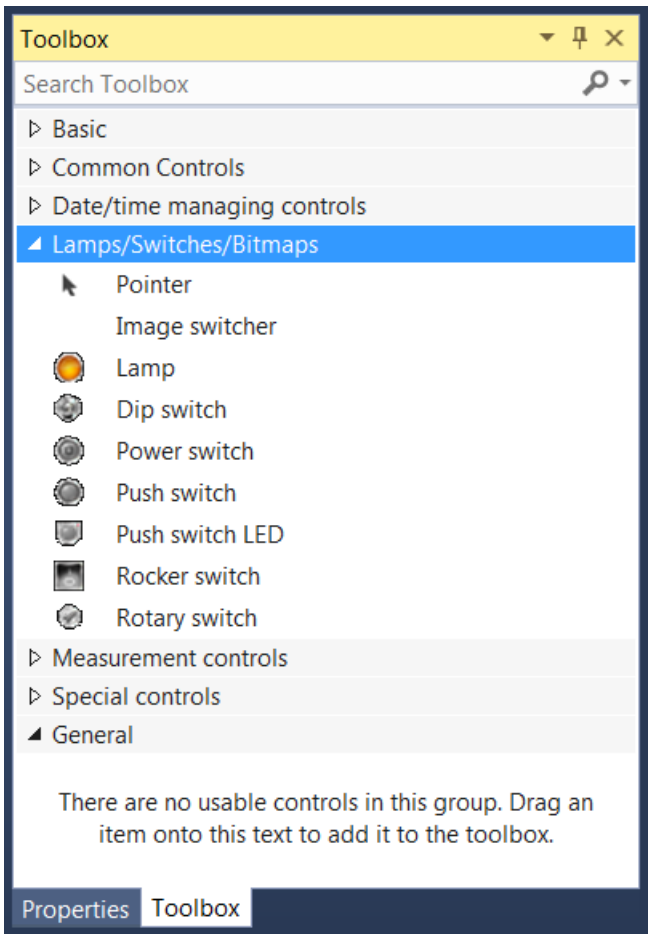
15.1.4 Werkzeugkasten

Der Werkzeugkasten stellt alle Visualisierungselemente bereit, die im Visualisierungseditor [▶ 394] eingefügt werden können. Die Elemente werden über Bibliotheken [▶ 420] im Projekt bereitgestellt. Die Auswahl im Werkzeugfenster hängt vom gerade aktiven Visualisierungsprofil [▶ 421] ab.

Falls das Werkzeugkastenfenster noch nicht sichtbar ist, kann es im Menü "Ansicht" über "Werkzeugkasten" geöffnet werden. Der Werkzeugkasten enthält die folgenden Kategorien:

- Allgemeine Steuerelemente [▶ 441]
- Basis [▶ 499]
- Datum-/ Zeit-Steuerelemente
- Lampen/ Schalter/ Bilder [▶ 556]
- Messgeräte [▶ 567]
- Spezielle Steuerelemente [▶ 612]

Unter diesen Kategorien werden die zugehörigen Visualisierungselemente [▶ 424] jeweils mit Symbol und Namen aufgelistet.

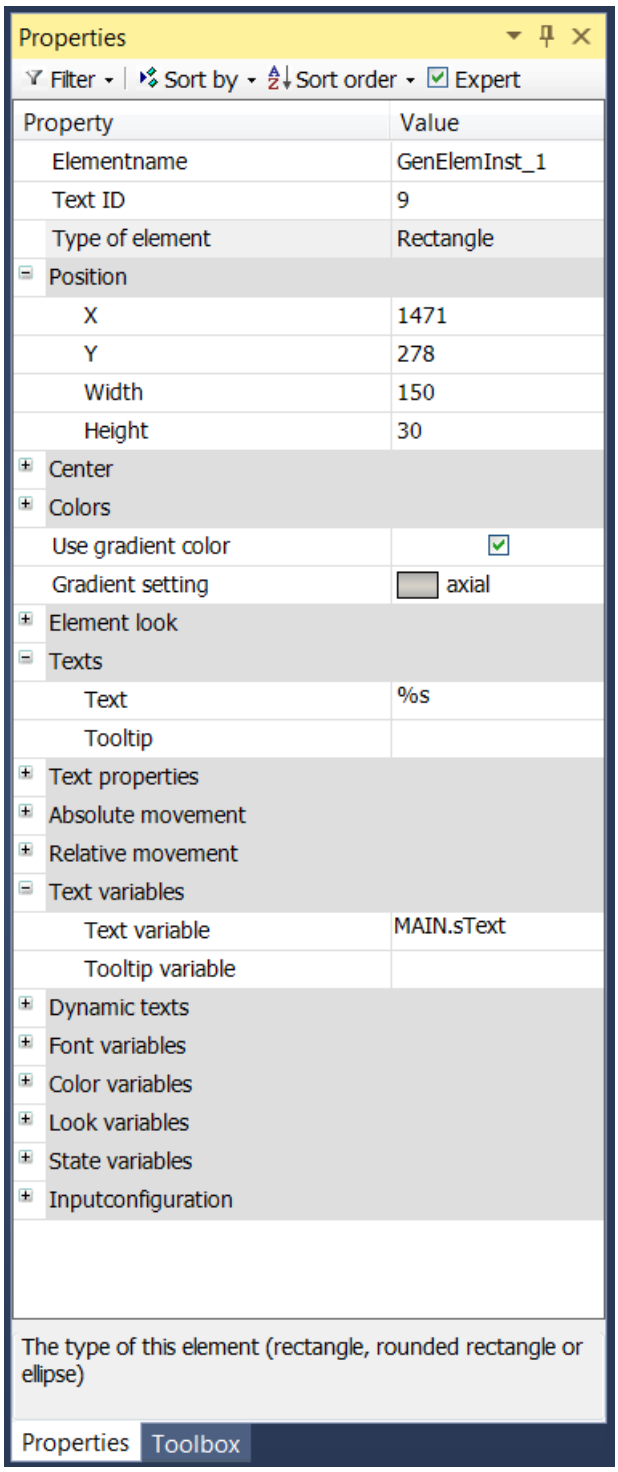


Mittels Drag-and-drop, also Anwählen und bei gedrückter Maustaste Ziehen, können die Visualisierungselemente im Visualisierungseditor eingefügt werden. Ein Pluszeichen während des Ziehens des Elements am Cursor-Symbol zeigt an, dass ein Element gerade ins Editorfenster eingefügt werden kann. Nach Loslassen der Maustaste wird das Element im Editor angezeigt.

Ein Element kann auch im Menü "Visualization" unter "Visualisierungselement einfügen" eingefügt werden. Hier steht die gleiche Auswahl an Elementen bereit wie im Werkzeugkasten. Wenn nötig, kann das Visualisierungsmenü über den Anpassen-Dialog (Befehlskategorie "Visualisierungskommandos") entsprechend angepasst werden.

15.1.5 Eigenschaftenfenster

Jedes Visualisierungselement [▶ 424] der aktuell im Visualisierungseditor [▶ 394] geöffneten Visualisierungsseite kann selektiert [▶ 394] werden. Dessen Position, Größe, Anordnung und Ausrichtung [▶ 394] kann dann direkt im Editor (Mausaktionen bzw. Visualisierungsbefehle) verändert werden. Die Konfiguration weiterer Parameter erfolgt im Eigenschaftenfenster.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

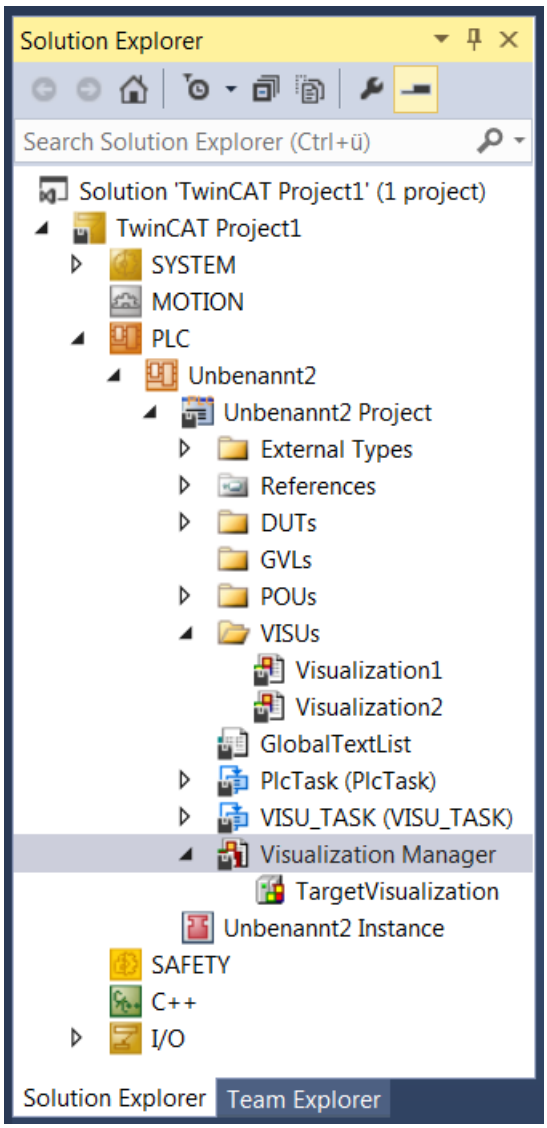
Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

15.2 Visualisierungsmanager

Der Visualisierungsmanager verwaltet allgemeine Einstellungen für alle Visualisierungsobjekte [▶ 422] eines SPS-Projekts. Das Visualisierungsmanager-Objekt () wird automatisch hinzugefügt, sobald das erste Visualisierungsobjekt im SPS-Projekt erstellt worden ist. Einen Überblick zur Visualisierung in SPS-Projekten finden Sie im Abschnitt "Visualisierung erstellen [▶ 393]".

Wenn es vom Gerät unterstützt wird, können unterhalb des Managers die Client-Objekte für die PLC HMI [▶ 635] und/oder die PLC HMI Web [▶ 640] eingefügt werden. Sie verwalten spezielle Einstellungen für die jeweilige Clientart. Die integrierte Visualisierung [▶ 634] wird automatisch verwendet, wenn kein Client-Objekt unterhalb des Visualisierungsmanagers eingefügt worden ist.

In dem in der folgenden Abbildung dargestellten Beispiel ist der Visualisierungsmanager verantwortlich für die Visualisierungsobjekte "Visualization1" und "Visualization2". Zudem wurde das TargetVisualization-Objekt hinzugefügt und damit die PLC HMI freigeschaltet.

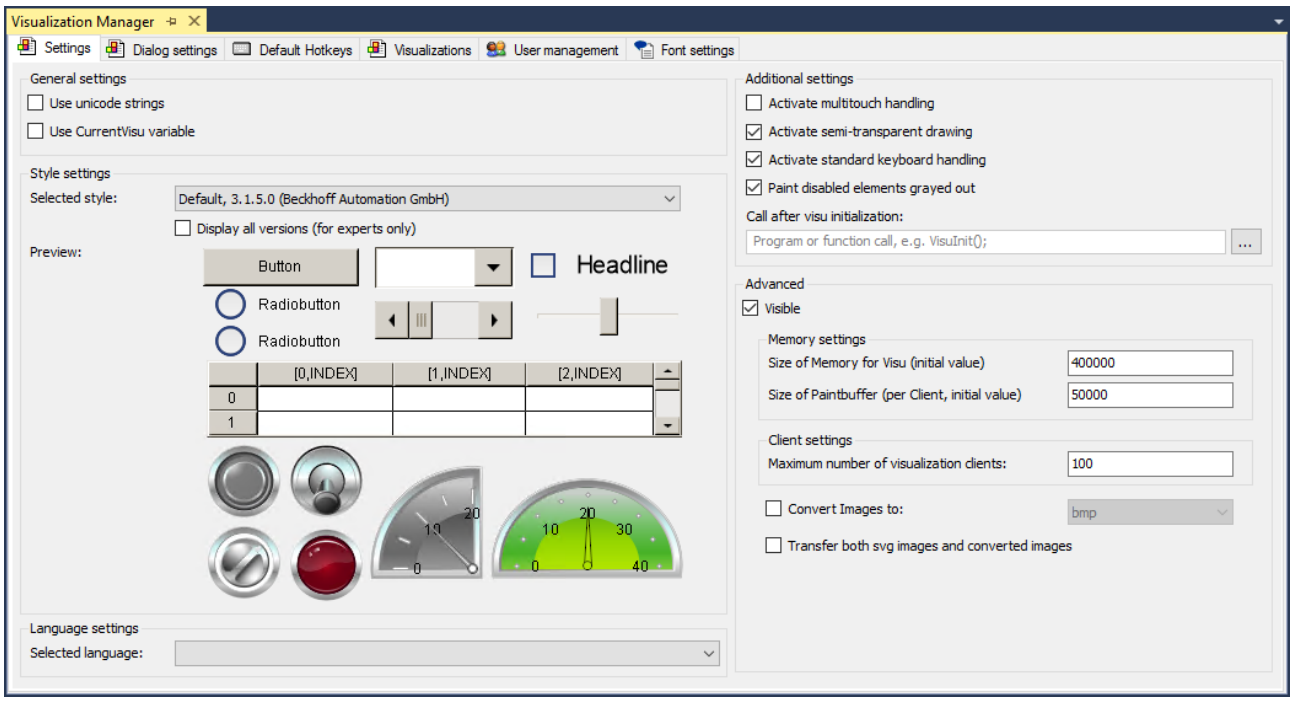


Um den Visualisierungsmanager-Editor zu öffnen, führen Sie einen Doppelklick auf den Eintrag aus. Der Editor wird in einem Fenster geöffnet, welches die folgenden Registerkarten aufführt:

- Einstellungen [▶ 406]
- Dialogeinstellungen [▶ 409]
- Standardtastaturkürzel [▶ 411]
- Visualisierungen [▶ 412]
- Benutzerverwaltung [▶ 412]
- Schriftart [▶ 419]

15.2.1 Einstellungen

Die Registerkarte enthält Einstellungen, die für alle Visualisierungen des SPS-Projekts gültig sind.



Allgemeine Einstellungen

Unicode Zeichenfolge verwenden	Wenn diese Option aktiviert ist, werden alle Zeichenfolgen [▶ 646], die in der Visualisierung verwendet werden, in Unicode-Format verarbeitet. Dafür muss zusätzlich in den Einstellungen des SPS-Projekts [▶ 421] "VISU_USEWSTRING" unter Compile > Settings > Compiler defines eingetragen werden.
CurrentVisu Variable verwenden	Das SPS-Projekt kennt und verwendet die globale Variable VisuElems.CurrentVisu des Typs STRING und enthält zur Laufzeit den Namen der gerade aktiven Visualisierung. Auf die Variable kann lesend zugegriffen werden, um den Namen der gerade aktiven Visualisierung zu erhalten, und schreibend, um einen Visualisierungswechsel hervorzurufen. Die Umschaltung erfolgt auf allen Anzeigegeräten parallel, sodass in allen verbundenen Clients dieselbe Visualisierungsseite angezeigt wird. Voraussetzung: Die Applikation enthält eine Visualisierung, die weitere Visualisierungen aufruft. Beispiele: • Zuweisung einer Variablen: `VisuElems.CurrentVisu := sVisuName;` • Zuweisung eines Textes: `VisuElems.CurrentVisu := 'Visualization1';`

Stilkonfiguration

Ausgewählter Stil	Jede Visualisierung stellt die Elemente in diesem Stil dar.
Alle Versionen anzeigen (nur für Experten)	Wenn diese Einstellung aktiviert worden ist, können in dem Auswahlmenü darüber alle auf dem System installierten Stilversionen ausgewählt werden.
Vorschau	Die Elemente, die in der Vorschau dargestellt sind, repräsentieren den ausgewählten Stil.

Spracheinstellung

Ausgewählte Sprache	Die ausgewählte Sprache wird beim Start einer Visualisierung verwendet.

Die Einstellung einer Standardsprache beim Start der Visualisierung ist nur in Verbindung mit der PLC HMI [▶ 635] und/ oder der PLC HMI Web [▶ 640] möglich. Bei der integrierten Visualisierung [▶ 634] wird beim Start automatisch die Textversion ‚Standard' verwendet. Eine Sprachumschaltung [▶ 648] zur Laufzeit über Schaltflächen auf der Visualisierung ist auch in der integrierten Visualisierung möglich.

Siehe auch:

- Text und Sprache [▶ 646]

Die Einstellungen für die Benutzerverwaltungsdialoge werden ab dem Build 4024.0 in dem separaten Tab Dialogeinstellungen [▶ 409] gelistet und sind dementsprechend dort dokumentiert.

Zusätzliche Einstellungen

Multitouch aktivieren	Zur Laufzeit erwartet die Visualisierung Benutzereingaben über Gesten und Touch-Ereignisse. Betroffene Elemente: • Elemente mit Eingabekonfiguration • Elemente des Typs Frame • Elemente des Typs Registersteuerelement
Semitransparentes Zeichnen aktivieren	Die Visualisierung zeichnet die Elemente in semitransparenter Farbe. Dafür können Sie bei der Definition einer Farbe zusätzlich einen Abstufungswert für die Transparenz angeben. In der Eigenschaft Transparenz ist die Durchsichtigkeit definiert. Voreinstellung: Aktiviert. Voraussetzung: Sie erstellen eine Visualisierung neu und die Darstellungsvarianten können semitransparent zeichnen.
Standardtastaturbedienung aktivieren	• Tabulator • Umschalt + Tabulator • Eingabe • Pfeil nach oben • Pfeil nach unten • Pfeil nach rechts • Pfeil nach links
Deaktivierte Elemente grau zeichnen	Die Visualisierung zeichnet deaktivierte Elemente ausgegraut.

Erweiterte Einstellungen

Sichtbar	Einstellungen für Speichereinstellungen, Dateiübertragungsmodus und Client-Einstellungen sind sichtbar. (Sie sind für Standardanwendungen nicht notwendig)

Wenn die integrierte Visualisierung verwendet wird, werden die nicht verfügbaren Einstellungen ausgegraut oder gar nicht dargestellt.

Speichereinstellungen

Speichergröße für Visu	Speichergröße in Bytes, den die Visualisierung zur Laufzeit alloziert. Voreinstellung: 400000
Größe des Zeichenpuffers (pro Client)	Speichergröße in Bytes, den die Visualisierung pro Darstellungsvariante alloziert und bei Zeichenaktionen verwendet. Voreinstellung: 50000

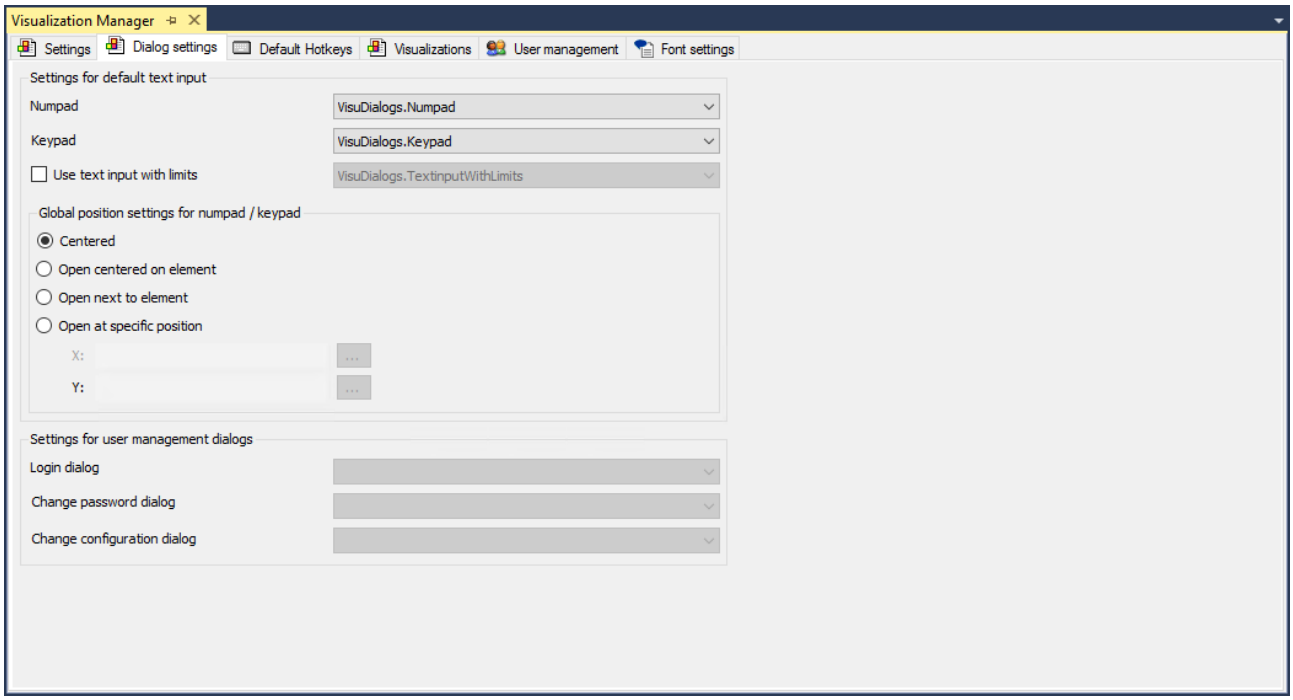
Client-Einstellungen

Maximale Anzahl von Visualisierungs-Clients	Begrenzt die Anzahl an Darstellungsvarianten, die gleichzeitig in Ausführung sind. Wenn Sie Elemente so konfigurieren, dass sie abhängig von der Darstellungsvariante variieren, dann müssen Sie die Anzahl an Darstellungsvarianten begrenzen. Eine Visualisierung erhält zur Laufzeit eine ID, welche die Darstellungsvariante identifiziert, und verarbeitet Daten dann dementsprechend. TwinCAT kann die ID mit der Systemvariablen CURRENTCLIENTID abfragen und erhält somit die Information, welche der laufenden Varianten betroffen ist. Beispiel: `arr[CURRENTCLIENTID].dwColor` Voraussetzung: Bibliothek VisuGlobalClientManager ist ins Projekt eingebunden.

Bilder konvertieren zu	Wenn diese Einstellung aktiviert ist, kann in einem Dropdown-Menü ausgewählt werden, ob die Bilder der Standardelemente [▶ 424] in das Format bmp oder png übersetzt werden sollen. Dies ist notwendig, wenn auf dem Zielsystem ein Windows Embedded Compact Betriebssystem installiert ist, da dort ausschließlich bmp, png und jpg Bilder im PLC HMI [▶ 635] Client dargestellt werden können.
SVG und konvertierte Bilder übertragen	Wenn diese Einstellung aktiviert ist, werden Bilddateien, die zuvor in bmp oder png konvertiert worden sind, auch im Ursprungsformat svg auf das Zielsystem geladen. Der PLC HMI [▶ 635] Client verwendet lokal auf dem Beckhoff CE Gerät die Bilddateien im bmp oder png Format und ein PLC HMI Web [▶ 640] Client die Bilddateien im svg Format. Diese Einstellung ist dann verfügbar, wenn sowohl die PLC HMI [▶ 635] als auch die PLC HMI Web [▶ 640] aktiviert sind.

15.2.2 Dialogeinstellungen

Diese Registerkarte enthält Standardeinstellungen für die Dialoge, die in der Visualisierung zur Laufzeit für eine Texteingabe und für die Benutzerverwaltung verwendet werden. Welcher Dialog verwendet wird, legen Sie in der Eingabekonfiguration des betreffenden Visualisierungselements fest.



Die globalen Einstellungen im Visualisierungsmanager sind nur für die Verwendung in einer PLC HMI [▶ 635] oder PLC HMI Web [▶ 640] wirksam.

Einstellungen für Standardtexteingabe

Für ein Element mit Standardtexteingabe erscheint zur Laufzeit ein Dialog, der die Eingabe unterstützt. Sie können bestimmen, welcher Dialog erscheint.

Numpad	Dialog in Form eines Ziffernblocks, den die Visualisierung zur Laufzeit öffnet, wenn der Anwender das Eingabefeld für eine Zahl aktiviert. Voreinstellung: VisuDialogs.Numpad
Keypad	Dialog in Form einer Tastatur, den die Visualisierung zur Laufzeit öffnet, wenn der Anwender das Eingabefeld für einen Text aktiviert. Voreinstellung: VisuDialogs.Keypad
Texteingabe mit Begrenzung verwenden	Voraussetzung: Als Darstellungsvariante ist die PLC HMI [▶ 635] oder PLC HMI Web [▶ 640] konfiguriert und die Standardtexteingabe für die Variante ist Tastatur. Dann unterstützt die Visualisierung zur Laufzeit eine Eingabe per Tastatur. Für Eingaben mit begrenztem Wertebereich können Sie statt des Eingabefelds einen Dialog aufrufen, der den Wertebereich anzeigt. Voreinstellung: VisuDialogs.TextinputWithLimits Dieser Dialog zeigt den Wertebereich an und übernimmt keinen Wert außerhalb dieser Grenzen.
Globale Positionseinstellungen für Numpad / Keypad	• Zentriert: Der Dialog wird in der Mitte des Bildschirms geöffnet. • Mittig auf dem Element öffnen: Der Dialog wird auf dem Element geöffnet und überdeckt dieses. • Neben dem Element öffnen: Der Dialog wird dynamisch optimiert neben dem Element platziert. • An definierter Position öffnen: Der Dialog wird an der hier definierten Position im Visualisierungsfenster geöffnet. X, Y: Variable oder direkte Zahl (Pixel) zur Definition der linken oberen Ecke des Dialogs im Koordinatensystem des Visualisierungsfensters. Hinweis: Der Ursprung des Koordinatensystems der Visualisierung liegt in der oberen linken Ecke. Die positive horizontale X-Achse verläuft nach rechts. Die positive vertikale Y-Achse verläuft nach unten.

Einstellungen für Benutzerverwaltungsdialoge

Sie können Ihre Visualisierung mit einer Benutzerverwaltung konfigurieren. Dazu konfigurieren Sie eine Eingabe auf ein Element, die bewirkt, dass ein Benutzerverwaltungsdialog erscheint. Um andere Visualisierungen als Benutzerverwaltungsdialog zu verwenden, müssen Sie die Voreinstellungen hier ändern.

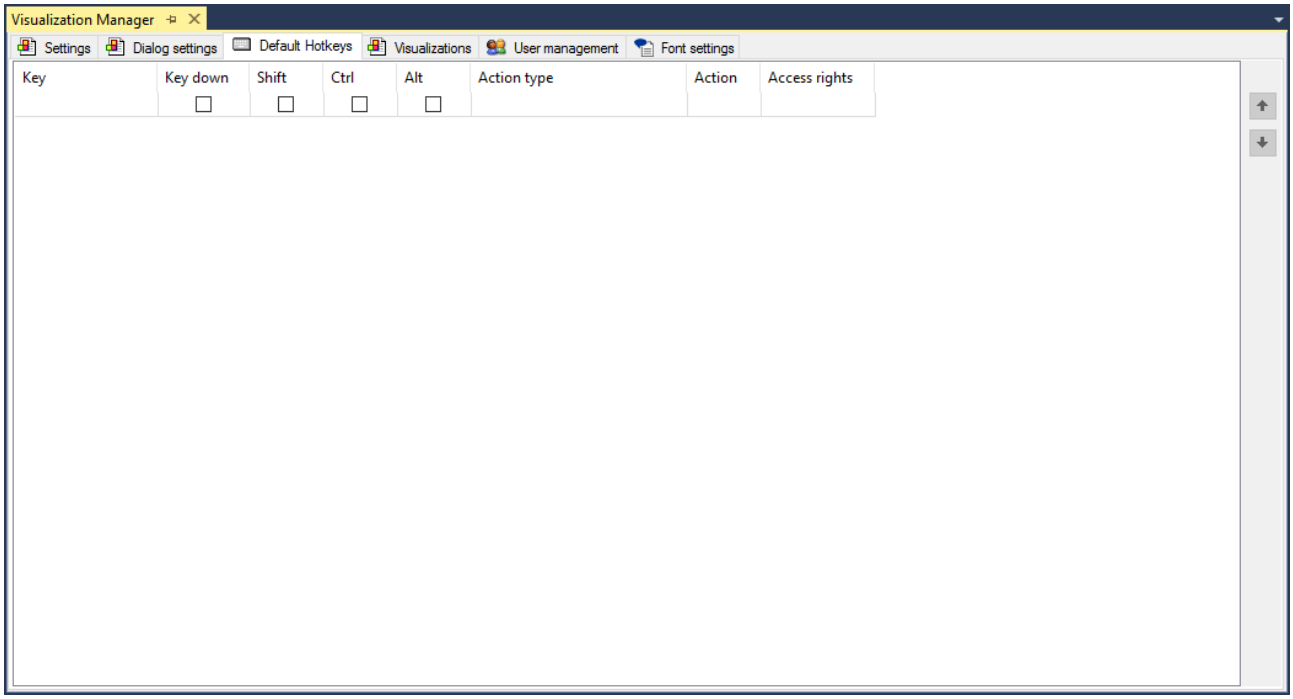
Dialog Einloggen	Benutzerverwaltungsdialog, der ein Einloggen ermöglicht, typischerweise eine Eingabeaufforderung für einen Benutzernamen und ein Passwort. Der Dialog erscheint auf ein Eingabeereignis auf ein Element, das als Folgeaktion Benutzerverwaltung, Aktion Anmelden ausführt. Voreinstellung: VisuUserManagement.VUM_Login
Dialog Passwortänderung	Benutzerverwaltungsdialog, der eine Passwortänderung ermöglicht, typischerweise eine Eingabeaufforderung für das aktuelle und das neue Passwort. Der Dialog erscheint auf ein Eingabeereignis auf ein Element, das als Folgeaktion Benutzerverwaltung, Aktion Benutzerpasswort ändern ausführt. Voreinstellung: VisuUserManagement.ChangePassword
Dialog Konfigurationsänderung	Benutzerverwaltungsdialog, der eine Konfigurationsänderung der Benutzerverwaltung ermöglicht, also typischerweise eine Anzeige der aktuellen Benutzerkonfiguration und eine Möglichkeit, diese zu ändern. Der Dialog erscheint bei einem Eingabeereignis, das als Folgeaktion Benutzerverwaltung, Aktion Benutzerverwaltung öffnen ausführt. Voreinstellung: VisuUserManagement.VUM_UserManagement

Siehe auch:

- Benutzerverwaltung [▶ 412]

15.2.3 Standardtastaturkürzel

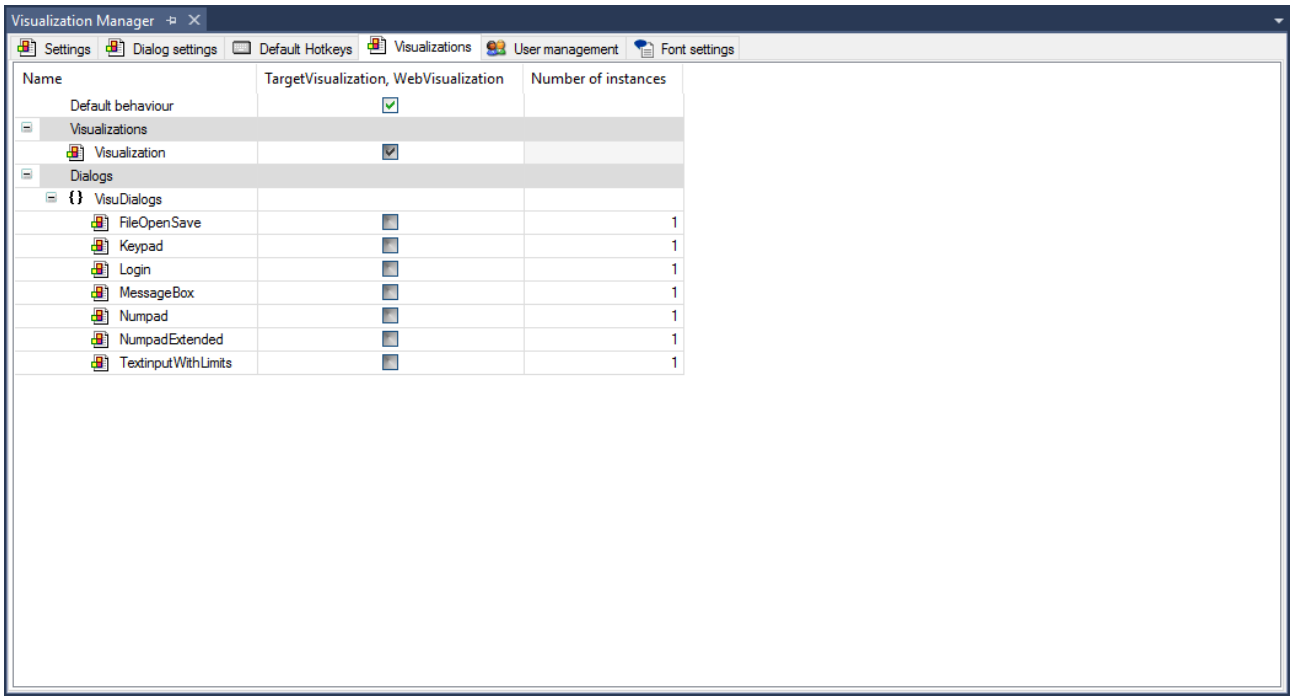
Diese Registerkarte enthält die Tastaturkonfiguration, die für alle Visualisierungsseiten des entsprechenden SPS-Projekts gilt. Das heißt, hier definierte Tasten/Tastenkombinationen können, wenn das jeweilige Gerät sie unterstützt, für Benutzereingaben in der Visualisierung im Onlinebetrieb verwendet werden.



Der Konfigurator ist genauso zu bedienen wie der, der im Visualisierungseditor für eine spezielle Visualisierung verwendet wird. Siehe somit die Beschreibung des Tastaturkonfigurationseditors [▶ 400]. Zusätzlich gelten immer geräteunabhängig einige bestimmte Standard-Tastaturkürzel [▶ 653] für das Navigieren in einer Visualisierung.

15.2.4 Visualisierungen

Diese Registerkarte listet alle verfügbaren Visualisierungen auf und ermöglicht für das Ladeverhalten eine Zuordnung der Visualisierungen abhängig von den Darstellungsvarianten.



Standardverhalten	Wenn diese Option aktiviert ist, werden die Visualisierungsobjekte des SPS-Projekts automatisch auf das jeweilige Zielsystem geladen. Welche das sind, zeigen die aktivierten Checkboxen an. Ist die Option nicht aktiviert, können Sie das Ladeverhalten für jede Visualisierung explizit festlegen.
Visualisierungen	Darunter sind alle erstellten Visualisierungen des Projekts aufgelistet.
Dialoge	Darunter sind alle Visualisierungen des Typs Dialog des Projekts und aus Bibliotheken aufgelistet. Hinweis: Unter der Spalte Anzahl Instanzen ist angegeben, wie oft der zugehörige Dialog instanziert wird.

Diese Funktionalität ersetzt die vorher mögliche Verwendung von "Visualisierungsreferenz" Objekten. Dieses Objekt kann jetzt nicht mehr neu eingefügt werden, bereits eingefügte funktionieren allerdings weiterhin.

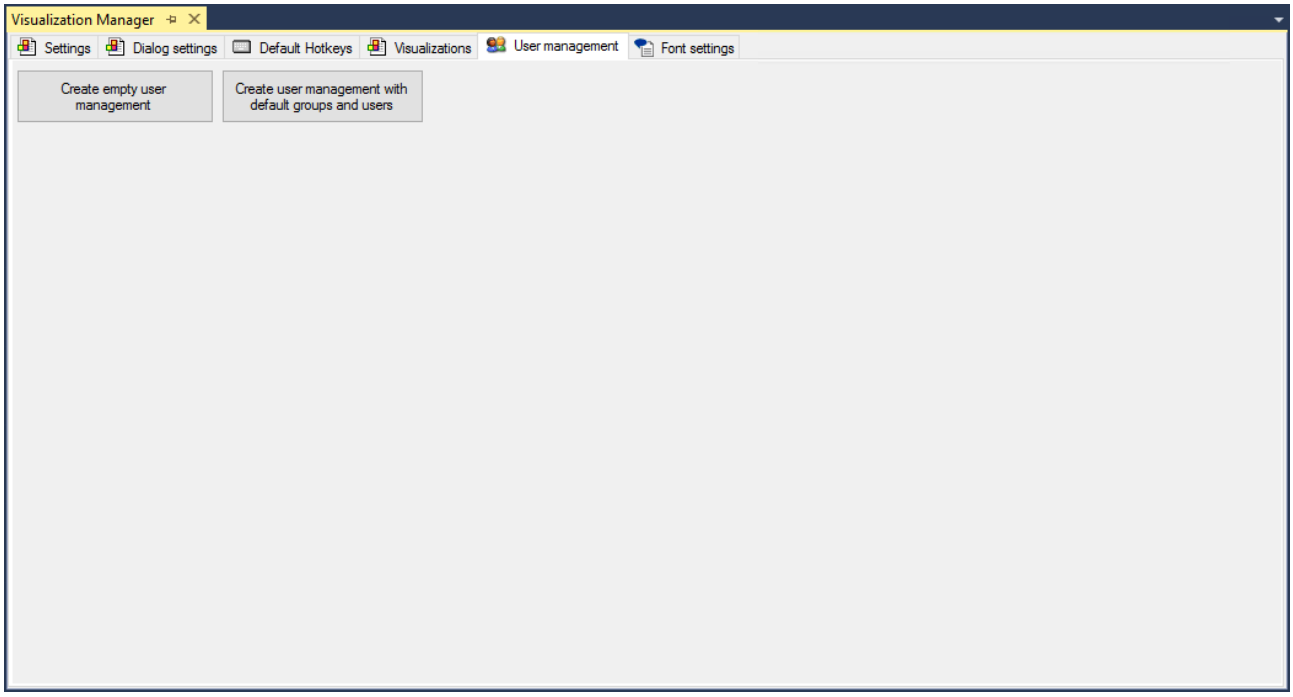
15.2.5 Benutzerverwaltung

Die Benutzerverwaltung wird verwendet, um die Sichtbarkeit und Bedienbarkeit von Visualisierungselementen benutzerabhängig zu handhaben. Auch das Umschalten von Visualisierungen kann benutzerabhängig konfiguriert werden. Die Benutzer werden in Gruppen organisiert.

Die Benutzerverwaltung kann nur in Kombination mit der PLC HMI [▶ 635] oder der PLC HMI Web [▶ 640] genutzt werden.

Erster Schritt

Zuerst müssen Sie eindeutige Gruppen- und Benutzer anlegen. Öffnen Sie die Registerkarte "Benutzerverwaltung" im Visualisierungsmanager-Editor. Ist bisher noch keine Benutzerverwaltungen konfiguriert, steht folgender Dialog zur Verfügung:

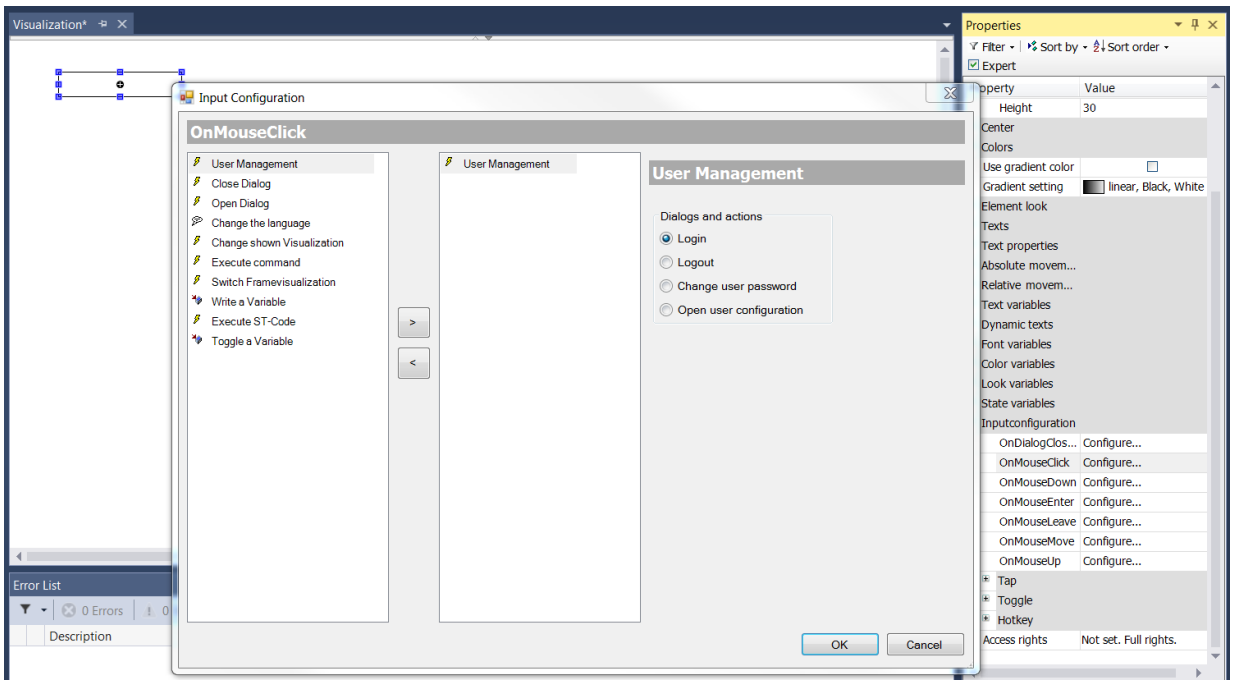


Leere Benutzerverwaltung anlegen	Die Benutzerverwaltung öffnet sich. Die Gruppe None ist angelegt.
Benutzerverwaltung mit Standardgruppen und -benutzern anlegen	Die Benutzerverwaltung öffnet sich. Folgende Gruppen und Benutzer sind angelegt: • Gruppe Admin mit Benutzer Admin • Gruppe Service mit Benutzer Service • Gruppe Operator mit Benutzer Operator • Gruppe None

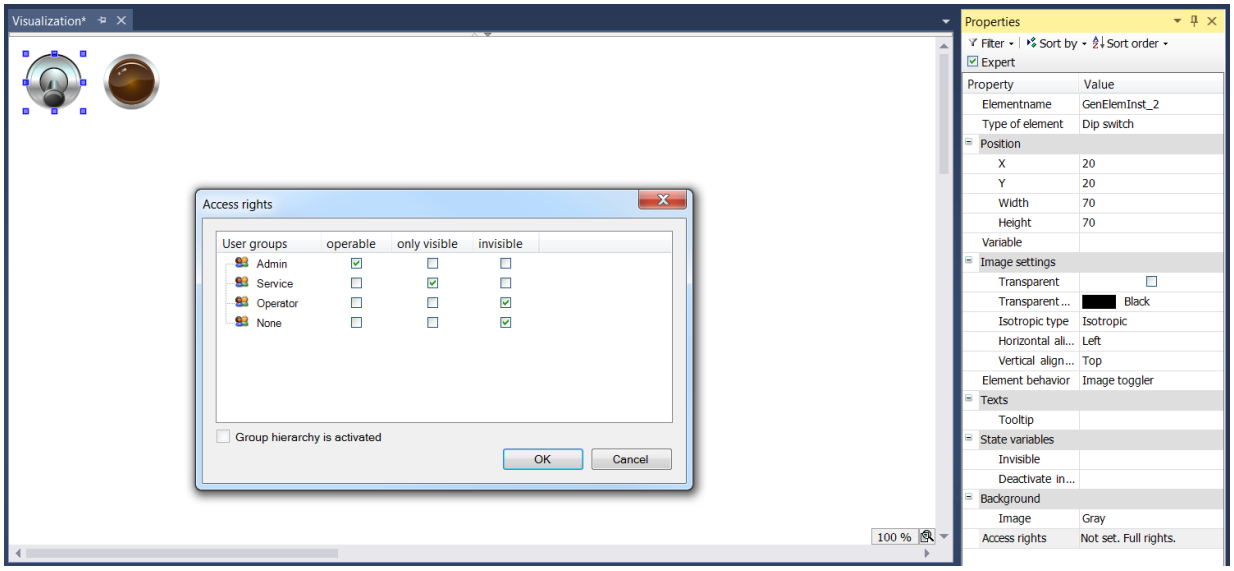
Die Programmierung der Visualisierung

1. Konfigurieren Sie Ihre Benutzerverwaltung, indem Sie Gruppen [▶ 415] und Benutzer [▶ 416] definieren.

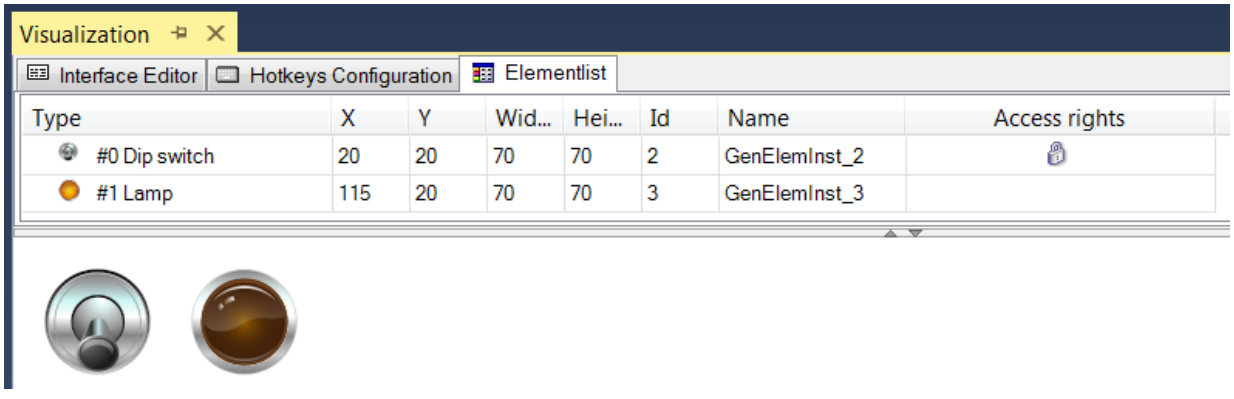
2. Integrieren Sie die Dialoge, welche die Bibliothek "VisuUserManagement" zur Verfügung stellt, in Ihre Visualisierung, indem Sie Elemente mit Eingabekonfiguration [▶ 458] programmieren. Auf ein definiertes Ereignis läuft ein Dialog programmiert mit Eingabekonfiguration ab. Sie können auch eigene Dialoge programmieren.



3. Programmieren Sie die Visualisierungselemente, indem Sie die Eigenschaft "Zugriffsrechte" in dessen Ansicht "Eigenschaft" setzen. Dann verhalten sie sich gruppenabhängig.



In der Elementliste sind die Elemente markiert, die eingeschränkte Rechte haben.



CSV-Datei mit den Daten des Benutzermanagements

Die Daten des Benutzermanagements werden als CSV-Datei in folgendem Format gespeichert:

- Benutzergruppen:

```
ID;group name;automatic logoff TRUE/FALSE;logoff time;unit logoff time;permission to change user data TRUE/FALSE
```

- Benutzer:

```
login name;full name;password encrypt TRUE/FALSE;password;group ID;user deactivated TRUE/FALSE
```

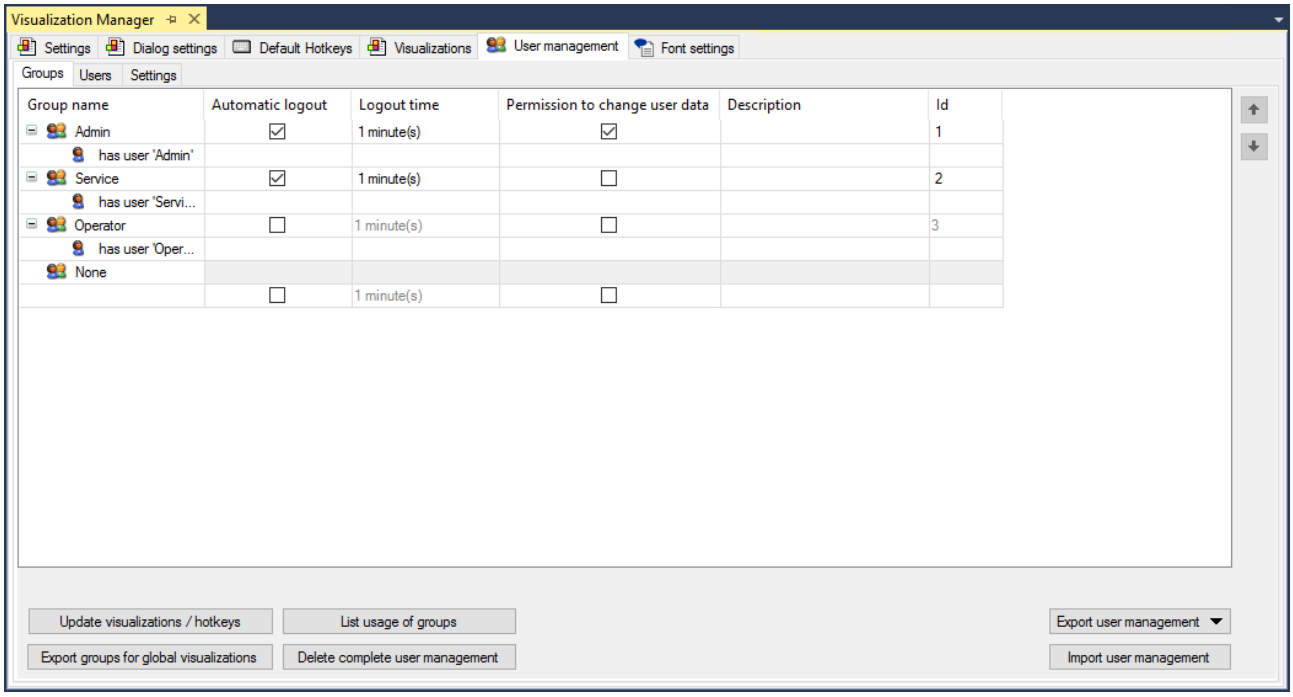
Verwenden Sie dieses Format, wenn Sie Daten der Benutzerverwaltung mit einem beliebigen Tool bearbeiten möchten. Wird "password encrypt" auf FALSE gesetzt, dann kann ein nicht verschlüsseltes Passwort eingegeben werden, wie im Beispiel bei Benutzer "Hugo". Nach einem Import wird es sofort verschlüsselt.

Beispiel:

```
V1.0.0.1
Usergroups:
1;Admin;TRUE;1;Minute;TRUE
2;Service;FALSE;5;Minute;FALSE
3;Operator;FALSE;1;Minute;FALSE
0;None;FALSE;1;Minute;FALSE
User:
HansM;Hans Mayer;TRUE;F9307D9940B6F7D78320E7E008377593;1;FALSE;administrator
PeterS;Peter Schmidt;TRUE;C5972629BF18E0E82D06FFF29B5BADFF;2|3;FALSE;team leader 1
UllaM;Ulla Müller;TRUE;569D35AC3272623AECDDCA021916C2AB;2|3;FALSE;team leader 2
ElkeF;Elke Fischer;TRUE;C634F54AF9343142159FE0435D93929D;3;FALSE;operator team 1
PaulK;Koch;TRUE;01E2CBD4AE5442D9EACE33669549A3CC;3;FALSE;operator team 2
```

15.2.5.1 Gruppen

In der Ansicht "Gruppen" werden alle Gruppen in hierarchischer Reihenfolge aufgelistet.



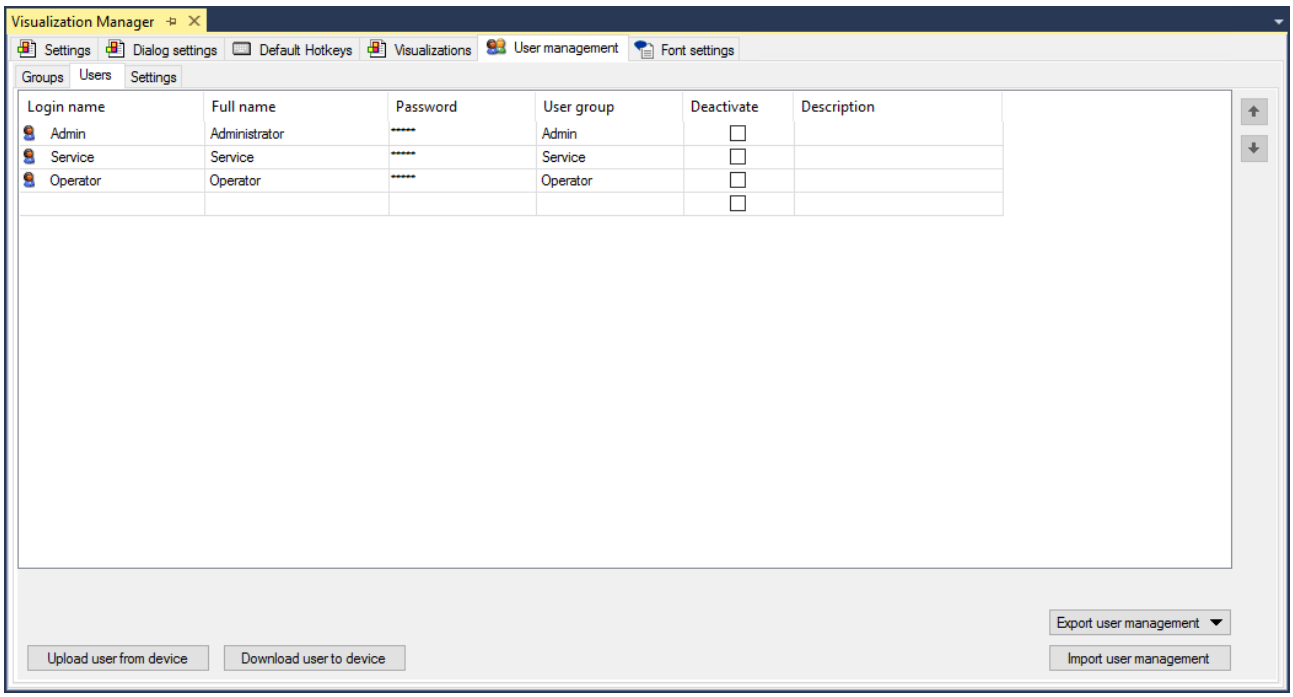
Gruppenname	Wird der Knoten der Gruppe expandiert, werden alle Benutzer, die dieser Gruppe zugeordnet sind, angezeigt.
Automatisches abmelden	Markieren Sie die Checkbox, um fest Abmeldezeit zu definieren.
Abmeldezeit	Geben Sie hier die Zeit ein, nach der der Benutzer abgemeldet werden soll. Verwenden Sie den Zeileneditor, um eine Zahl einzugeben, und die Auswahlliste, um die Einheit zu setzen.
Recht zum Ändern der Benutzerdaten	Markieren Sie die Checkbox, um dieser Gruppe das Recht einzuräumen, die Benutzerdaten zu bearbeiten, wenn die Visualisierung online ist.
Beschreibung	Hier können Sie Kommentare und Bemerkungen zu den Gruppen eingeben. Dieser Text ist nur im Programmiersystem verfügbar und wird nicht in die Laufzeit geladen.
ID	Eindeutige ID für jede Gruppe. Wird vom System automatisch vergeben.
Gruppe hinzufügen	Klicken Sie in die Editierzeile am Ender der Tabelle in der Spalte Gruppe, um eine neue Gruppe anzulegen.
Gruppe löschen	Markieren Sie eine Gruppe, indem Sie ein Feld in der zugehörigen Zeile der Tabelle selektieren. Verwenden Sie [Entf], um die Zeile und damit eine Gruppe zu löschen. None kann nicht gelöscht werden.

Schaltflächen

Visualisierungen / Tastaturkürzel aktualisieren	Öffnet den Dialog „Visualisierungen und Hotkeys aktualisieren“. Aktualisierung, wenn Gruppen zu einem Zeitpunkt geändert wurden, als Visualisierungen oder Tastaturkürzel bereits eingeschränkte Zugriffsmöglichkeiten hatten.
Verwendung der Gruppen auflisten	Auflistung der Visualisierungen und Tastaturkürzel mit eingeschränkten Zugriffsrechten. Die Auflistung wird in der Ansicht Meldungen angezeigt.
Gruppen für globale Visualisierungen exportieren	Mit einem Klick auf diesen Button werden die oben definierten Gruppen nach Extras > Optionen > TwinCAT > PLC Environment > Visualisierung Benutzerverwaltung übertragen und dort unter "Folgende Benutzergruppenliste verwenden" aufgeführt.
Benutzerverwaltung löschen	Die Benutzerverwaltung wird gelöscht, aber die Zugriffsrechte der Elemente bleiben bestehen.
Benutzerverwaltung exportieren	Ein Standarddialog öffnet sich, um eine CSV-Datei in einem beliebigen Verzeichnis mit beliebigem Namen zu speichern.
Benutzerverwaltung importieren	Ein Standarddialog öffnet sich, um eine CSV-Datei, die sich in einem beliebigen Verzeichnis mit beliebigem Namen befindet, zu laden
 , 	Wollen Sie die Gruppen hierarchisch in einer bestimmten Reihenfolge organisieren, dann klicken Sie auf das Pfeil-nach-oben Symbol, um die Gruppe um eine Zeile nach oben zu verschieben, beziehungsweise auf das Pfeil-nach-unten Symbol, um die Gruppe um eine Zeile nach unten zu verschieben. Die Reihenfolge in der Tabelle reflektiert die hierarchische Reihenfolge, die so in die Laufzeit geladen werden kann und dann im Onlinebetrieb zur Verfügung steht. Eine Gruppe einer höheren Hierarchie kann nicht weniger Zugriffsrechte für ein Element haben als eine Gruppe niedrigerer Hierarchie.

15.2.5.2 Benutzer

In der Ansicht "Benutzer" werden alle Benutzer aufgelistet.



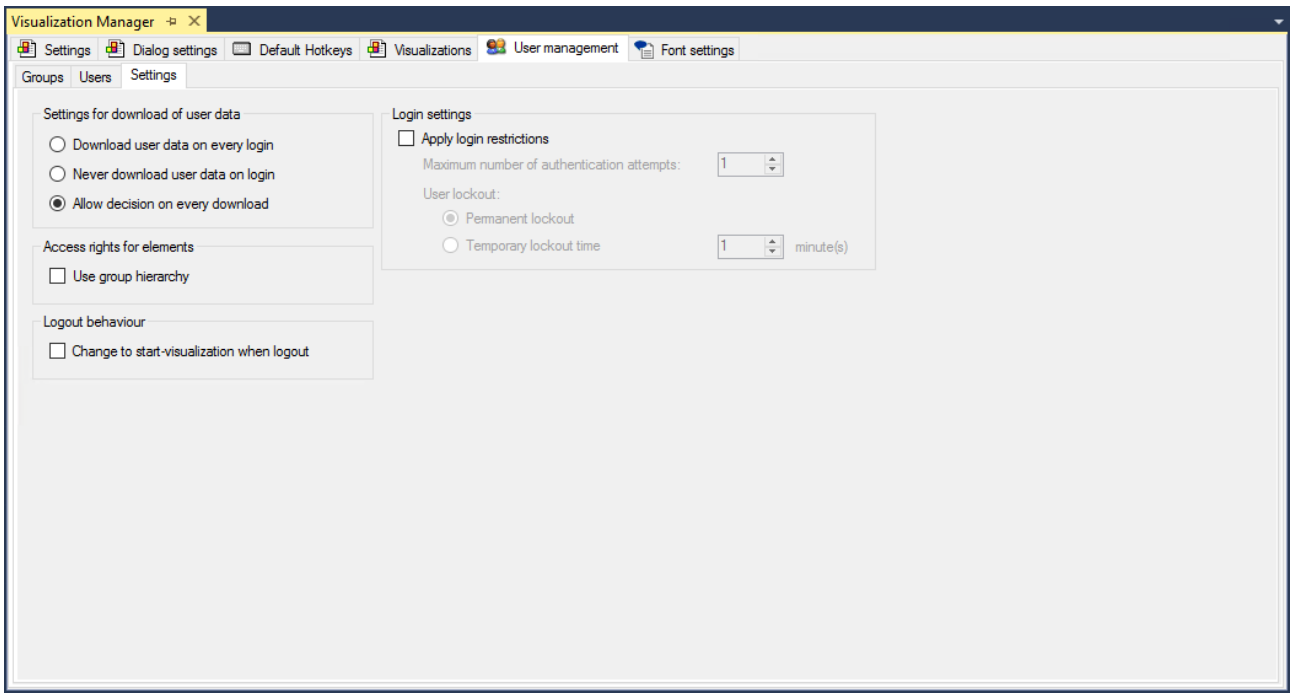
Anmeldenamen	Geben Sie einen eindeutigen und kompilierbaren Namen ein, mit dem sich der Benutzer zur Laufzeit in der Visualisierung anmeldet.
Vollständiger Name	Geben Sie den vollständigen Namen des Benutzers ein. Dieser darf mehrmals in der Benutzerverwaltung existieren.
Passwort	Geben Sie ein Passwort ein, dass verschlüsselt wird. Jeder Benutzer bekommt standardmäßig seinen Anmeldenamen als Passwort.
Benutzergruppe	Geben Sie eine oder mehrere Gruppe(n) an, der der Benutzer angehört. Gehört ein Benutzer mehreren Benutzergruppen an, so kann er in der Visualisierung ein Element bedienen oder sehen, sobald eine dieser Gruppen das Recht dazu hat. Weiter Details finden Sie im Kapitel Zugriffsrechte [▶ 443].
Deaktivieren	Markieren Sie die Checkbox, wenn der Benutzer deaktiviert werden soll.
Beschreibung	Hier können Sie Kommentare und Bemerkungen zum Benutzer eingeben. Dieser Text ist nur im Programmiersystem verfügbar und wird nicht in die Laufzeit geladen.

Schaltflächen

Benutzer von Gerät laden	Hier werden die Daten der Benutzerverwaltung aus der SPS hochgeladen. Sind bereits Benutzerdaten konfiguriert, dann werden sie überschrieben.
Download der Benutzer auf Gerät	Hier werden die Daten des Benutzermanagements in die SPS geladen. Die bisherige Benutzerverwaltung wird dabei überschrieben.
Benutzerverwaltung exportieren	Ein Standarddialog öffnet sich, mit dem eine CSV-Datei in einem beliebigen Verzeichnis mit beliebigem Namen gespeichert werden kann. Die CSV-Datei enthält die Daten der Gruppen und Benutzer.
Benutzerverwaltung importieren	Ein Standarddialog öffnet sich, mit dem eine CSV-Datei, die in einem beliebigen Verzeichnis mit beliebigem Namen liegt, geladen und hier angezeigt werden kann, wenn das Format kompatibel ist.

15.2.5.3 Einstellungen

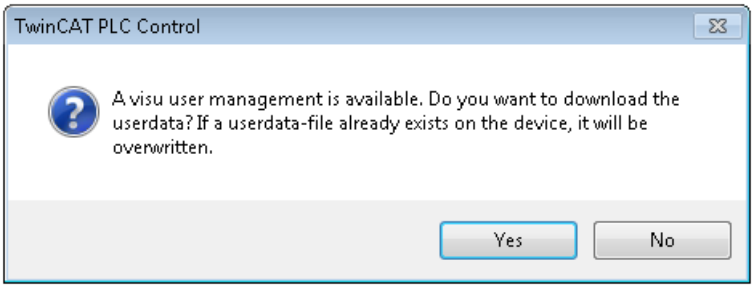
Bei dieser Ansicht handelt es sich um die Einstellungen für den Download der Benutzerdaten.



Wählen Sie ein Verfahren zum Einloggen:

Einstellungen für den Download der Benutzerdaten	
Download bei jedem Einloggen	Die Daten der Benutzerverwaltung, die im Programmiersystem gespeichert sind, werden mit jedem Einloggen auf die SPS geladen. Bereits existierende Daten werden überschrieben.
Kein Download beim Einloggen	Die Daten der Benutzerverwaltung werden niemals heruntergeladen, auch wenn sie sich geändert haben sollten.
Bei jedem Download neu entscheiden	Ein Dialog geführter Download wird ermöglicht.

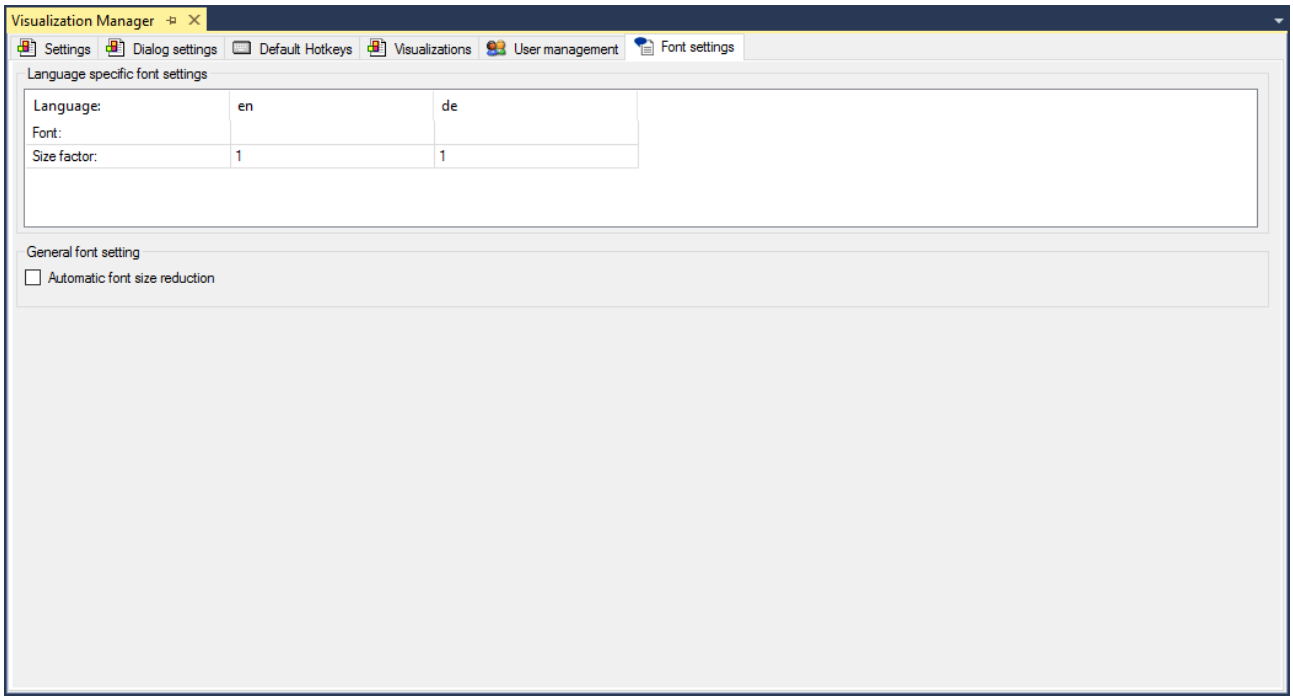
Falls die Einstellung "Bei jedem Download neu entscheiden" gewählt worden ist, wird der folgende Dialog vor jedem Download angezeigt.



Zugriffsrechte für Elemente	
Gruppenhierarchie benutzen	Die Gruppen sind entsprechend ihrer Position in ihrer Konfiguration Gruppen hierarchisch angeordnet und Zugriffsrechte lassen sich dementsprechend nur hierarchisch setzen.
Logout-Verhalten	
Beim Logout zur Startvisualisierung wechseln	Markieren Sie die Checkbox, wenn sich nach dem Ausloggen eines Benutzers automatisch die Startvisualisierung öffnen soll.
Login-Einstellungen	
Login-Beschränkungen anwenden	Markieren Sie die Checkbox, um die Einstellungen "Maximale Anzahl Authentifizierungsversuche" und "Benutzersperre" für die Visualisierungen der SPS zu verwenden.
Maximale Anzahl Authentifizierungsversuche	Definieren Sie die maximal mögliche Anzahl von Authentifizierungsversuchen. Mögliche Werte: [1 … 10] Wenn die maximale Anzahl von Authentifizierungsversuchen erreicht wird, wird der Benutzer, abhängig von der Einstellung "Benutzersperre", temporär oder permanent gesperrt.
Benutzersperre	Wählen Sie einen der beiden Sperrtypen "permanent" oder "temporär" aus: • Permanente Sperre: Der Benutzer wird dauerhaft gesperrt. Benutzer mit dem Recht zum Ändern der Benutzerdaten können nicht dauerhaft gesperrt werden. • Temporäre Sperre, Zeit: Angabe in Minuten, mögliche Werte [1 … 2880]. Benutzer mit dem Rechte zum Ändern der Benutzerdaten haben als temporäre Sperre immer 10 Minuten, auch wenn an dieser Stelle ein anderer Wert eingetragen wird.

15.2.6 Schriftart

Diese Registerkarte bietet Einstellungen, um sprachabhängig Schriftart und -größe eines Textes anzupassen. Sie gelten für alle Visualisierungen des Projektes.



Sprachspezifische Schriftarteinstellungen

Sprache	Es gibt eine Auswahl vorgegebener Sprachen. Die im Projekt verwendeten Sprachen erweitern diese Auswahl. Dafür werden alle Textlisten des Projekts durchsucht.
Schriftart	Die Schriftart, welche die Visualisierung abhängig von der Sprache verwendet.
Größenfaktor	Der Faktor wirkt auf die Schriftgrößen aller Texte in der Visualisierung. Wenn der Faktor kleiner 1 ist, führt das zu einer Verkleinerung der Schriftgröße. Wenn der Faktor 1 ist, werden alle Texte unverändert, wie in den Eigenschaften definiert, dargestellt. Voreinstellung: 1
Rote Hervorhebung einer Zelle	Die Einstellung kann zur Laufzeit nicht angewendet werden, da die entsprechende Sprache in den Textlisten des Projekts oder der Bibliotheken nicht mehr vorhanden ist.

Kontextmenü einer selektierten Tabellenzeile	
Löschen	Die zugehörige Spalte wird entfernt. Das ist vor allem dann ratsam, wenn Einstellungen der Spalte rot hervorgehoben sind.
Kopieren	Alle Einstellungen der Spalte werden in die Zwischenablage übernommen.
Einfügen	Alle Einstellungen der Spalte werden mit den Werten aus der Zwischenablage überschrieben.

Allgemeine Schriftarteneinstellung

Automatisches Verkleinern der Schriftgröße	Wenn der darzustellende Text in der eingestellten Formatierung nicht in das Textfeld passt, wird die Schriftgröße automatisch verkleinert, bis der Text vollständig in das Textfeld passt. Tipp: Damit wird bei einer Sprachumschaltung in eine Sprache, die mehr Platz benötigt, verhindert, dass ein Text nicht vollständig angezeigt wird. Vorausgesetzt, es steht eine ausreichend kleine Schriftart zur Verfügung.

Siehe auch:

- Text und Sprache in Visualisierungen [▶ 646]

15.3 Visualisierungsbibliotheken

Die als Funktionsblock programmierten Visualisierungselemente [▶ 424] werden über Bibliotheken bereitgestellt. Wenn ein Visualisierungsobjekt [▶ 422] im Projekt eingefügt wird, werden bestimmte Visualisierungsbibliotheken im SPS-Projekt eingebunden. Die Namen und Versionen dieser Bibliotheken sind im aktuell verwendeten Visualisierungsprofil [▶ 421] definiert. Das Profil legt auch genau fest, welche Elemente aus diesen Bibliotheken in dem Werkzeugkasten [▶ 402] des Visualisierungseditors [▶ 394] bereitstehen.

Eine Visualisierungsbibliothek ist immer als spezieller Typ einer ‚Platzhalter-Bibliothek' angelegt. Das bewirkt, dass die genaue Version der zu verwendenden Bibliothek nicht festgelegt ist, solange sie nicht in einem Projekt eingebunden wird. Erst dann bestimmt das aktuelle Visualisierungsprofil, welche Version tatsächlich benötigt wird. Beachten Sie, dass dieser Typ von Bibliothek sich von den gerätespezifischen Platzhalter-Bibliotheken unterscheidet, bei denen die Platzhalter aus der Gerätebeschreibung aufgelöst werden.

Siehe nachfolgend die Basis-Bibliotheken, die standardmäßig eingebunden werden, sobald ein Visualisierungsobjekt in ein Standardprojekt eingefügt wird. Sie referenzieren weitere Bibliotheken, die hier nicht aufgeführt sind. Im Standardfall müssen Sie die Visualisierungsbibliotheken nicht explizit einfügen oder in Ihren Applikationen verwenden:

- System_VisuElem3DPath (nur für TwinCAT 3.1 Build <4020.0)
- System_VisuElemMeter
- System_VisuElems
- System_VisuElemsDateTime (nur für TwinCAT 3.1 Build <4020.0)
- System_VisuElemsSpecialControls

- System_VisuElemsWinControls
- System_VisuElemTextEditor
- System_visuinputs
- System_VisuNativeControl

15.4 Voraussetzungen

Da die TwinCAT-3-Visualisierung nach Standard IEC61131-3 realisiert ist, müssen bestimmte Bibliotheken [▶ 420] im Projekt eingebunden sein, die die erforderlichen Funktionen und Visualisierungselemente bereitstellen. Um die Auswahl der Visualisierungsbibliotheken und daraus wiederum genau die Auswahl an Elementen zu definieren, die im Visualisierungseditorobjekt bereitstehen sollen, werden Visualisierungsprofile [▶ 421] verwendet. Mit dem Hinzufügen der ersten Visualisierungsseite werden die durch das Profil vorgegebenen Bibliotheken automatisch zum SPS-Projekt hinzugefügt.

15.5 SPS-Projekteigenschaften

In den SPS-Projekteigenschaften können Standardeinstellungen geändert werden, beispielsweise in der Kategorie "Visualization" die Standardverzeichnisse für Textlistendateien und Bilddateien. Der Dialog kann über einen Rechtsklick auf das SPS-Projekt im Kontextmenü unter "Eigenschaften" geöffnet werden.

15.6 Visualisierungsprofile

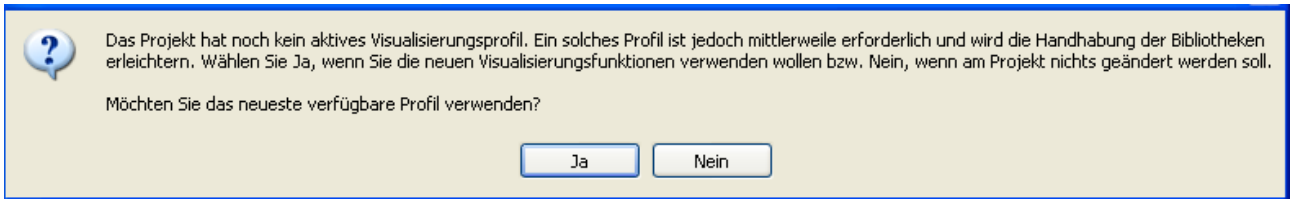
Jedes Visualisierungsprojekt, also ein Projekt, das mindestens ein Visualisierungsobjekt [▶ 422] enthält, muss auf einem Visualisierungsprofil basieren. Dieses Profil definiert folgendes:

- Die Namen und Versionen der Visualisierungsbibliotheken [▶ 420] die automatisch im Projekt eingebunden werden, sobald ein Visualisierungsobjekt angelegt [▶ 422] wird.
- Eine Auswahl von aus den eingebundenen Bibliotheken stammenden Visualisierungselementen, die in dem Werkzeugkasten [▶ 402] des Visualisierungseditors bereitstehen sollen.

Welches Profil standardmäßig im Projekt verwendet wird, ist in den Projekteigenschaften [▶ 421] unter "Visualization Profile" definiert. Dort kann auch jederzeit auf ein anderes Profil umgeschaltet werden. Beachten Sie, dass das gewählte Profil für alle Visualisierungsobjekte des Projekts gilt.

Wenn das Profil gewechselt wird, erscheint eine Meldung, dass dies möglicherweise ein Einloggen ohne Online-Change oder Download verhindert. Das Wechseln des Profils verursacht eine automatische Aktualisierung im Bibliotheksverwalter bezüglich der erforderlichen Bibliotheken. Diese müssen nicht händisch angepasst werden.

Wenn ein altes Projekt geöffnet wird, das noch nicht mit einem Visualisierungsprofil angelegt wurde, werden Sie gefragt, ob auf den neuen Profil-Mechanismus umgeschaltet werden soll oder nicht.



Wenn "Ja" geantwortet wird, wird das neueste Profil verwendet. Bei "Nein" wird das älteste verfügbare Profil in den Projekteinstellungen eingetragen (genannt "Kompatibilitätsprofil"), aber im Projekt werden keine weiteren Änderungen vorgenommen.

15.7 Visualisierungsobjekt

Eine Visualisierung [▶ 393] in einem SPS-Projekt kann sich aus verschiedenen Visualisierungsseiten zusammensetzen. Eine Visualisierungsseite kann in einer anderen verwendet [▶ 644] werden und es ist möglich, zwischen verschiedenen Seiten zur Laufzeit zu wechseln [▶ 645]. Jede dieser Visualisierungsseiten wird durch ein eigenes Visualisierungsobjekt dargestellt.

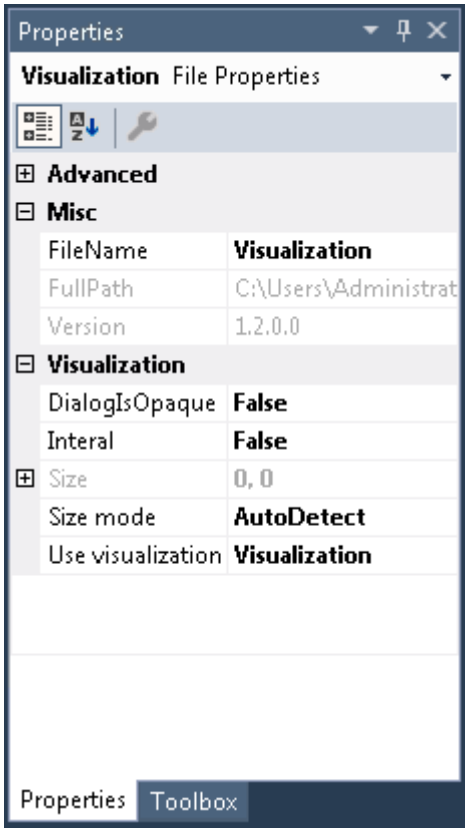
Visualisierungsobjekt anlegen

Um ein Visualisierungsobjekt in einem SPS-Projekt anzulegen, kann mithilfe eines Rechtsklicks auf das SPS-Projekt oder einer der darunter liegenden Ordner (Vorgesehen ist "Visus") im SPS-Projekt ein Kontextmenü geöffnet werden. Über den Menüpunkt "Add" kann dann ein Visualisierungsobjekt ausgewählt werden.

Beim Hinzufügen des ersten Visualisierungsobjekts werden automatisch die Visualisierungsbibliotheken [▶ 420] und der Visualisierungsmanager [▶ 405] zum Projekt hinzugefügt. Das neu angelegte Visualisierungsobjekt wird im Visualisierungseditor [▶ 394] geöffnet.

Einstellungen eines Visualisierungsobjekts

Wenn ein Visualisierungsobjekt selektiert wird, werden dessen Einstellungen wie folgt im Eigenschaftenfenster angezeigt:



Misc

FileName	Name des Visualisierungsobjekts
FullPath	Speicherort des Objekts. An dieser Stelle kann der Speicherort nicht geändert werden. Aus diesem Grund wird er ausgegraut dargestellt.

Visualization

Dialog Is Opaque	Wenn diese Einstellung aktiv ist, wird der von diesem Dialog verdeckte Bildschirmbereich nicht aktualisiert. Dies wirkt sich positiv auf die Zeichen- und Eingabeperformance aus. Hinweis: Nutzen Sie diese Option nur, wenn der von Ihnen gezeichnete Dialog rechteckig und volldeckend ist, also keine transparenten Anteile enthält.
Internal	Eine als intern gekennzeichnete Visualisierung ist innerhalb eines SPS-Projektes wie gewohnt sichtbar und verwendbar. Wenn diese Visualisierung in einer Bibliothek abgespeichert wird, ist sie in dem SPS-Projekt, in dem die Bibliothek verwendet wird, nicht mehr sichtbar und verwendbar.
Size	Größe der Visualisierungsseite – Die Einstellung ist ausgegraut, falls unter "Size mode" der Eintrag "AutoDetect" ausgewählt worden ist. • Width: Breite in Pixel • Height: Höhe in Pixel
Size mode	Hier kann eingestellt werden, ob und wie sich die Größe der Visualisierungsseite anpassen soll. • AutoDetect: Die Größe der Visualisierungsseite wird ermittelt, bei der alle gerade enthaltenen Visualisierungselemente sichtbar sind. • AutoDetectWithBgImage: Die Größe der Visualisierungsseite wird ermittelt, bei der alle gerade enthalten Visualisierungselemente und das Hintergrundbild [▶ 395] sichtbar sind. • Specified: Die Größe der Seite wird unter "Size" festgelegt. Dabei ist zu berücksichtigen, ob alle Visualisierungselemente und gegebenenfalls das Hintergrundbild in diesen Bereich passen und damit vollständig sichtbar sind. Falls das aus diesen Möglichkeiten resultierende Seitenverhältnis nicht zum Bildschirm passt, wird die Visualisierungsseite mit entsprechenden weißen Rändern angezeigt. Auf diese Weise wird eine Verzerrung der Visualisierungselemente verhindert.
Use visualization as	In dieser Einstellung kann einer der folgenden Visualisierungstypen für das Objekt festgelegt werden: • Visualization: Bei dieser Standardeinstellung ist das Visualisierungsobjekt als eine eigenständige Visualisierungsseite deklariert. • Dialog: Das Visualisierungsobjekt stellt einen Dialog dar. Wenn diese Einstellung aktiviert worden ist, kann das Visualisierungsobjekt als Dialog [▶ 428] in anderen Visualisierungsobjekten genutzt werden. • NumKeybad: Das Visualisierungsobjekt stellt ein Nummernfeld/ eine Tastatur dar. Wenn diese Einstellung aktiviert worden ist, kann das Visualisierungsobjekt als Nummernfeld oder Tastatur zum Beispiel für das Beschreiben einer Variablen [▶ 436] in der Visualisierung genutzt werden. Die Schnittstelle eines solchen Nummernfelds/ einer Tastatur muss genauso aussehen wie die des Standardnummernfeldes/ der Standardtastatur, die durch die Bibliothek "VisuDialogs" bereitgestellt werden.

Visualisierungsobjekt bearbeiten

Der Visualisierungseditor [▶ 394] zum Erstellen von Visualisierungen arbeitet in Kombination mit dem Werkzeugkasten [▶ 402], die die Visualisierungselemente aus den eingebundenen Elemente-Bibliotheken bereitstellt, sowie mit dem Eigenschafteneditor [▶ 403] zur Konfiguration der Visualisierungselemente. Um ein Visualisierungsobjekt zu öffnen, führen Sie einen Doppelklick auf das Objekt im Projektbaum aus.

Definition einer Startvisualisierung

Die "Startvisualisierung", das heißt das Visualisierungsobjekt, welches als erstes beim Start eines PLC HMI [▶ 635] oder PLC HMI Web [▶ 640] Clients dargestellt werden soll, muss im TargetVisualization [▶ 638]- beziehungsweise WebVisualization [▶ 641]-Objekt eingetragen werden.

15.8 Visualisierungselemente

Die verschiedenen Visualisierungselemente stehen nach dem Öffnen eines Visualisierungsobjekts [▶ 422] im Werkzeugkasten [▶ 402] zur Verfügung. Sie sind in die folgenden Kategorien unterteilt:

- Allgemeine Steuerelemente [▶ 441]
- Basis [▶ 499]
- Lampen/ Schalter/ Bilder [▶ 556]
- Messgeräte [▶ 567]
- Spezielle Steuerelemente [▶ 612]

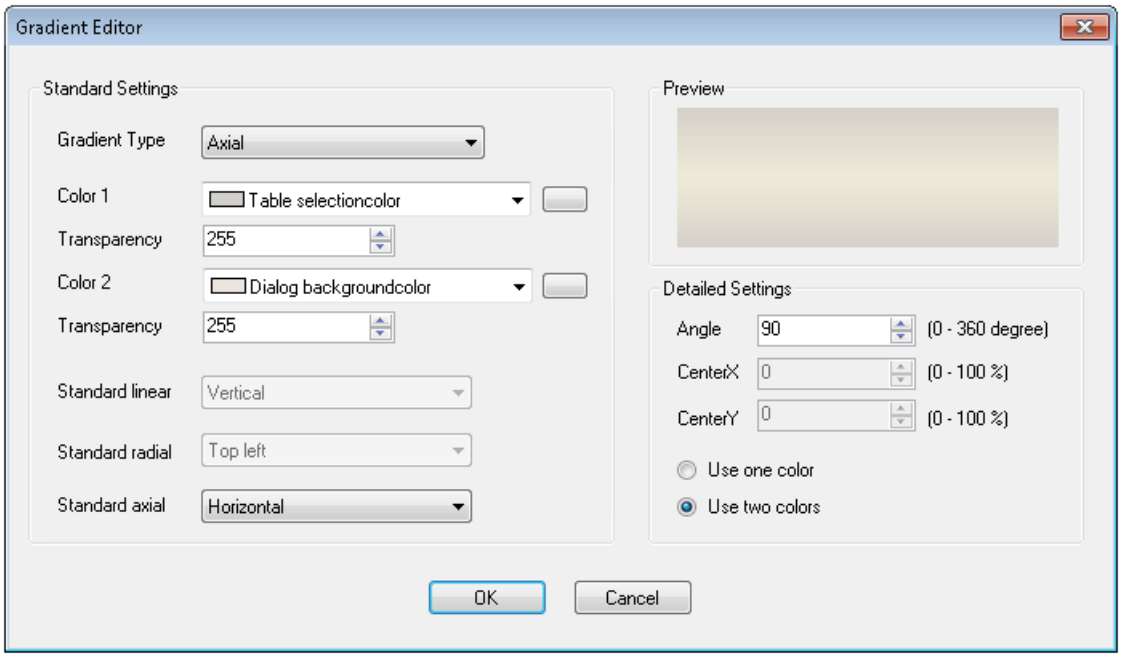
15.8.1 Allgemeine Konfiguration

Es gibt die folgenden drei allgemein gültigen Dialoge, die bei verschiedenen Visualisierungselementen genutzt werden:

- Farbverlaufdialog [▶ 424]
- Eingabekonfiguration [▶ 425]
- Zugriffsrechtedialog [▶ 440]

15.8.1.1 Farbverlaufeditor

Ein Mausklick in das Wertefeld der Farbverlaufauswahl im Eigenschaftenfenster öffnet den Farbverlaufeditor.



Standardeinstellungen

Farbverlauf	Es kann zwischen den drei folgenden Farbverlauftypen ausgewählt werden: • Linear • Radial • Axial
Farbe 1	Erste Farbe des Gradienten – Sie kann aus der Combobox oder aus dem Farbdialog, zu öffnen über die Schaltfläche  , gewählt werden.
Transparenz	Mit einem Wert zwischen 0 und 255 kann die Stärke der ersten Farbe festgelegt werden.
Farbe 2	Zweite Farbe des Gradienten – Sie kann aus der Combobox oder aus dem Farbdialog, zu öffnen über die Schaltfläche  , gewählt werden.
Transparenz	Mit einem Wert zwischen 0 und 255 kann die Stärke der zweiten Farbe festgelegt werden.
Standard linear	Vordefiniert Einstellungen des Farbverlauftyps Linear (Winkel des Gradienten): • Horizontal • Vertikal • Von oben links • Von oben rechts • Horizontale Farbe umgeschaltet • Vertikale Farbe umgeschaltet • Von unten links • Von unten rechts
Standard radial	Vordefinierte Einstellungen des Farbverlauftyps Radial (Position der Mitte): • Mitte • Oben links • Oben rechts • Unten links • Unten rechts
Standard axial	Vordefinierte Einstellungen des Farbverlauftyps Axial (Winkel des Verlaufs): • Horizontal • Vertikal • Von oben rechts • Von oben links

Detaillierte Einstellungen

Winkel (Grad)	Nur für Farbverlauftyp Linear und Axial verfügbar
Mitte X (%)	X-Position des Mittelpunktes (0-100%) – Nur für Farbverlauftyp Radial verfügbar.
Mitte Y (%)	Y-Position des Mittelpunktes (0-100%) – Nur für Farbverlauftyp Radial verfügbar.
Eine Farbe benutzen	Farbverlauftyp zwischen Farbe 1 und der gleichen Farbe mit anderer Helligkeit – Die Helligkeit kann von 0 (schwarz) bis 100 (weiß) eingestellt werden.
Zwei Farben benutzen	Farbverlaufstyp zwischen den beiden ausgewählten Farben "Farbe 1" und "Farbe 2".

15.8.1.2 Eingabekonfiguration

Eine oder mehrere der im Folgenden beschriebenen Nachfolgeaktionen können im Dialog Eingabekonfiguration für ein bestimmtes Ereignis [▶ 458] (z.B. OnMouseClick) des Elements definiert werden. Der Dialog öffnet auf einen Mausklick in das Wertefeld dieses Ereignisses im Eigenschaftenfenster. Der Name des Ereignisses, für das diese Nachfolgeaktionen selektiert werden, ist als Titel in der grauen Zeile sichtbar.

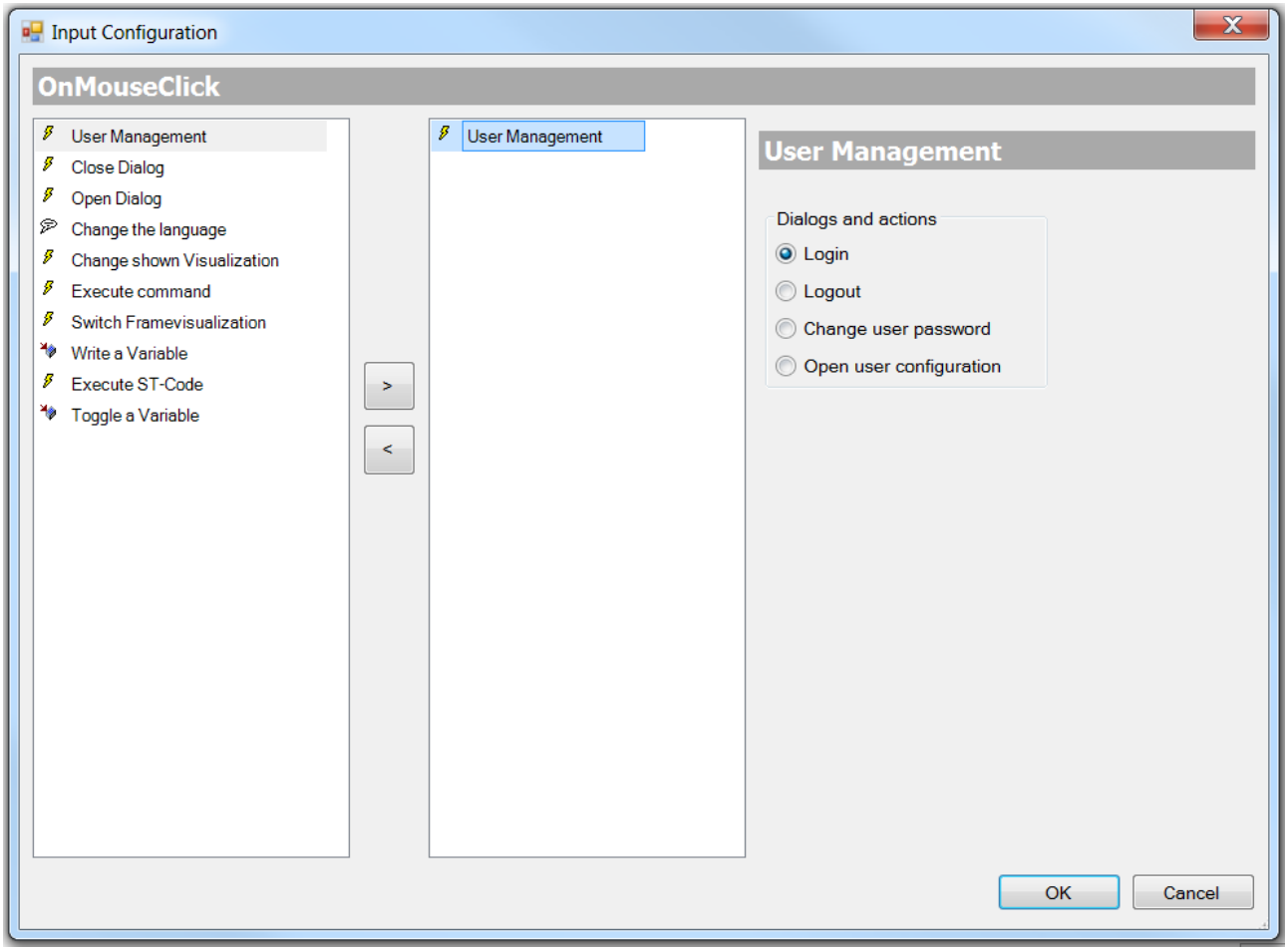
Um eine Nachfolgeaktion hinzuzufügen, muss zunächst die entsprechende Aktion auf der linken Seite des Dialogs markiert werden. Dann kann durch Betätigen der Schaltfläche, dessen Pfeil nach rechts zeigt, die Aktion hinzugefügt werden. Um eine Aktion wieder zu löschen, muss sie auf der rechten Seite markiert werden und die Schaltfläche, dessen Pfeil nach links zeigt, betätigt werden.

Die folgenden Aktionen stehen zur Verfügung:

- Benutzerverwaltung [▶ 426]
- Dialog schließen [▶ 428]
- Dialog öffnen [▶ 428]
- Sprachumschaltung [▶ 430]
- Visualisierungswechsel [▶ 430]
- Internes Kommando [▶ 431]
- Framevisualisierung umschalten [▶ 433]
- Variable schreiben [▶ 436]
- ST-Code ausführen [▶ 438]
- Variable umschalten [▶ 439]

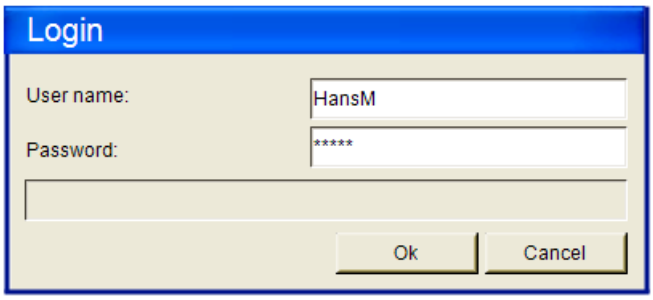
Benutzerverwaltung

Die Benutzerverwaltung als Nachfolgeaktion kann nur gewählt werden, wenn zuvor eine Benutzerverwaltung [▶ 412] angelegt oder die Bibliothek "VisuUserManagement" manuell hinzugefügt worden ist.

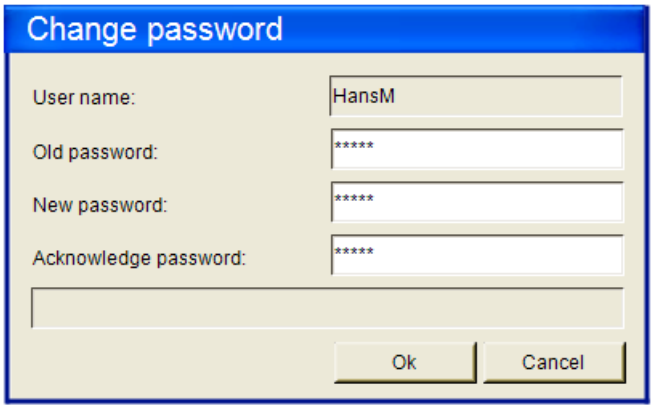


Über die Benutzerverwaltung können die vier folgenden Standarddialoge und -aktionen hinzugefügt werden. Mit ihrer Hilfe ist es dem Visualisierungsnutzer im Onlinebetrieb möglich:

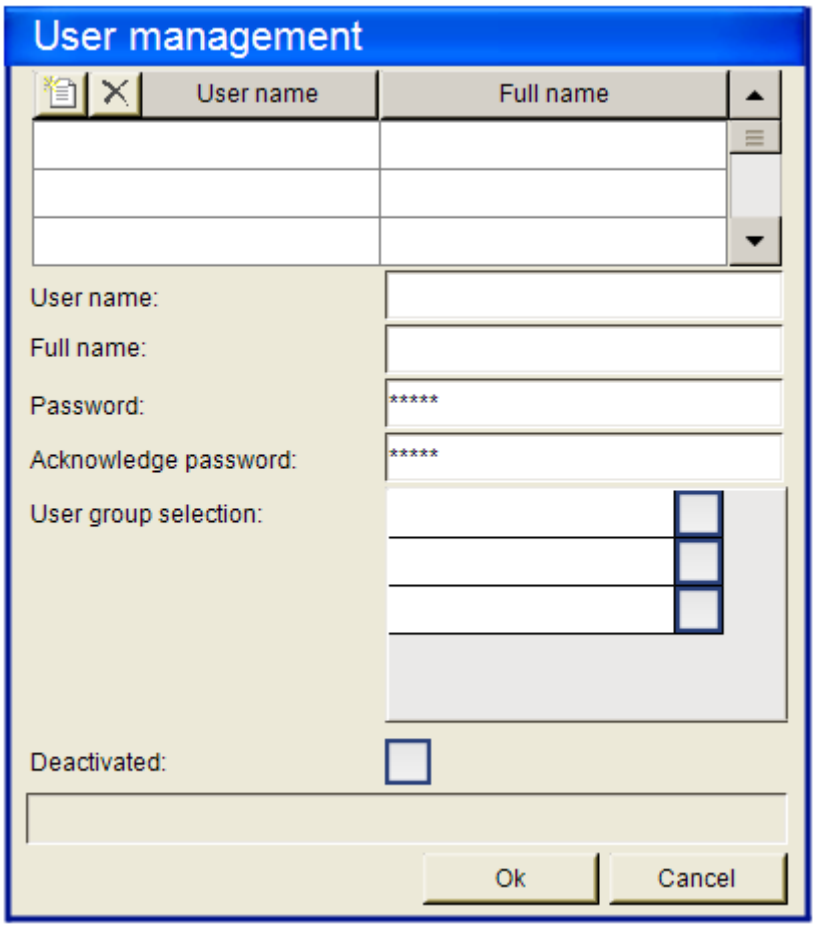
- sich anzumelden



- sich abzumelden
- das Benutzerpasswort zu ändern

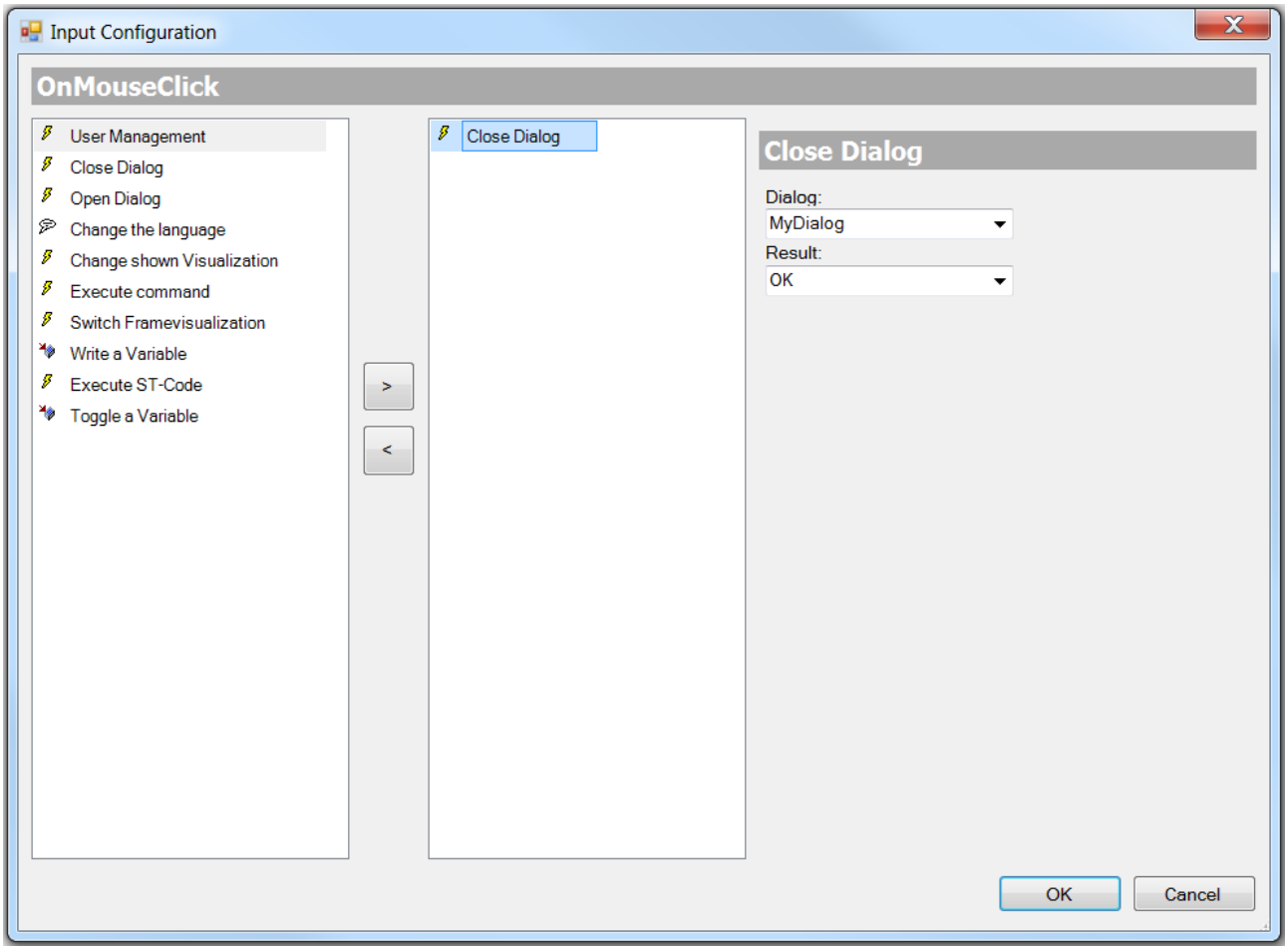


- die Benutzerverwaltung zu öffnen



Dialog schließen

Hier kann festgelegt werden, dass auf das Ereignis hin der angegebene Dialog mit dem angegebenen Resultat geschlossen wird. Wählen Sie den gewünschten Dialog von der Auswahlliste, die alle momentan verfügbaren Eingabedialoge anbietet.



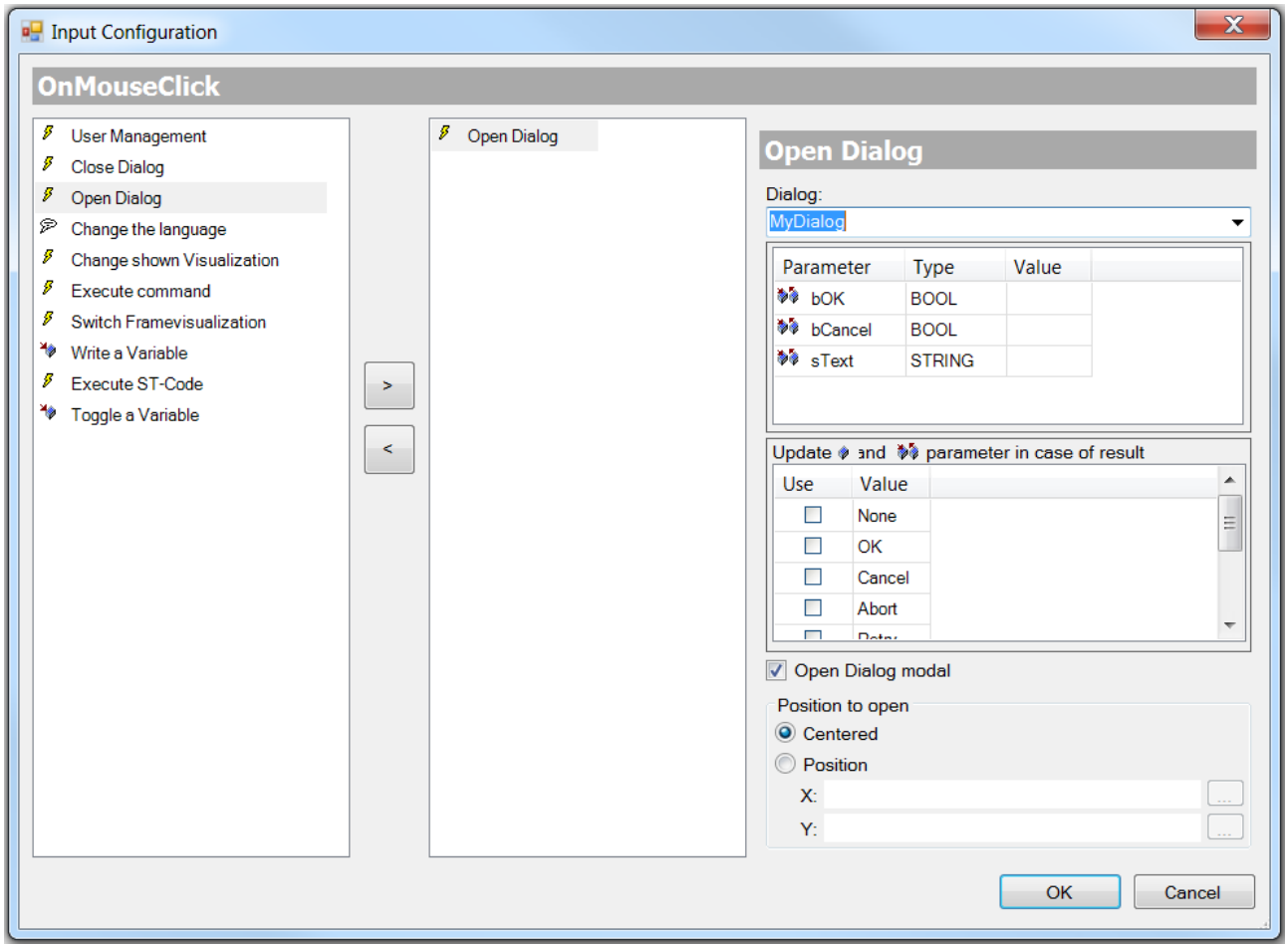
Dialog	Auswahl eines Dialogs aus der Auswahlliste
Ergebnis	Die Liste bietet die Standardoptionen an, die in Dialogen verwendet werden. In diesen Standardoptionen ist eine Benutzerreaktion gefordert. • OK • Cancel • Abort • Retry • Ignore • Yes • No

Beachten Sie, dass das Resultat, das vom zuletzt geschlossenen Dialog stammt, abgerufen werden kann und dass eine entsprechende Reaktion darauf in einem beliebigen Element derselben Visualisierung konfiguriert werden kann. Verwenden Sie dazu die Konfigurationsoption "OnDialogClosed".

Dialog öffnen

Hier kann definiert werden, dass auf eine Mausaktion hin ein Dialog geöffnet werden soll, der durch eine andere (Standard- oder vom Benutzer erstellte) Visualisierung repräsentiert wird. Die Auswahlliste bietet alle Visualisierungen an, die in Ihren Objekteigenschaften [▶ 422] den Verwendungszweck "Dialog" eingetragen haben. Somit kann ein selbst erstellter Dialog als Benutzereingabemaske in einer Visualisierung verwendet werden.

Zumindest die Standardobjekte "VisuDialogs.Login" und "VisuDialogs.FileOpenSave" sind verfügbar, wenn VisuDialogs.library im Projekt eingebunden ist. Es können zudem auch selbst erstellte Eingabedialoge genutzt werden.

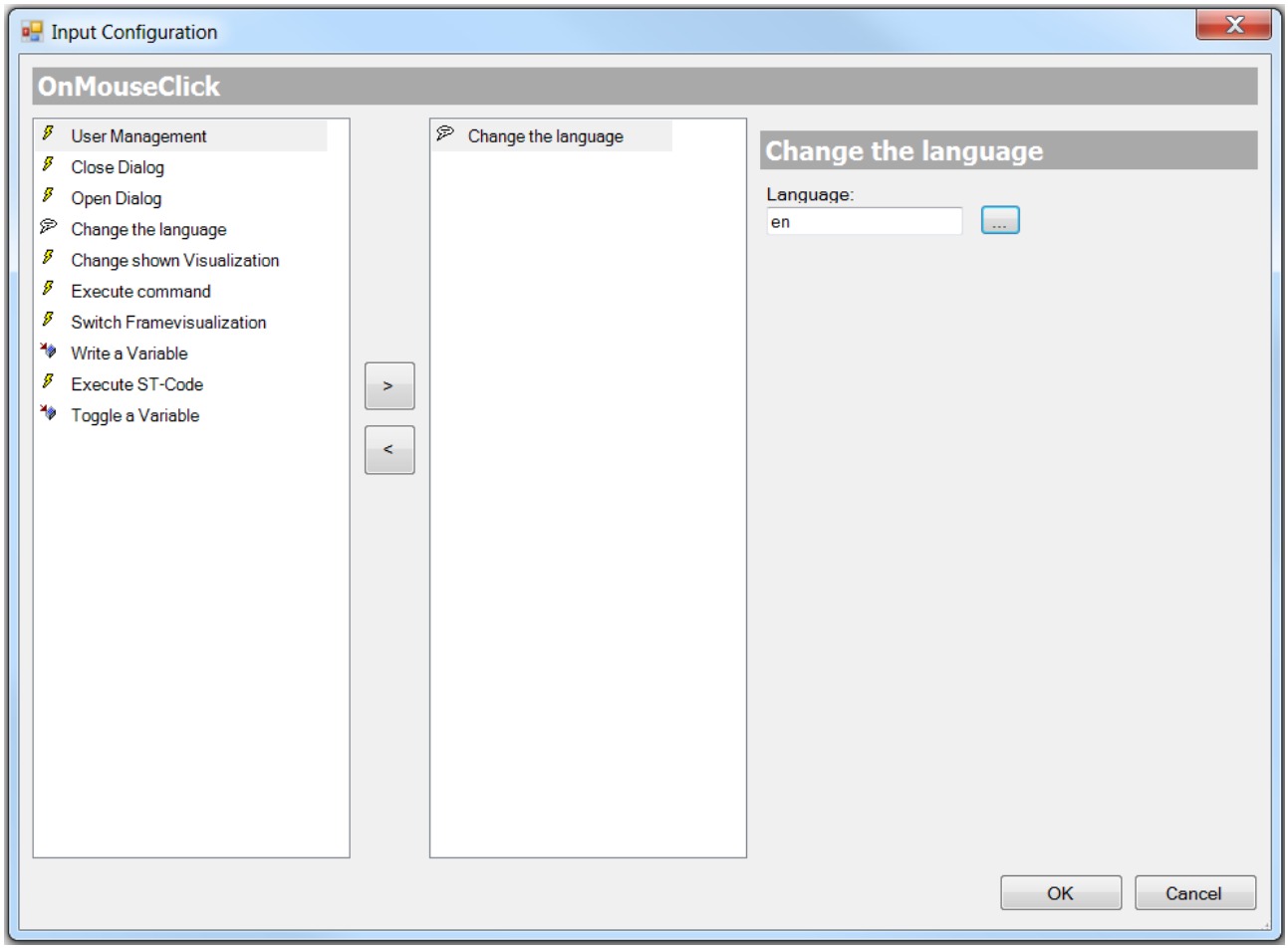


Dialog	Auswahl eines Dialogs aus der Auswahlliste
Parameter	Für die ausgewählte Dialogvisualisierung werden die Eingangs- (VAR_INPUT), Ausgangs- (VAR_OUTPUT) und Ein-/ Ausgangsparameter (VAR_IN_OUT,) die im Schnittstellen-Editor [▶ 395] definiert worden sind, dargestellt. (Parameter, Typ, Wert) In der letzten Spalte "Wert" können die Parameter mit Variablen aus der SPS verbunden werden. Die Werte der Variablen werden dann beim Öffnen der Dialogvisualisierung in die Parameter geschrieben (VAR_INPUT, VAR_IN_OUT) und gegebenenfalls beim Schließen des Dialogs zurückgelesen. (VAR_OUTPUT, VAR_IN_OUT.
Ereignis	Es muss angegeben werden, auf welches Ergebnis der Dialogvisualisierung hin Ihre Ausgangs- und Ein-/Ausgangsparameter geschrieben werden sollen: Setzen Sie einen Haken in der Verwenden-Spalte des gewünschten Ergebniswertes. Mögliche Werte: • Keinen • OK • Abbrechen • Komplett abbrechen • Erneut versuchen • Ignorieren • Ja • Nein

Beachten Sie, dass die Ausgangs- und Ein-/Ausgangsparameter einer Dialog-Visualisierung nicht geschrieben werden, bevor der Dialog geschlossen wird! Bis dahin werden die Werte nur auf dem Stack gespeichert, d.h. werden nicht als Referenzen sondern als Kopien gehandhabt.

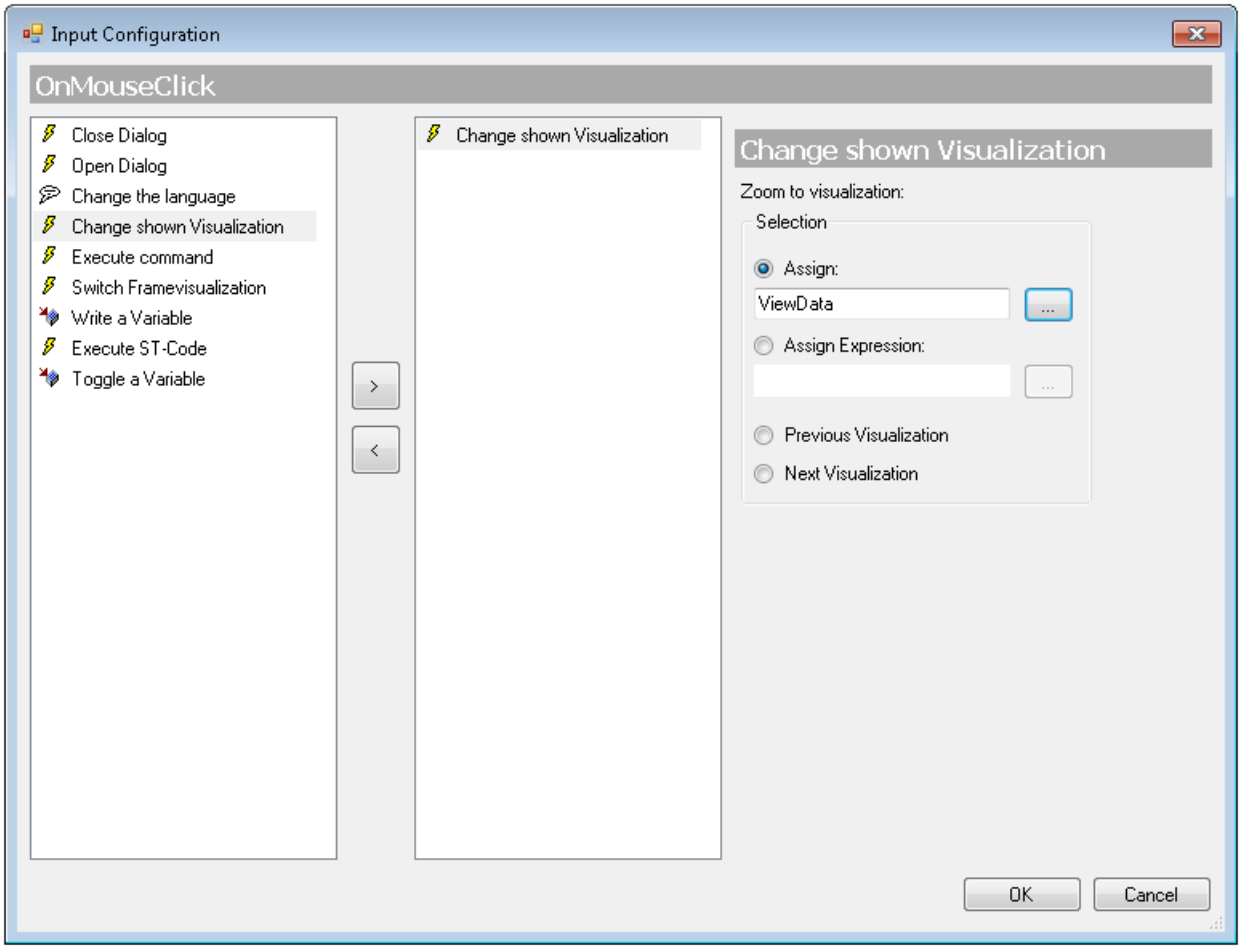
Sprachumschaltung

Hier kann eine Landessprache angegeben werden, die auf die Mausaktion hin für die Darstellung der Visualisierungstexte verwendet wird. Verwenden Sie dabei das Sprachkürzel, das in der/ den zugehörigen Textliste(n) [▶ 145] verwendet wird. Alternativ können Sie auf die Taste [...] klicken, um die gewünschte Sprache im Eingabeassistenten auszuwählen. (Siehe auch Sprache und Text [▶ 646])



Visualisierungswechsel

Hier kann ein Wechsel der aktuell sichtbaren Visualisierungsseite als Nachfolgeaktion eingestellt werden. (Siehe auch Abschnitt "Umschalten zwischen Visualisierungsseiten")

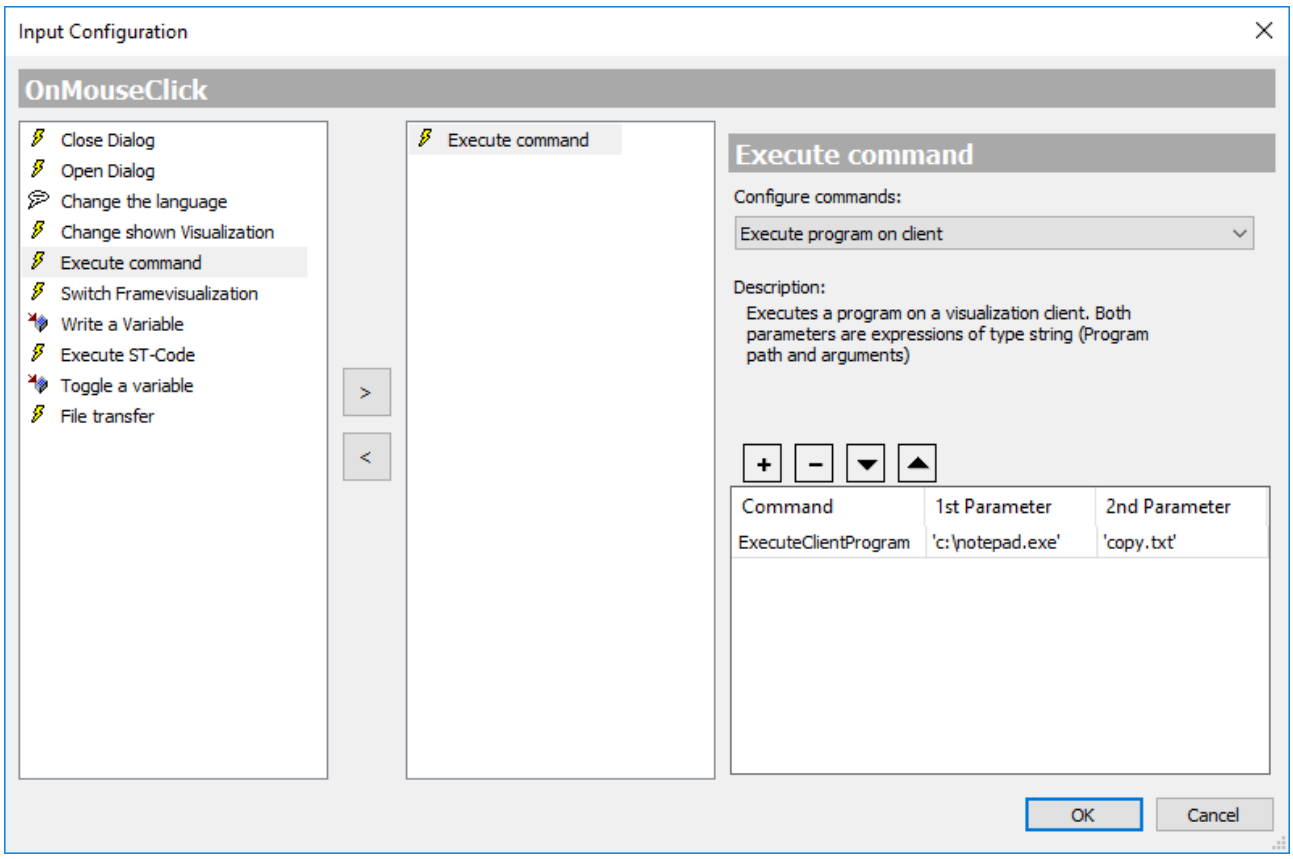


Wählen Sie eine der folgenden Optionen zur Festlegung, welche Visualisierungsseite auf die Mausaktion im Onlinebetrieb hin angezeigt werden soll:

Zuweisen	Die Visualisierungsseite kann direkt eingegeben werden. Dazu wird idealerweise die Eingabehilfe verwendet, die über die Schaltfläche [...] geöffnet werden kann.
Ausdruck zuweisen	Hier kann eine Variable vom Typ String angegeben werden, die von dem SPS-Projekt verwendet wird und den Namen der Visualisierungsseite liefert. Beispiel: sVisualizationName : STRING := ‘MyVisualization‘;
	Die Abfolge, in der Visualisierungsseiten nacheinander über Benutzereingaben angezeigt wurden, wird intern gespeichert. Mit den folgenden beiden Optionen können diese Informationen genutzt werden.
Vorhergehende Visualisierung	Die zuvor angezeigte Visualisierungsseite wird wieder angezeigt. Wenn zuvor noch keine aufgerufen worden ist, wird weiterhin die aktuelle gezeigt.
Nächste Visualisierung	Die Visualisierungsseite, die in der aufgezeichneten Abfolge der Visualisierungswechsel hinter der aktuellen steht, wird angezeigt. Dies ist als nur möglich, wenn aus dieser vorher bereits ein Visualisierungswechsel über "Vorhergehende Visualisierung" vorgenommen worden ist.

Internes Kommando

Hier können ein oder mehrere Kommandos definiert werden, die auf die Mausaktion hin ausgeführt werden sollen.



Wählen Sie aus der Liste "Befehle konfigurieren" einen Befehl aus und betätigen Sie die Schaltfläche  , um ihn in der Tabelle im unteren Teil des Dialogs einzufügen. Diese Tabelle enthält alle ausgewählten Befehle der Eingabekonfiguration.

Eine kurze Beschreibung zum oben in der Liste ausgewählten Befehl wird immer im Mittelteil des Dialogs angezeigt. Ergänze Sie dementsprechend die Parameter des Befehls unten in den Tabellenspalten "1. Parameter" und "2. Parameter". Die Spalte "Kommando" zeigt den internen Kommandonamen. Sehen Sie hierzu auch in der folgenden Tabelle die Beschreibung der einzelnen Kommandos.

Mit der Schaltfläche  kann der gerade ausgewählte Eintrag von der Liste entfernt werden. Die konfigurierten Befehle werden später, wenn die Benutzereingabe auf das Visualisierungselement erfolgt, entsprechend ihrer Anordnung in der Tabelle von nach unten nacheinander aufgeführt. Um die Reihenfolge zu verändern, können die Einträge mit den Schaltflächen  und  verschoben werden.

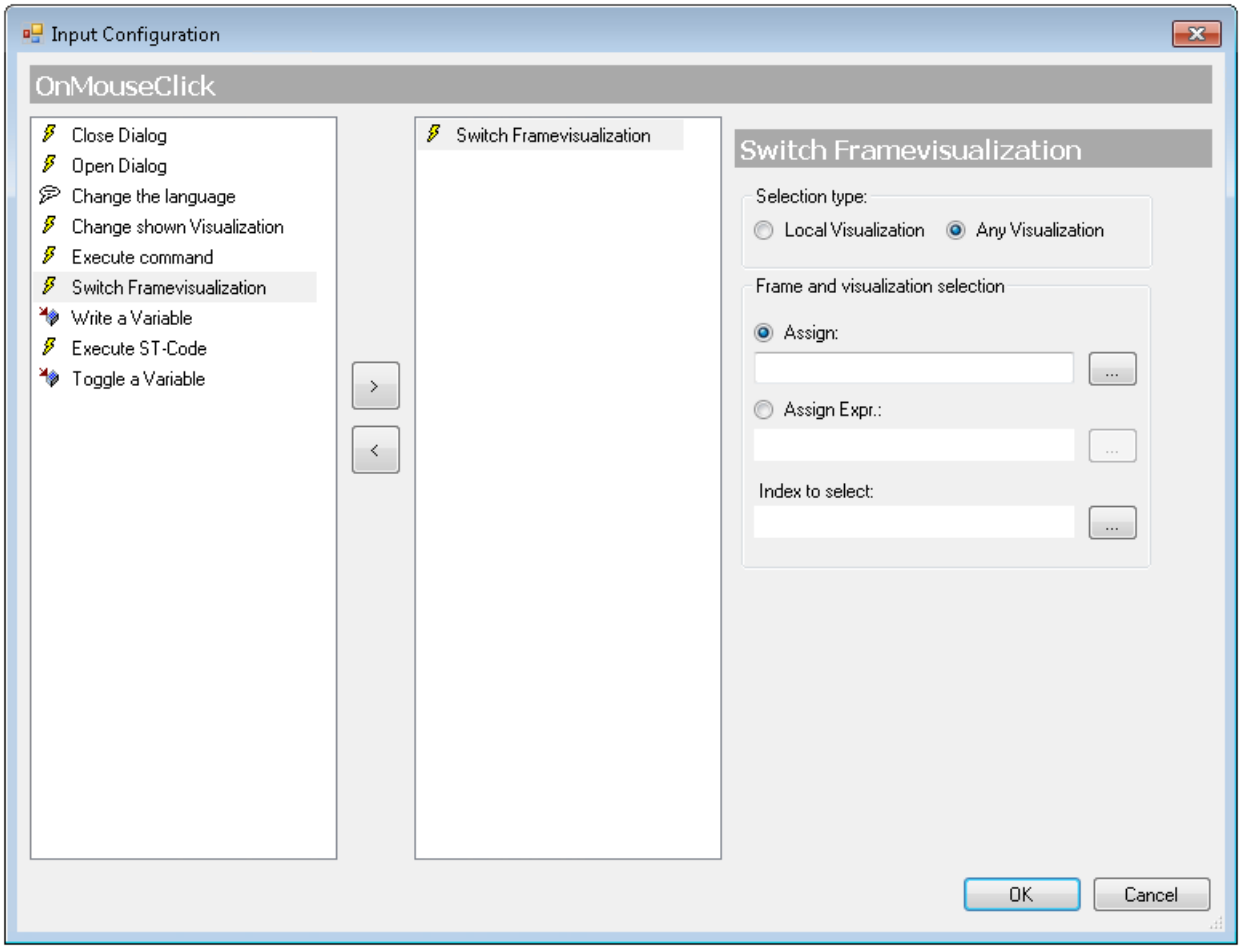
Die folgenden Kommandos stehen zur Auswahl:

Programm auf der Steuerung ausführen Programm beim Client ausführen	Das angegebene Programm (*.exe) wird auf der Steuerung bzw. beim Visualisierungs-Client ausgeführt. 1. Parameter: Zeichenfolge (String), Pfad der Programmdatei (Beispiel: c:\programs\notepad.exe) 2. Parameter: Zeichenfolge (String), Argumente für das auszuführende Programm, z.B. Name einer Datei, die das Programm öffnen soll (Beispiel: copyfile.txt)
Drucken	Der Standarddialog "Drucken" wird geöffnet, wo Einstellungen zum Einstellen des Druckerbereichs und der Druckerparameter etc. vorgenommen werden können. Darüber kann die aktuelle Visualisierung ausgedruckt werden. Dieser Befehl wird nur innerhalb der PLC HMI unter Windows unterstützt.
Zu URL (WebVisu) navigieren	Voraussetzung: Visualisierung wird als PLC HMI Web ausgeführt. Bei Eintreten des Eingabeereignisses navigiert die Visualisierung zur Webseite er angegebenen URL 1. Parameter: Webadresse URL - als Variable des Typs String, um die Startseite programmatisch zu setzten. (Beispiel: MAIN.sUrl) - als Literal in einfachen Hochkommata. (Beispiel: http://www.beckhoff.com 2. Parameter: Wenn kein Parameter angegeben ist, wird die Webseite in einem neuen Fenster oder einer neuen Registerkarte dargestellt. Wenn "replace" angegeben ist, wird die PLC HMI Web durch die Webseite ersetzt.

Framevisualisierung umschalten

Voraussetzung: In einer Visualisierung im Projekt gibt es mindestens ein Frame-Element, dem über die Frame-Auswahl [▶ 555] verschiedene Visualisierungen zugeordnet wurden. Diese Visualisierungen erhalten innerhalb des Frames eine Indizierung 0, 1, 2, usw. und im Standardfall wird die erste der zugeordneten Visualisierungen (Index 0) online im Frame angezeigt.

Im vorliegenden Dialog können Sie nun konfigurieren, dass beim Ausführen der Mausaktion auf das aktuelle Visualisierungselement in einem bestimmten Frame-Element eine bestimmte der ihm zugeordnete Visualisierung angezeigt wird. Mit dem aktuellen Element kann also ein Umschalter für die Anzeige von Visualisierung in einem Frame programmiert werden. (Siehe auch Abschnitt "Umschalten zwischen Visualisierungsseiten innerhalb eines Frames [▶ 644]")



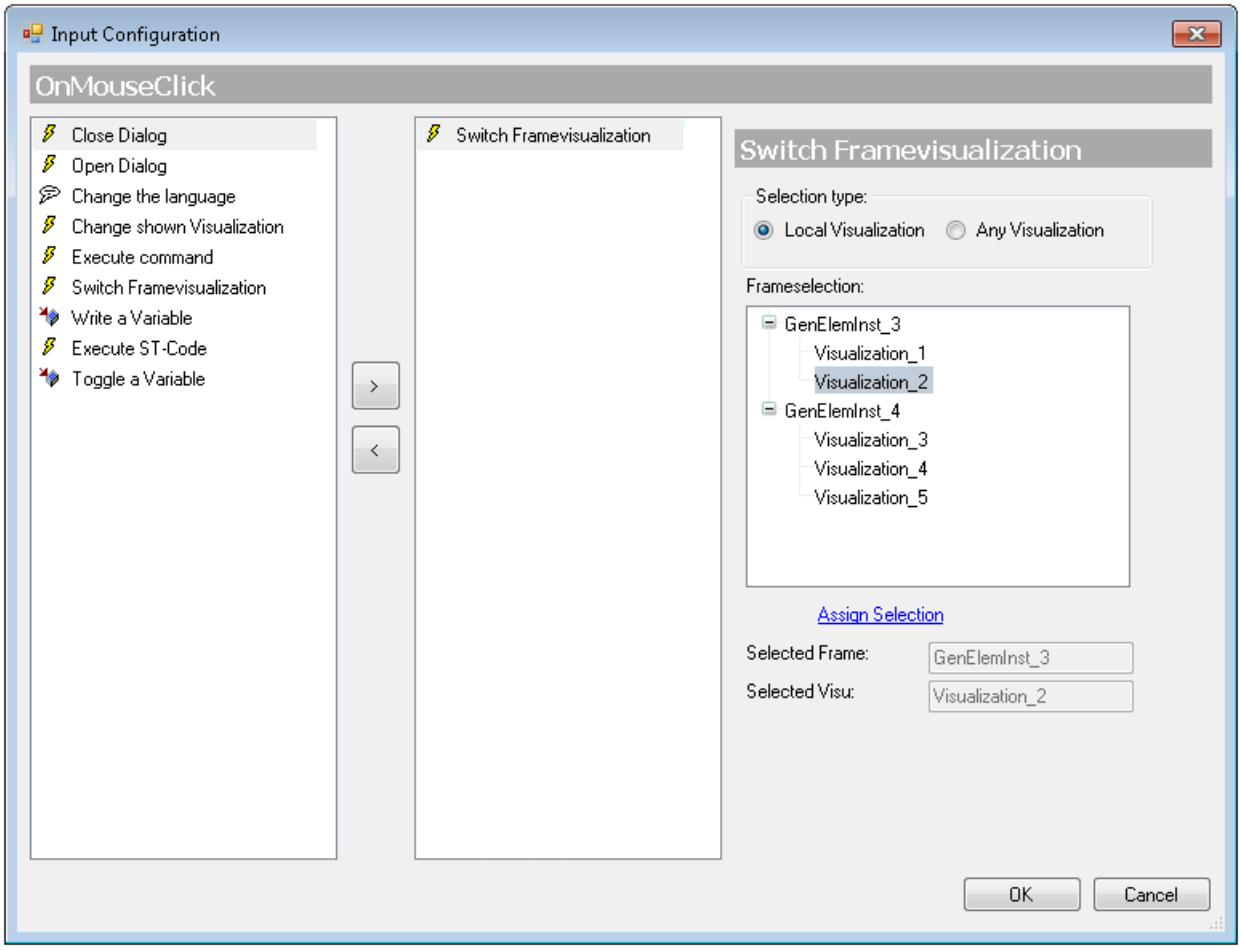
Die Auswahlart des betroffenen Frame-Elementes

- Kann sich auf die aktuelle Visualisierung beschränken, was einen einfacheren Konfigurationsdialog, allerdings ohne dynamische Konfigurationsmöglichkeit, bedeutet. (→ Lokale Visualisierung)
- Kann für alle im Projekt verfügbaren Visualisierungen erweitert werden, die auch dynamische Definitionen des Frames und der angezeigten Visualisierung erlauben (→ beliebige Visualisierung)

Lokale Visualisierung

Nur Frames der aktuellen Visualisierung können angesprochen werden und stehen somit hier im Feld "Auswahl des Frames" zur Auswahl. Dies ist ein einfacher Dialog zur schnellen direkten Konfiguration in der lokalen Visualisierung. Er bietet allerdings nicht die Möglichkeit, die gewünschte Visualisierung über eine Projektvariable anzugeben. Wenn dies benötigt wird, wählen Sie "beliebige Visualisierung".

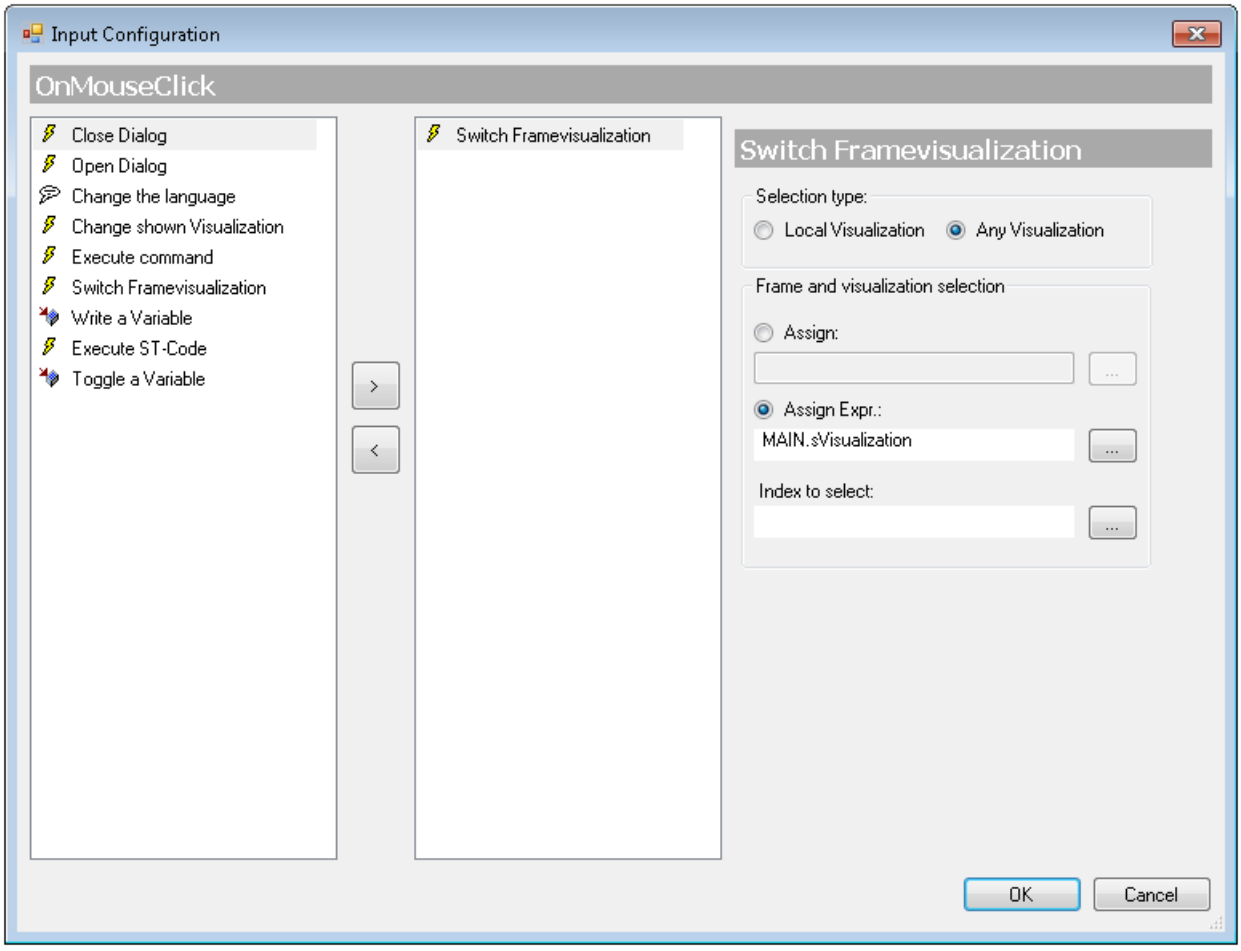
Die Auswahl des Frame zeigt die lokalen Frame-Elemente und eingerückt darunter jeweils die zugeordneten Visualisierungen.



Wählen Sie unterhalb des betroffenen Frames die gewünschte Visualisierung, die auf die Mausaktion hin angezeigt werden soll. Klicken Sie auf "Auswahl zuweisen", um die Einstellungen zu speichern. Die aktuelle Auswahl wird daraufhin in den Feldern Frameauswahl und Visualisierungsauswahl angezeigt.

Beliebige Visualisierung

Alle Frames des Projekts und ihre zugeordneten Visualisierungen stehen zur Auswahl. In diesem Fall können Frame und Visualisierung auch über Projektvariablen, also dynamisch angegeben werden.



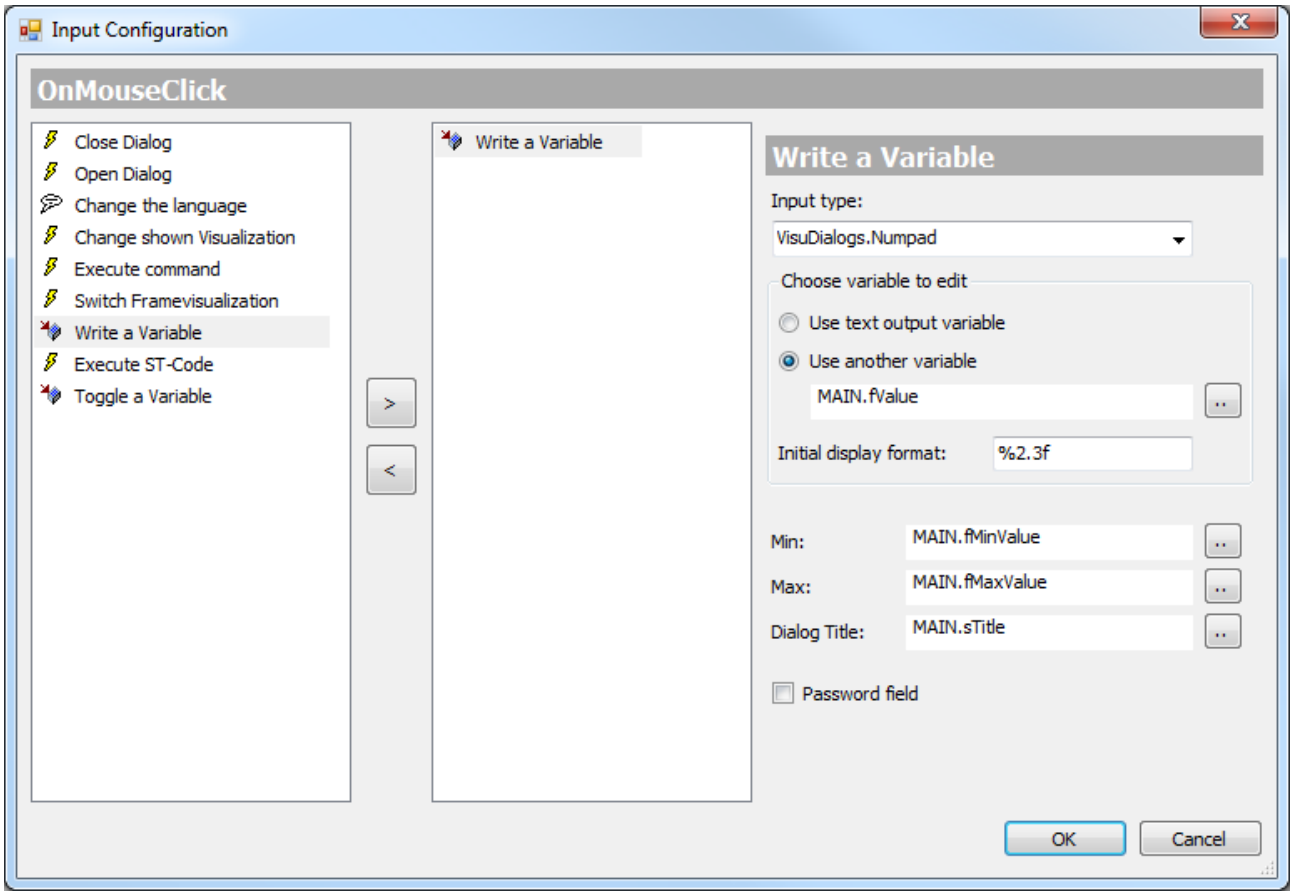
Die "Auswahl von Frame und Visualisierung" wird dann über folgende Felder und Optionen vorgenommen:

Direkte Zuweisung	Wenn diese Option aktiviert ist, kann der Name des betroffenen Frame-Elements direkt mit der entsprechenden Visualisierung eingegeben werden. Über die Schaltfläche [...] kann die Eingabehilfe dazu verwendet werden. Beispiel: Visualization_1.MyFrame
Zuweisung über Ausdruck	Diese Option kann alternativ aktiviert werden, um eine Projektvariable vom Typ String für die Angabe des Frame-Elements zu verwenden. Über die Schaltfläche [...] kann die Eingabehilfe geöffnet werden. Die Variable muss dann den Namen des Frame-Elementes und der entsprechende Visualisierungsnamen liefern. Beispiel: sVisualization : STRING := 'Visualization_1.MyFrame';
	Die gewünschte Visualisierung, die bei der Mausaktion im ausgewählten Frame angezeigt werden soll, wird über ihren Index angegeben. Dieser Index ist eine mit 0 beginnende aufsteigende, ganzzahlige Nummerierung der Visualisierung, die einem Frame in der Frame-Auswahl [▶ 555] zugewiesen wurden. Somit wird initial standardmäßig immer die erste Visualisierung in dieser Liste angezeigt.
Auszuwählender Index	Geben Sie hier den Index der gewünschten Visualisierung direkt oder über eine in der Applikation verwendete Projektvariable an. Die Schaltfläche [...] öffnet die Eingabehilfe zur Auswahl einer Variablen.

Variable schreiben

Wenn für ein Visualisierungselement eine Eingabekonfiguration des Typs "Variable schreiben" existiert, dann wird, sobald die entsprechende Mausaktion ausgeführt wird, das Element eine Möglichkeit bieten, einen Wert einzugeben. Der Wert kann dann als eine Zeichenfolge über ein Numpad oder Keypad eigegeben

werden und wird nach Beendigung der Eingabe auf die Projektvariable geschrieben, die hier im Eingabekonfigurationsdialog angegeben ist. Der Wert wird entsprechend des Datentyps der Projektvariablen als Text oder als numerischer Wert interpretiert.



Wählen Sie von der Auswahlliste im rechten Teil des Dialogs einen der folgenden Eingabetypen:

Standard	Es wird der Standard-Eingabetyp verwendet. Sie können den Standard in den Einstellungen des Visualisierungsmanagers festlegen.
Texteingabe	Es öffnet sich ein Feld für die Wert-/ Texteingabe. Um Werte einzugeben, muss eine Tastatur vorhanden sein.
Texteingabe mit Grenzen	Es öffnet sich ein Feld für die Wert-/ Texteingabe. Zusätzlich werden oberhalb die Eingabegrenzen angezeigt, welche bei Unterschreiten oder Überschreiten rot dargestellt werden. Um Werte einzugeben, muss eine Tastatur vorhanden sein.
VisuDialogs.Keypad	Die Mausaktion wird eine simulierte Tastatur öffnen. Sie können durch Mausklicks auf die entsprechenden Tastenfelder eine Zeichenfolge eingeben.
VisuDialogs.Numpad	Die Mausaktion wird ein simuliertes Nummernfeld öffnen. Sie könne durch Mausklicks auf die entsprechenden Tastenfelder eine numerische Zeichenfolge eingeben.
	Zusätzlich zu den Standard-Eingabetypen werden alle Visualisierungen angeboten, welche in ihren Eigenschaften als "Nummernfeld", "Tastatur" oder "Dialog für Eingabekonfiguration" definiert wurden. (Siehe Objekteigenschaften [▶ 422])

Für die Benutzung von Tastatur oder Nummernfeld muss die Bibliothek "VisuDialogs" im Bibliotheksmanager hinzugefügt werden.

Wählen Sie dann die zu bearbeitende Variable aus:

Textausgabevariable verwenden	Der Wert wird in die Variable geschrieben, die als Textausgangsvariable in dem Visualisierungselement angegeben ist.
Andere Variable verwenden	Angabe einer Projektvariablen. Hierzu können Sie über die Schaltfläche ... die Eingabehilfe öffnen.
Initiales Darstellungsformat	Angabe eines Platzhalters mit Formatierungsangabe Beispiel: %2.3f
Min	Angabe der Untergrenze für die einzugebende Variable
Max	Angabe der Obergrenze für die einzugebende Variable
Dialogtitel	Angabe als Text oder über eine Textvariable, was in der Titelleiste des Dialogs angezeigt werden soll
Passwortfeld	Falls Sie wünschen, dass Ihre Eingabe durch "*********" (wie im Fall von Passworten) versteckt dargestellt werden soll, so aktivieren Sie das Kontrollkästchen vor dem Passwortfeld.

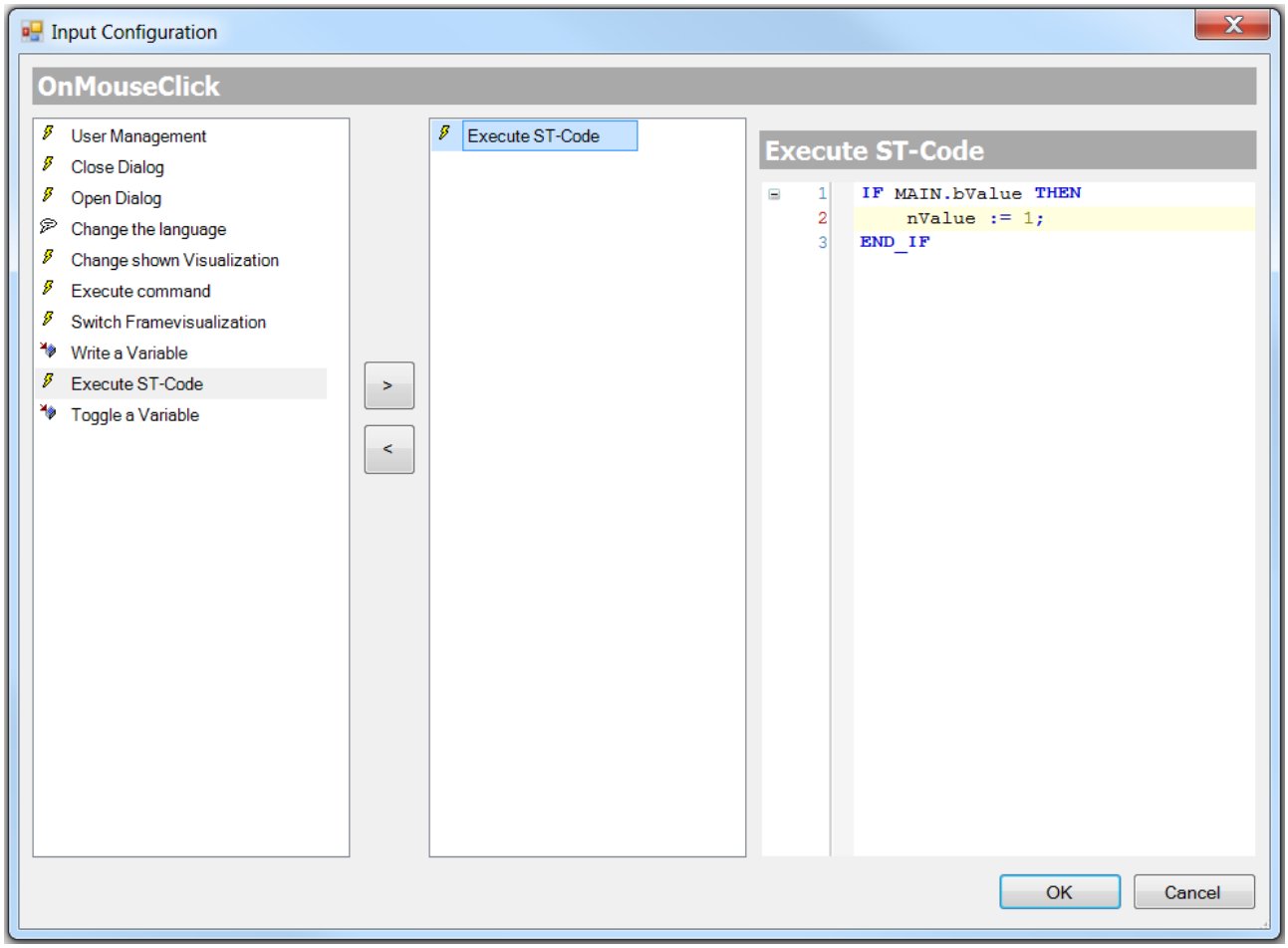
Beispiel

Sehen Sie als Beispiel die obige Dialogabbildung und nehmen Sie an, es handele sich um den Konfigurationsdialog für ein Rechteckelement. Wenn im Onlinebetrieb die Maustaste auf diesem Rechteck gedrückt wird, erscheint ein Nummerntastaturfeld mit dem Titel, der von der Variablen "MAIN.sTitle" geliefert wird.

Durch Mausklick auf 1, 2 und 3 kann beispielsweise der Wert "123" eingegeben werden. Dieser Wert wird im oberen Teil des Dialogs angezeigt, ebenso wie die Minimal- und Maximalwerte für die Eingabe, die durch "MAIN.fMinValue" und "MAIN.fMaxValue" gegeben werden. Sobald die Eingabe mit einem Mausklick auf die Taste OK bestätigt wird, wird "123" auf die Variable "MAIN.fValue" geschrieben. Wenn "fValue" als STRING-Variable definiert ist, erhält es den Wert "123". Wenn "fValue" eine numerische Variable ist, erhält sie den Wert 123.

ST-Code ausführen

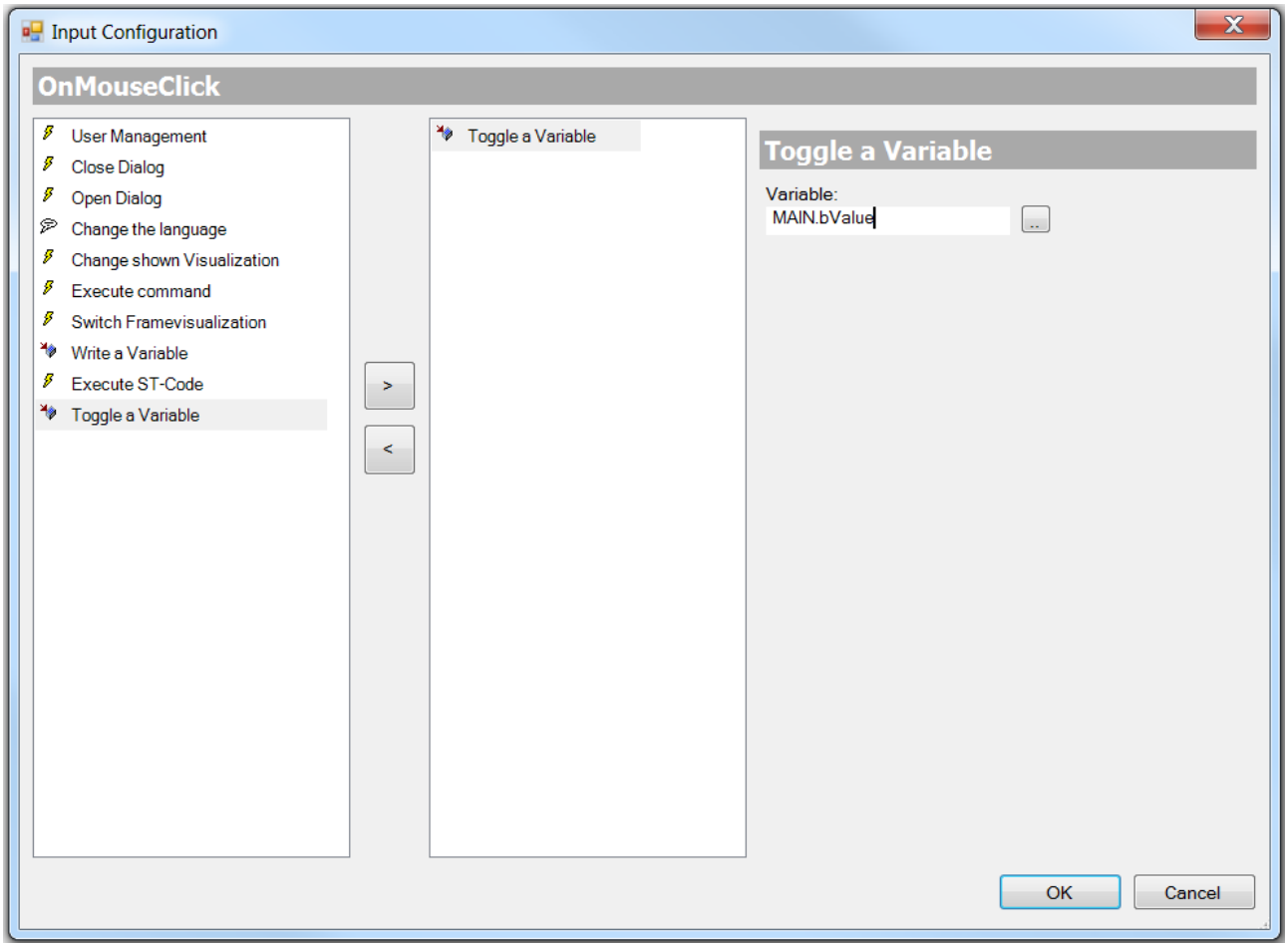
Im Eingabefeld dieser Nachfolgeaktion kann Code in Strukturiertem Text eingegeben werden, der auf eine Mausaktion hin ausgeführt werden soll.



Ein komplexes ST-Programm ist nur möglich, wenn die PLC HMI [▶ 635] und/oder die PLC HMI Web [▶ 640] aktiviert worden ist. Ansonsten sind nur einfache Zuweisungen ausführbar.

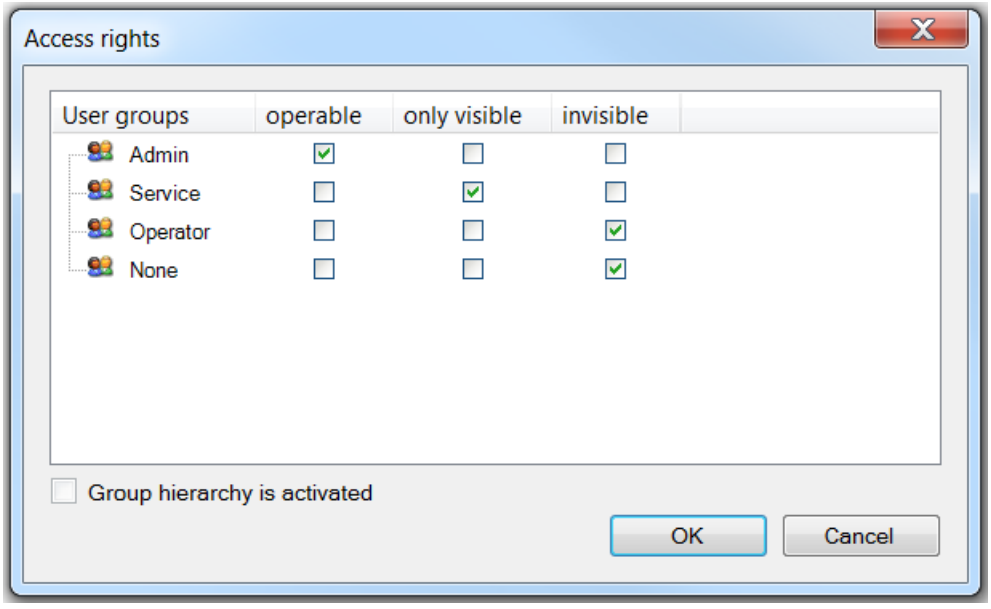
Variable umschalten

Hier kann eine boolesche Variable eingegeben werden, die bei wiederholter Mausaktion zwischen TRUE und FALSE wechseln soll.



15.8.1.3 Zugriffsrechtedialog

Ein Mausklick in das Wert-Feld der Zugriffsrechte in den Eigenschaften [▶ 403] eines Elements öffnet den folgenden Dialog:



Benutzergruppen	Die Gruppen, die im Visualization Manager unter Benutzermanagement → Gruppen konfiguriert wurden, sind hier aufgelistet. Markieren Sie eine Checkbox in der Zeile der Gruppe, um an diese Zugriffsrechte zu vergeben.
Funktionsbereit	Ist die Checkbox markiert, bietet das Element volle Funktionalität.
Nur sichtbar	Das Element bietet keine Funktionalität, ist aber sichtbar
Unsichtbar	Das Element wird für die entsprechenden Benutzergruppen nicht angezeigt.
Gruppenhierarchie wird verwendet	Ist im Visualization Manager unter Benutzermanagement → Einstellungen die Gruppenhierarchie auf "Verwenden" gesetzt, dann wird dieses hier angezeigt und die deaktivierte Checkbox markiert.

Standardgruppe "None"

"None" ist die Standardgruppe und kann nicht gelöscht werden. Ist kein Benutzer eingeloggt, dann verhalten sich die Visualisierungselemente entsprechend der Konfiguration von "None".

Werden die Zugriffsrechte eines Element beschränkt, ist es empfehlenswert, "None" mit den geringsten Rechten zu belegen.

15.8.2 Allgemeine Steuerelemente

15.8.2.1 Beschriftung

Das Element Beschriftung dient dazu einer Visualisierung Beschriftungen, Überschriften und jede andere Art von Text hinzuzufügen.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche … die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Boolesche Variable. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.2 Combobox Integer

Die Combobox Integer erlaubt dem Benutzer, einen Wert aus einem Drop-down-Menü auszuwählen. Der Index des selektierten Wertes wird in eine Variable geschrieben. Die Einträge können aus einer Textliste oder als Bilderliste aus der Bildersammlung herangezogen werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Variable vom Typ Integer, deren Wert der Index des ausgewählten Listenelements ist.
Textliste	Name (String) der Textliste, die alle Einträge des Drop-down-Menüs enthält und indiziert.
Bildersammlung	Name (String) der Bildersammlung, die alle Bilder des Drop-down-Menüs enthält und indiziert.
Maximaler Wert	Maximale Anzahl an Einträgen im Drop-down-Menü

Texte

Tooltip	Hier kann der Text eingestellt werden, der als Tooltip des Elements verwendet werden soll. Er erscheint in der Visualisierung nur zur Laufzeit.

Texteigenschaften

Verwendung von	Hier kann zwischen den folgenden beiden Einstellungen gewählt werden: • Standard Stilwerten • Benutzerdefinierte Einstellungen ◦ Horizontale Ausrichtung ◦ Vertikale Ausrichtung ◦ Schriftart ◦ Farbe Schriftart

Benutzerdefinierte Texteigenschaften

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Unterbereich

Unterbereich verwenden	Wenn diese Option aktiviert ist, wird nur ein Unterbereich der zugewiesenen Listeneinträge verwendet.
Indexstart	Variable vom Typ Integer, die den Eintrag indiziert, der als erstes zu verwenden ist
Indexende	Variable vom Typ Integer, die den Eintrag indiziert, der als letztes zu verwenden ist
Filter missing textentries	Wenn diese Option aktiviert ist, werden nur die Einträge der Textliste mit vorhandenem Text angezeigt.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

15.8.2.3 Combobox Array

Die Combobox Array erlaubt dem Benutzer, eine Zeile mit Werten aus einem Drop-down-Menü auszuwählen. Der Index der selektieren Zeile wird in eine Variable geschrieben.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Variable vom Typ Integer, deren Wert der Index der ausgewählten Zeile ist.
Datenarray	Variable vom Typ Array, welche dargestellt werden soll. Die Struktur der Variablen bestimmt die Anzahl der Spalten und Zeilen der Tabelle.

Spalten

Das Element Combobox Array visualisiert eine Array- oder Strukturvariable in Tabellenansicht. Der Index von Arrayelementen bzw. von Komponenten einer Struktur wird in einer Spalte bzw. einer Zeile angezeigt. Zweidimensionale Arrays oder Arrays einer Struktur werden in mehreren Spalten angezeigt. Die visualisierte Variable wird in der Eigenschaft ‚Datenarray' festgelegt. Wenn dort eine Variable zugewiesen ist, kann die Darstellung der Tabellenspalten, in welchen die jeweiligen Arrayelemente dargestellt werden, festgelegt werden. Für jede Spalte, die einem Index [<n>] zugeordnet ist, ist eine individuelle Konfiguration möglich.

Spalten • [<n>]	Aufgrund der Struktur der Variablen, die unter "Datenarray" definiert ist, wird die Anzahl der Spalten ermittelt und über den Index <n> beschrieben. Beispiel: `aStringTable : ARRAY [0..2, 0..4] OF STRING := [,BMW', ,Audi', ,Mercedes', ,VW', ,Fiat', ,150', ,150', ,150', ,150', ,100', ,blau', ,grau', ,silber', ,blau', ,rot'` → Es werden drei Spalten gebildet [0], [1] und [2].
Max. Array-Index	numerische Variable, über die ein maximaler Array-Index festgelegt wird (optional)
Zeilenhöhe	Höhe der Zeilen in Pixel
Anzahl sichtbarer Zeilen	Hier kann festgelegt werden, wie viele Zeilen im Drop-down-Menü dargestellt werden sollen. Falls die Anzahl der Zeilen kleiner ist als die Anzahl der Zeilen des Arrays, wird eine Scrollbar angezeigt.
Größe der Scrollbar	Breite der vertikalen Scrollbar in Pixel

Spalte [<n>]

Breite	Breite der Spalte in Pixel
Bildspalte	Wenn diese Option aktiviert ist, werden in dieser Spalte Bilder aus der globalen Bildersammlung oder von benutzerdefinierten Bildersammlungen angezeigt. Dabei legen die Werte der Zellen eine Tabelle die ID des Bildes in der Bildersammlung fest.
Bildkonfiguration • Füllmodus • Transparenz • Transparenzfarbe	Hier können die folgenden Konfigurationen vorgenommen werden: • Füllmodus Zelle ausfüllen: Das Bild wird auf die Zellengröße angepasst ohne Berücksichtigung des Breiten-Höhen-Verhältnisses. Zentriert: Das Bild wird unter Beibehaltung der Proportionen innerhalb der Zelle zentriert. Selbst, wenn die Höhe bzw. die Breite einzeln angepasst wird, bleibt das Breiten-Höhen-Verhältnis bestehen. • Transparenz Wenn diese Option aktiviert ist, wird die unter Transparenzfarbe eingestellte Farbe transparent dargestellt. • Transparenzfarbe Hier kann eine Farbe aus einer Auswahlliste oder mithilfe des Farbauswahldialogs, welcher über die Schaltfläche … geöffnet werden kann, ausgewählt werden. Die Farbe wird transparent dargestellt, wenn die entsprechende Option Transparenz aktiviert ist.
Textausrichtung in Spalte	Hier kann die Ausrichtung der Spalte geändert werden: • Links • Zentriert • Rechts

Texte

Tooltip	Hier kann der Text eingestellt werden, der als Tooltip des Elements verwendet werden soll. Er erscheint in der Visualisierung nur zur Laufzeit.

Texteigenschaften

Verwendung von	Hier kann zwischen den folgenden beiden Einstellungen gewählt werden: • Standard Stilwerten • Benutzerdefinierte Einstellungen ◦ Horizontale Ausrichtung ◦ Vertikale Ausrichtung ◦ Schriftart ◦ Farbe Schriftart

Benutzerdefinierte Texteigenschaften

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche  öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn  geklickt wird.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

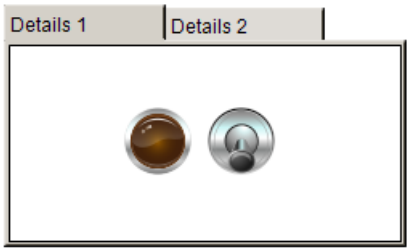
Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

15.8.2.4 Registersteuerelement

Ein Registersteuerelement ermöglicht es, mehrere Visualisierungsseiten innerhalb eines Fensters anzuzeigen. Es werden Registerkarten zum Umschalten zwischen den Visualisierungsseiten verwendet. Mit der Variable "Umschaltvariable [▶ 451]" kann die anzuzeigende Seite aber auch aus dem Programmcode festgelegt werden.



Wenn aufgrund der zu geringen Breite des Elements nicht alle Registerkarten angezeigt werden können, werden an dem Element oben rechts zwei Navigationspfeile eingeblendet.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Referenzierte Visualisierungen

Hier kann der Dialog "Frame-Auswahl [▶ 555]" geöffnet werden, mit dessen Hilfe die zu referenzierenden Visualisierungsseiten ausgewählt werden können. Nach der Auswahl einer oder mehrerer Visualisierungsseiten werden diese darunter gegebenenfalls mit ihren Platzhaltern [▶ 644] aufgelistet. Bei Änderungen der Platzhalter wird automatisch der Dialog "Aktualisierung der Frameparameter [▶ 555]" für alle Instanzen geöffnet.

Umschaltvariable

Mit dieser Eigenschaft können Visualisierungen eines Frames umgeschaltet werden.

Variable	Integer-Variable, deren Wert die ID der anzuzeigenden Visualisierung enthält. Die ID einer Visualisierung wird durch deren Reihenfolge in der Liste der zugewiesenen Visualisierungen in der Frame-Auswahl [▶ 649] bestimmt. Zum Beispiel ergibt der erste Eintrag in dieser Liste die ID 0 und der zweite Eintrag die ID 1. Die Visualisierungen, welche einem Frame zugewiesen sind, können mit einer Variablen umgeschaltet werden. Der Wert (ID) der Visualisierung wird durch die Reihenfolge dieses Elementes in der Liste der zugewiesenen Visualisierungen im Dialog "Konfiguration der Framevisualisierungen" bestimmt. Der erste Eintrag in dieser Liste ergibt für Ganzzahlwert den Wert 0, der zweite Eintrag den Wert 1 usw.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

Registerbreite	Breite der Reiter in Pixel
Skalierungsart	Hier ist zu spezifizieren, wie das Fenster auf Änderungen der Größe zu reagieren hat: • Isotropisch Das referenzierte Element behält seine Proportionen. Somit bleibt, auch wenn Höhe und Breite unabhängig geändert wurden, das Verhältnis von Höhe und Breite der Visualisierung unverändert. • Anisotropisch Das Registersteuerelement folgt der Größe, so dass Höhe und Breite der referenzierten Visualisierung unabhängig geändert werden können. • Unskaliert Die originale Größe der Visualisierung bleibt erhalten, unabhängig von der Größe des Registersteuerelements. • Unskaliert und scrollbar Verwenden Sie diese Option, dann wird die referenzierte Visualisierung ohne Skalierung angezeigt. Ist sie größer als der Fensterbereich des Frames, dann wird der Frame mit Scrollbalken versehen, um den angezeigten Bereich der Visualisierung verschieben zu können. Wenn Sie die Position des Scrollbalkens mit einer Variablen setzen wollen, verwenden Sie die Eigenschaften "Variable Scrollposition horizontal" und "Variable Scrollposition vertikal".
Deaktivieren des Hintergrundzeichens	Um die Performance der Visualisierung zu optimieren, werden die nicht-animierten Elemente des Frame-Elements als Hintergrund-Bitmap gezeichnet. Dies könnte dazu führen, dass die Elemente nicht in der erwarteten Reihenfolge dargestellt werden. Um dieses Verhalten zu vermeiden, kann die Funktion deaktiviert werden.

Scrollbalken-Einstellungen

Die Scrollbalken-Einstellungen sind nur sichtbar, wenn bei der Skalierungsart "Unskaliert und scrollbar" eingetragen ist. Es ist sehr empfehlenswert, die Variablen Client-spezifisch zu verwenden. Ändern sich die Variablen oder wird ein Scrollbalken mithilfe der Maus verschoben, dann wirkt sich die Änderung in diesem Fall nur auf den Frame des betroffenen Clients aus. Andernfalls werden alle Clients aktualisiert.

Variable Scrollposition horizontal	Variable, die die Position des horizontalen Scrollbalkens enthält.
Variable Scrollposition vertikal	Variable, die die Position des vertikalen Scrollbalkens enthält.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

15.8.2.5 Schaltfläche

Eine Schaltfläche ermöglicht dem Benutzer, eine dem Steuerelement zugeordnete Funktion auszulösen. Eine optische Anpassung ist durch Zuweisen eines Bildes [▶ 454] möglich oder auch durch Konfigurieren einer Reliefansicht [▶ 453].

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Farbverlauf verwenden	Standardmäßig ist die Checkbox deaktiviert. Falls sie aktiviert wird, wird das zugehörige Element mit Farbverlauf gezeichnet.
Farbverlaufsauswahl	Der Farbverlaufeditor [▶ 424] öffnet sich.

Schaltflächenhöhe	Höhe in Pixel, die die Reliefansicht des Schaltflächenelements bestimmt

Farbverlauf und Transparenz werden unter Windows CE nicht unterstützt.

Bild

Statische ID	Statisch der Identifikator der Bilddatei, mit der die Schaltfläche darzustellen ist. Die ID (String) ist in der Bildersammlung definiert. Der Name der Sammlung sollte dabei vorangestellt werden, um den Eintrag eindeutig zu machen. Nur bei Bilddateien, die im GlobalImagePool verwaltet werden, muss keine Sammlung angegeben werden, da diese Sammlung immer als erste durchsucht wird. Mit Klick auf die Schaltfläche [...] öffnet die Eingabehilfe und listet alle verfügbaren Bildersammlungen mit ihren Bildern auf.
Skalierungsart	Hier ist anzugeben, wie die Bilddatei auf Änderungen der Schaltflächengröße reagieren soll: • Isotropisch: Das Bild behält seine Proportionen; d.h. das Verhältnis Höhe-Breite bleibt erhalten, auch wenn Höhe und Breite der Schaltfläche separat verändert werden. • Anisotropisch: Das Bild behält seine Proportionen; d.h. das Verhältnis Höhe-Breite bleibt erhalten, auch wenn Höhe und Breite der Schaltfläche separat verändert werden. • Unskaliert: Das Bild behält die Originalgröße, auch wenn sich die Größe der Schaltfläche ändert.
Transparenz	Wenn diese Option aktiviert ist, wird die Bitmap mit der in "Transparenzfarbe" gesetzten Farbe transparent dargestellt.
Transparenzfarbe	Hier kann die Farbe eingestellt werden, die im Bild voll-transparent dargestellt werden soll. Voraussetzung: Transparenz ist aktiviert
Horizontale Ausrichtung	Hier ist die horizontalen Ausrichtung der Bitmap zu definieren: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Hier ist die vertikalen Ausrichtung der Bitmap zu definieren: • Oben • Zentriert • Unten

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche [...] öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn [...] geklickt wird.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.
Innere Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um seinen Rotationspunkt gedreht wird; positive Werte = mathematisch positiv = Uhrzeigersinn. Im Gegensatz zu "Rotation" (s.o.) rotiert das Element selbst. Der Rotationspunkt (Zentrum ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Relative Bewegung

Das Element kann relativ zu seiner fixen Position verschoben werden. Die obere-linke, und untere-rechte Kante des Elements werden um den durch eine Integer-Variable definierten Wert (Pixel) in X- bzw. Y-Richtung verschoben. Im Gegensatz zur absoluten Bewegung wird eine relative Position definiert, d.h. der Abstand zur ursprünglichen Position. Das Element kann dadurch in der Form verändert werden. Positive Werte verschieben die horizontalen Kanten nach unten bzw. die vertikalen nach rechts.

Bewegung links-oben • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in Y-Richtung verschoben wird.
Bewegung rechts-unten • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in Y-Richtung verschoben wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := ‘Arial’;)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart‘. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach "16#" dargestellt. Farben, dessen Definition mit "16#00" beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in der Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements im Alarmzustand bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in den Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Schaltflächenzustandsvariable

Boolescher Wert	Boolesche Variable, die die Anzeige der Schaltfläche als ‚gedrückt' steuert, wenn die Variable TRUE ist, und als "nicht gedrückt", wenn die Variable FALSE ist.

Bild-ID-Variable

Bild-ID	Projektvariable vom Typ String, die die Bild-ID enthält, mit der das Element darzustellen ist. Die ID ist in der Bildersammlung [▶ 153] definiert. Der Name der Sammlung sollte dabei vorangestellt werden, um den Eintrag eindeutig zu definieren. Nur bei Bilddateien, die im "GlobalImagePool" verwaltet werden, muss keine Sammlung angegeben werden, da diese Sammlung immer als erste durchsucht wird.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

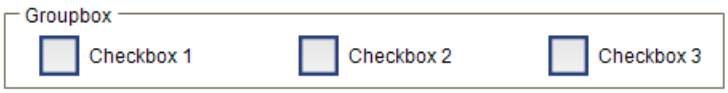
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.6 Groupbox

Visualisierungselemente können via Drap-and-drop Prinzip in eine Groupbox gezogen werden, um eine optische Einheit zu bilden. Sie können mehrmals verschachtelt werden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,

- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist.

Texteigenschaften

Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche [...] öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn [...] geklickt wird.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

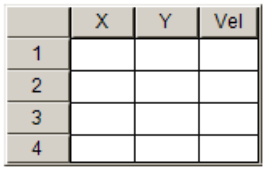
Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

15.8.2.7 Tabelle

Eine Tabelle kann in einer Visualisierung eingefügt werden, um ein- oder zweidimensionale Arrays, Strukturen oder lokale Variablen einer POU darzustellen. Ein Beispiel für die Konfiguration einer Tabelle finden Sie im Abschnitt "Konfiguration einer Tabelle [▶ 468]".



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Datenarray	Hier die zu visualisierende Variable mit vollständigem Pfad spezifizieren. Die Struktur der Variablen bestimmt die Anzahl der Spalten und Zeilen der Tabelle. Wenn die Anzahl der Array-Elemente der Variablen variiert, verwenden Sie "Max. Array-Index". Um die Tabelle zu aktualisieren, falls sich die Größe der Array- oder Strukturvariablen geändert hat, setzen Sie den Cursor in das Datenarray-Wertefeld und drücken die Eingabetaste. Beispiel: Das folgende Array bildet eine Tabelle mit drei Spalten und vier Zeilen. `aData : ARRAY [1..3,1..4] OF INT;`
Max. Array-Index	Numerische Variable, die dynamisch den maximalen Array-Index festlegt, um Überschreitungen der Array-Grenzen zu vermeiden.

Wird der Datenarray-Wert geändert, also eine neue Variable zugewiesen, werden die weiteren Tabelleneigenschaften auf ihren Standardwert zurückgesetzt.

Spalten

Das Tabellenelement visualisiert eine Variable in Tabellenansicht. Der Index des Arrays bzw. die Komponente der Struktur wird in einer Spalte bzw. Zeile dargestellt. Bei zweidimensionalen Arrays bzw. einem Array von Strukturen wird in mehreren Spalten visualisiert. In diesen Einstellungen des Elements wird das Aussehen der Tabellenspalten definiert, in welchen die Werte der einzelnen Arrayelemente/ Strukturkomponenten/ Variablen angezeigt werden. Für jede zu einem bestimmten Index gehörenden Spalte ist eine eigene Konfiguration möglich.

Spalten
• [<n>]
	Aufgrund der Struktur der Variablen, die unter ‚Datenarray' definiert ist, wird die Anzahl der Spalten ermittelt und über den Index <n> beschrieben.
	Beispiel:
	`aDataArray1 : ARRAY [2..8] OF INT;`
	→ Es wird eine Spalte [0] gebildet.
	`aDataArray2 : ARRAY [1..3, 1..10] OF INT;`
	→ Es werden drei Spalten gebildet [0], [1] und [2].

Zeilenkopf anzeigen	Wenn diese Option aktiviert ist, wird die Beschriftung der Zeilen in Form von zugeordneten Zeilenindizes angezeigt.
Spaltenkopf anzeigen	Wenn diese Option aktiviert ist, wird die Beschriftung der Spalten in Form von zugeordneten Zeilenindizes angezeigt.
Zeilenhöhe	Höhe der Zeilen in Pixel
Breite Zeilenkopf	Breite der Spalte, die den Zeilenkopf enthält, in Pixel
Größe der Scrollbar	Breite des vertikalen bzw. die Höhe des horizontalen Scrollbalkens in Pixel

Spalte [<n>]

Spaltenüberschrift	Hier kann die Spaltenbeschriftung geändert werden, indem ein neuer Titel eingetragen wird. Standardmäßig wird der Name des Arrays bzw. der Struktur mit dem zur Spalte gehörenden Index bzw. Strukturkomponente als Überschrift verwendet.
Breite	Breite der Spalte in Pixel
Bildspalte	Wenn diese Option aktiviert ist, werden in dieser Spalte Bilder aus der globalen Bildersammlung oder von benutzerdefinierten Bildersammlungen angezeigt. Dabei legen die Werte der Zellen eine Tabelle die ID des Bildes in der Bildersammlung fest.
Bildkonfiguration • Füllmodus • Transparenz • Transparenzfarbe	Hier können die folgenden Konfigurationen vorgenommen werden: • Füllmodus Zelle ausfüllen: Das Bild wird auf die Zellengröße angepasst ohne Berücksichtigung des Breiten-Höhen-Verhältnisses. Zentriert: Das Bild wird unter Beibehaltung der Proportionen innerhalb der Zelle zentriert. Selbst, wenn die Höhe bzw. die Breite einzeln angepasst wird, bleibt das Breiten-Höhen-Verhältnis bestehen. • Transparenz Wenn diese Option aktiviert ist, wird die unter Transparenzfarbe eingestellte Farbe transparent dargestellt. • Transparenzfarbe Hier kann eine Farbe aus einer Auswahlliste oder mithilfe des Farbauswahldialogs, welcher über die Schaltfläche  geöffnet werden kann, ausgewählt werden. Die Farbe wird transparent dargestellt, wenn die entsprechende Option Transparenz aktiviert ist.
Textausrichtung der Überschrift	Hier kann die Ausrichtung des Spaltenkopfs geändert werden: • Links • Zentriert • Rechts
Vorlage verwenden	Wenn diese Option aktiviert ist, wird in jeder Zelle dieser Tabellenspalte ein weiteres Visualisierungselement (vom Typ Rechteck, abgerundetes Rechteck oder Ellipse [▶ 499]) eingefügt. Die Eigenschaftenliste wird automatisch um die Eigenschaften dieses Elements unter Vorlage erweitert.
Textausrichtung der Überschrift aus der Vorlage	Wenn die Option aktiviert ist, werden die in der eingefügten Vorlage getroffenen Einstellungen für Schrift(-größe) und Ausrichtung auch für die Spaltenüberschrift angewendet. Die Aktivierung dieser Option hat nur bei gleichzeitiger Aktivierung von ‚Vorlage verwenden' eine Auswirkung.
Vorlage	Unter Vorlage werden sukzessive die Eigenschaften aller Elemente, die der Spalte zugewiesen wurden, aufgelistet und können dort, wie unter Rechteck, abgerundetes Rechteck oder Ellipse [▶ 499] beschrieben, editiert werden. Dieser Eintrag ist nur vorhanden, wenn "Vorlage verwenden" aktiviert ist.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := 'Arial';)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart'. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Selektion

Selektionsfarbe	Füllfarbe für selektierte Tabellenzellen
Selektionstyp	Hier ist zu definieren, wie bei einem Klick auf eine Tabellenzelle die Auswahl erfolgt: • Keine Auswahl • Zellenauswahl: Nur die angeklickte Zelle ist ausgewählt. • Zeilenauswahl: Die Zeile, in der sich die angeklickte Zelle befindet, wird selektiert. • Spaltenauswahl: Die Spalte, in der sich die angeklickte Zelle befindet, wird selektiert. • Zeilen- und Spaltenauswahl: Die Zeile und die Spalte, in der sich die angeklickte Zelle befindet, werden selektiert.
Rahmen um ausgewählte Zellen	Wenn diese Option aktiviert ist, wird um die selektierten Zellen ein Rahmen gezeichnet.
Variable für Spaltenauswahl	Variable vom Typ Integer, in der der Index der Spalte der selektierten Zelle gespeichert wird. Im Fall, dass das Datenarray auf eine Struktur zeigt, werden die Strukturkomponenten beginnend mit 0 indiziert. Zu beachten ist, dass dieser Index nur dann die korrekte Position im Array repräsentiert, wenn keine Spalten von der Anzeige in der Tabelle ausgenommen wurden.
Variable für Zeilenauswahl	Variable vom Typ Integer, in der der Index der Zeile der selektierten Zelle gespeichert wird
Variable für Gültigkeit der Spaltenauswahl	Boolesche Variable, die dann TRUE ist, wenn die Variable für die Spaltenauswahl einen gültigen Wert enthält
Variable für Gültigkeit der Zeilenauswahl	Boolesche Variable, die dann TRUE ist, wenn die Variable für die Spaltenauswahl einen gültigen Wert enthält

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

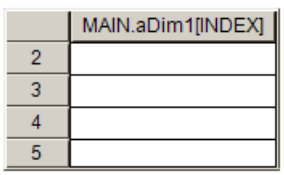
Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.7.1 Konfiguration einer Tabelle

Zunächst ist die in der Tabelle zu visualisierende Variable zu spezifizieren. Dazu wird in "Datenarray" eine Variable mit vollständigem Pfad eingetragen. Die Struktur der Variablen bestimmt die Anzahl an Zeilen und Spalten der Tabelle. Im Falle eines zweidimensionalen Arrays bestimmt der erste Index die Anzahl der Spalten und der zweite Index die Anzahl der Zeilen. Bei einer Struktur repräsentieren die einzelnen Komponenten der Struktur die Spalten:

- Beispiel 1:

```
aDim1 : ARRAY [2..5] OF INT;
```



- Beispiel 2:

```
aDim2 : ARRAY [0..2, 0..3] OF INT;
```

	MAIN.aDim2[0,INDEX]	MAIN.aDim2[1,INDEX]	MAIN.aDim2[2,INDEX]
0			
1			
2			
3			

- Beispiel 3:

```
TYPE ST_ProductInformation :
STRUCT
    nOrderInPieces : INT;
    bStocked : BOOL;
    nArcticelNumber : INT;
END_STRUCT
END_TYPE

stProductInformation : ST_ProductInformation;
```

	MAIN.stProductInformation.nOrderInPieces	MAIN.stProductInformation.bStocked	MAIN.stProductInformation.nArticelNumber
0			

- Beispiel 4:

```
aProductInformation : ARRAY [0..3] OF ST_ProductInformation;
```

	MAIN.aProductInformation[INDEX].nOrderInPieces	MAIN.aProductInformation[INDEX].bStocked	MAIN.aProductInformation[INDEX].nArticelNumber
0			
1			
2			
3			

- Beispiel 5:

```
fbTimer : TON;
```

	MAIN.fbTimer.IN	MAIN.fbTimer.PT	MAIN.fbTimer.Q	MAIN.fbTimer.ET	MAIN.fbTimer.M	MAIN.fbTimer.StartTime
0						

- Beispiel 6:

```
aTimers : ARRAY [0..3] OF TON;
```

	MAIN.aTimers[INDEX].IN	MAIN.aTimers[INDEX].PT	MAIN.aTimers[INDEX].Q	MAIN.aTimers[INDEX].ET	MAIN.aTimers[INDEX].M	MAIN.aTimers[INDEX].StartTime
0						
1						
2						
3						

Wenn der Eintrag im Feld "Datenarray" geändert wird, werden alle übrigen Einstellungen für die Elementeigenschaften wieder auf die Standardeinstellung zurückgesetzt!

Im Folgenden wird das Beispiel 4 genutzt.

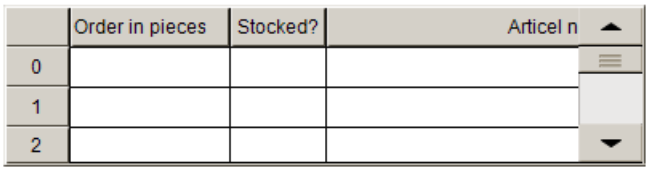
Nachdem eine Variable in "Datenarray" eingetragen worden ist, können die Eigenschaften jeder einzelnen Spalte geändert werden. Als erstes werden die Spaltenbeschriftungen in aussagekräftigere Titel abgeändert. Dann werden die Breite der Spalten und die Ausrichtung der Überschrift geändert.

Es gibt zwei Möglichkeiten, die Breite der Spalte zu ändern:

1. In den Elementeigenschaften kann unter der jeweiligen Spalte in der Eigenschaft "Breite" der Wert geändert werden.
2. Direkt im Tabellenelement kann die Maus auf den Trennstrich zwischen zwei Spalten gefahren werden. Das dort entstehende Zeigerbild  zeigt an, dass es nun möglich ist, bei betätigter Maustaste die Größe der Spalte zu ändern.

	Order in pieces	Stocked?	Articel number
0			
1			
2			
3			

Die Höhe der Zeilen wird in der Eigenschaft "Zeilenhöhe" auf 25 Pixel festgesetzt. Das folgende Bild zeigt zudem die verbreitete Bildlaufleisten, deren Größe unter "Größe des Scrollbars" eingestellt werden kann. Die Scrollbars werden zur Tabelle hinzugefügt, sobald die Tabelle auf eine Größe unter der Summe der Zeilenhöhen bzw. der Spaltenbreiten verkleinert wird.



Um die Bildlaufleisten wieder verschwinden zu lassen, wird die Größe der Tabelle angepasst. Ihre Höhe ergibt sich aus der Zeilenhöhe multipliziert mit der Anzahl an Zeilen (+1 im Fall von eingeschalteter Spaltenbeschriftung). Die Breite ist die Summe aus den einzelnen Spaltenbreiten und der Breite der Zeilenbeschriftung, sofern sie aktiviert ist.

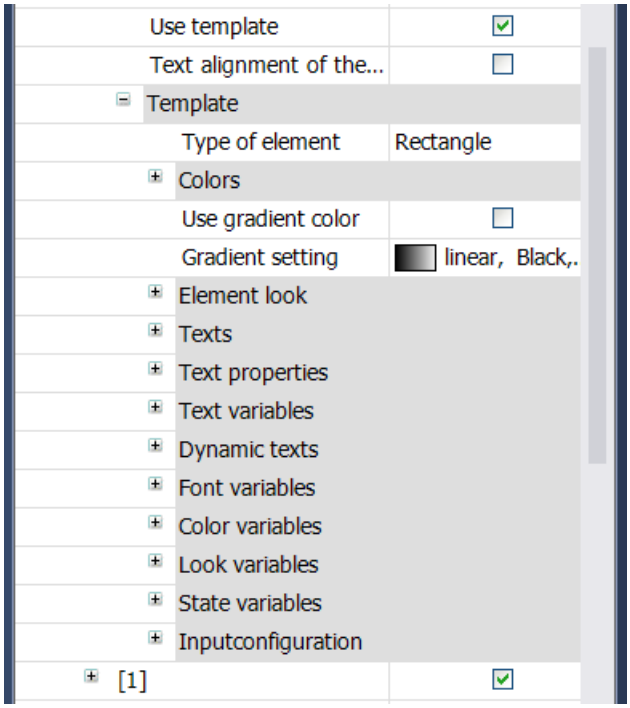
Die Schriftart und -größe können innerhalb der Dialogbox ausgewählt werden, welche sich nach einem Klick auf die Schaltfläche  in "Schriftart" öffnet. Im Beispiel wird die Schriftfarbe unter "Farbe Schriftart" in "Blau" geändert.

Zur Laufzeit sieht die Tabelle wie folgt aus:

	Order in pieces	Stocked?	Articel number
0	250	TRUE	32136832
1	480	FALSE	45983920
2	1500	TRUE	46001235
3	680	TRUE	55129547

Beachten Sie, dass die horizontale Ausrichtung der Zellen grundsätzlich in den Einstellungen der jeweiligen Spalten in "Textausrichtung der Überschrift" erfolgt. Außer es wird, wie im folgenden Teil beschrieben, spaltenweise eine Vorlage für die Zellen genutzt.

Die Tabellenfelder selbst können spaltenweise editiert werden. Aktivieren Sie zu diesem Zweck für die ausgewählte Spalte das Kontrollkästchen "Vorlage verwenden", woraufhin innerhalb der Spalteneigenschaften ein weiterer Punkt "Vorlage" eingefügt und aufgeklappt wird.



Drei verschiedene Vorlagentypen stehen zur Auswahl. Standardmäßig ist "Rechteck" eingetragen. Das kann jedoch durch Klicken in das Wertefeld der Eigenschaft "Elementtyp" in "Abgerundetes Rechteck" oder "Ellipse" geändert werden. Die zugehörigen Konfigurationsmöglichkeiten zeigen die Elementeigenschaften der ausgewählten Vorlage. Wie gewöhnlich können Sie die Füll- und Rahmenfarben im Normal- und Alarmzustand wählen, die auch mithilfe einer Variablen umgeschaltet werden können. Die Ausrichtung der Einträge in den Zellen der Spalten wird nun mit Ausnahme der Überschriftzellen durch die Einstellung in der Vorlage bestimmt.

Anstelle der standardmäßig eingetragenen Array-Komponenten kann in "Textvariable" auch jede andere Variable des Projekts eingetragen werden. Somit ist es insbesondere möglich, die Array-Elemente auch in veränderter Reihenfolge in der Tabelle anzeigen zu lassen.

Ein Vertauschen der Spalten kann auch in der Tabelle selbst vorgenommen werden. Klicken Sie hierzu innerhalb der Titelzeile in die Mitte einer Spalte und ziehen die Spalte mit gedrückter Maustaste auf ihre neue Position.

In diesem Beispiel wurde für die erste und dritte Spalte ein Template benutzt.

	Order in pieces	Stocked?	Articel number
0	250	TRUE	32136832
1	480	FALSE	45983920
2	1500	TRUE	46001235
3	680	TRUE	55129547

Um eine Selektion einer Zelle sichtbar zu machen, werden abschließend die Einstellungen unter "Selektion" angepasst. Zunächst wird die Selektionsfarbe auf "Hellblau" gestellt und die Auswahlart auf "Zeilenauswahl". Wenn nun zur Laufzeit eine Zelle selektiert wird, werden automatisch auch alle anderen Zellen zugehörigen Zeile hervorgehoben.

	Order in pieces	Stocked?	Articel number
0	250	TRUE	32136832
1	480	FALSE	45983920
2	1500	TRUE	46001235
3	680	TRUE	55129547

15.8.2.8 Textfeld

Mit diesem Element ist es möglich Text anzuzeigen, der direkt in den Eigenschaften [▶ 474] oder variabel in "Textvariablen [▶ 475]" zugewiesen wird. Im Gegensatz zum Rechteck kann der Rahmen mit Schatten dargestellt werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche ... die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche ... geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Normalzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Transparenz wird unter Windows CE nicht unterstützt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach „16#“ dargestellt. Farben, dessen Definition mit „16#00“ beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements im Alarmzustand bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehen

Die Einstellungen unter Aussehen sind statische Definitionen bezüglich der Rahmenlinie und der Füllung des Elements.

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Füllart	Definiert die Art der Füllung für die Füllfarbe: • Ausgefüllt: Die Füllfarbe ist sichtbar • Unsichtbar: Die Füllfarbe ist nicht sichtbar
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Schattenart	Hier kann die Art des Schattens der Umrisslinie festgelegt werden: • Versenkt • Kein Schatten • Erhoben • Aus Stil [▶ 407]: Standardeinstellung

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.9 Scrollbalken

Dieses Element erzeugt eine Bildlaufleiste. Wenn ein Anwender zur Laufzeit der Visualisierung das Bildlauffeld auf eine andere Position zieht, wird dementsprechend die Eigenschaft "Wert [▶ 478]" gesetzt. Die Ausrichtung der Bildlaufleiste kann von horizontal in vertikal geändert [▶ 479] werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Wert	Variable vom Typ Integer, welche die Position des Bildlauffeldes im Scrollbalken enthält.
Minimaler Wert	Kleinstmöglicher Wert der Bildlaufleiste
Maximaler Wert	Größtmöglicher Wert der Bildlaufleiste
Seitengröße	Wenn ein Anwender zur Laufzeit in die Bildlaufleiste klickt, wird die Position des Bildlauffeldes um die Seitengröße in Richtung der Mausposition verschoben. Die Seitengröße kann als fester Wert in Pixeln oder als Variable vom Typ Integer angegeben werden.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Balken

Ausrichtung	Hier kann die Ausrichtung des Balken festgelegt werden: • Horizontal • Vertikal Sie können auch direkt in der Visualisierung die Ausrichtung konfigurieren, indem Sie die Breite und Höhe des Scrollbalkens ändern.
Laufrichtung	Wenn die Ausrichtung horizontal ist, können Sie die Laufrichtung wie folgt wählen: • Links nach rechts • Rechts nach links Wenn die Ausrichtung vertikal ist, können Sie die Laufrichtung wie folgt wählen: • Von unten nach oben • Von oben nach unten

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die

Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Normalzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Transparenz wird unter Windows CE nicht unterstützt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut:
16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach „16#“ dargestellt. Farben, dessen Definition mit „16#00“ beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements im Alarmzustand bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche [...] öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn [...] geklickt wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := ‘Arial’;)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart‘. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

15.8.2.10 Schieberegler

Der Balken des Schiebereglers kann mithilfe der Maus verschoben werden, wodurch eine dem Schieberegler zugewiesene Variable ihren Wert innerhalb der definierten Grenzen verändert.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Numerische Variable, welche die Position des Reglers enthalten wird.
Seitengröße	Seitengröße • Als fester Wert, zum Beispiel 10 • Als ganzzahliger Variable • Voraussetzung: Die Elementeigenschaft „Zum Klick Springen“ ist **nicht** aktiviert.
Zum Klick springen	Verhalten des Schiebereglers zur Laufzeit der Visualisierung bei Klick in die Schiebereglerleiste: • Aktiviert: Der Schieberegler springt auf die angeklickte Position. • Deaktiviert: Der Schieberegler springt um den Wert, der in Elementeigenschaft „Seitengröße“ definiert ist, in die Richtung, die der Klick vorgibt.

Skala

Skala anzeigen	Wenn diese Option aktiviert ist, wird die Skala angezeigt.
Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala. Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenformat (C-Syntax)	Wird ein Skalenformat, wie zum Beispiel „%d s“ konfiguriert, dann wird zusätzlich eine Skalenbeschriftung angezeigt.
Skalenverhältnis	Größe der Skala in % der Gesamtgröße

Balken

Diagrammart	Die Drop-down-Liste stellt verschiedene Möglichkeiten zur Verfügung, wie Balken und Skale platziert werden können: • Skala neben Balken • Skala im Balken • Balken in Skala • Keine Skala
Ausrichtung	Der Balken kann horizontal oder vertikal ausgerichtet sein. Die Ausrichtung entsteht aus dem Verhältnis von Breite zu Höhe und kann hier nicht editiert werden. Diese Änderung erfolgt im Visualisierungseditor [▶ 394], indem ein Eckpunkt es Elements mit der Maus "gegriffen" und horizontal oder vertikal gezogen wird.
Laufrichtung	Ist die Ausrichtung horizontal, kann gewählt werden zwischen: • Links nach rechts • Rechts nach links Ist die Ausrichtung vertikal, kann gewählt werden zwischen: • Von unten nach oben • Von oben nach unten

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.11 SpinControl

Das Element "SpinControl" erlaubt es, den Wert einer Variablen per Mausklick auf die kleinen Pfeiltasten an der rechten Seite des Elements zu inkrementieren oder zu dekrementieren.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Numerische Variable, deren Wert nach einer Benutzereingabe ansteigt oder abnimmt.
Zahlenformat	Formatierungssequenz, mit der der Variablenwert angezeigt wird. Beispiel: %i für eine Integer-Variable
Intervall	Intervall (Schrittweite), um den der Variablenwert nach einer Benutzereingabe inkrementiert oder dekrementiert wird.

Value range

Minimaler Wert	Hier ist der maximalmögliche Wert anzugeben.
Maximaler Wert	Hier ist der minimalmögliche Wert anzugeben.

Texteigenschaften

Verwendung von	Hier kann zwischen den folgenden beiden Einstellungen gewählt werden: • Standard Stilwerten • Benutzerdefinierte Einstellungen ◦ Horizontale Ausrichtung ◦ Vertikale Ausrichtung ◦ Schriftart ◦ Farbe Schriftart

Benutzerdefinierte Texteigenschaften

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche [...] öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn [...] geklickt wird.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach „16#“ dargestellt. Farben, dessen Definition mit „16#00“ beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.

Transparenz wird unter Windows CE nicht unterstützt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

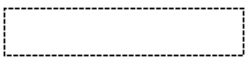
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.12 Unsichtbare Eingabe

Das Element wird im Editor mit einer gestrichelten Linie angezeigt, aber im Onlinebetrieb ist diese unsichtbar. Das Verhalten des Elements ist in der Eingabekonfiguration zu spezifizieren.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,

- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick

- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Eingabe deaktiviert	Boolesche Variable. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.13 Ladebalken

Ein Ladebalken dient der grafischen Darstellung eines Fortschritts. Dies erfordert eine Variable, deren Wert als Balken angezeigt wird. Die Grenzen [▶ 493] der Variable sowie der Anzeigestil [▶ 493] kann eingestellt werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Variable	Numerische Variable, deren Wert als Ladebalken visualisiert wird.
Minimaler Wert	Kleinstmöglicher Wert
Maximaler Wert	Größtmöglicher Wert
Stil	Hier einen der folgenden Stile auswählen: • Blöcke • Balken

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Texte

Text	Hier kann der Text eingestellt werden, der links bzw. oberhalb der Schaltflächenleiste ausgegeben wird. Wenn das Element horizontal ausgerichtet ist, ist der Text links positioniert. Wenn das Element vertikal ausgerichtet ist, ist der Text oberhalb positioniert.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Boolesche Variable. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.

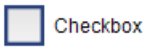
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.14 Checkbox

Mit einer Checkbox kann eine boolesche Variable gesetzt und rückgesetzt werden. Ist der Haken gesetzt, wird die Variable auf TRUE geschrieben.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Boolesche Variable, in der der Zustand der Checkbox abgespeichert wird.
Rahmengröße	Rahmengröße der Checkbox Standardeinstellung: Stil [▶ 407]

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist.

Texteigenschaften

Verwendung von	Hier kann zwischen den folgenden beiden Einstellungen gewählt werden: • Standard Stilwerten • Benutzerdefinierte Einstellungen ◦ Horizontale Ausrichtung ◦ Vertikale Ausrichtung ◦ Schriftart ◦ Farbe Schriftart

Benutzerdefinierte Texteigenschaften

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Zugriffsrechte

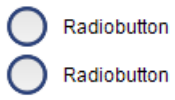
Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.2.15 Radiobutton

Ein Radiobutton erlaubt es eine beliebige Anzahl an Optionen durch Verwendung der Schaltfläche

zu konfigurieren, um sie in einer oder mehreren Spalten anzuordnen.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Variable vom Typ Integer, die mit dem Index des selektierten Auswahlknopfes beschrieben wird.
Anzahl der Spalten	Spaltenanzahl, mit der die Radiobuttons angeordnet werden.
Anordnung der Radiobuttons	Hier auswählen, wie die Radiobuttons angeordnet werden sollen: • Von links nach rechts: Die Radiobuttons werden Zeile für Zeile angeordnet, bis alle Buttons platziert worden sind. • Von oben nach unten: Die Radiobuttons werden spaltenweise angeordnet, bis alle Buttons platziert worden sind.
Rahmengröße	Größe der Radiobuttons Standardeinstellung: Von Stil [▶ 407]
Zeilenhöhe	Definition der Zeilenhöhe Standardeinstellung: Von Stil [▶ 407]

Texteigenschaften

Verwendung von	Hier kann zwischen den folgenden beiden Einstellungen gewählt werden: • Standard Stilwerten • Benutzerdefinierte Einstellungen ◦ Horizontale Ausrichtung ◦ Vertikale Ausrichtung ◦ Schriftart ◦ Farbe Schriftart

Benutzerdefinierte Texteigenschaften

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche  öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn  geklickt wird.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Radiobuttoneinstellungen

Radiobutton • Bereiche ◦ [<n>]	Mit einem Klick auf die Schaltfläche  wird ein neuer Radiobutton im Editor erzeugt und ein weiterer Bereich wird im Eigenschafteneditor aufgelistet. Für jeden Radiobutton wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf Löschen wird der zugehörige Radiobutton mit seinen Einstellungen gelöscht.

Bereiche [<n>]

Text	Buttonname Standardwert: Radiobutton
Tooltip	Text, der als Tooltip auszugeben ist
Zeilenabstand in Pixel	Abstand in Pixel zum oberen Button

Zugriffsrechte

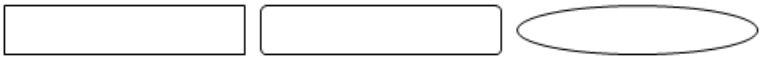
Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3 Basis

15.8.3.1 Rechteck, abgerundetes Rechteck und Ellipse

Die Visualisierungselemente Rechteck, abgerundetes Rechteck und Ellipse sind vom gleichen Elementtyp. Sie können nur durch Ändern der Eigenschaft Elementtyp gegenseitig konvertiert werden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Normalzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Farbverlauf verwenden	Standardmäßig ist die Checkbox deaktiviert. Falls sie aktiviert wird, wird das zugehörige Element mit Farbverlauf gezeichnet.
Farbverlaufsauswahl	Der Farbverlaufeditor [▶ 424] öffnet sich.

Farbverlauf und Transparenz werden unter Windows CE nicht unterstützt.

Aussehen

Die Einstellungen unter Aussehen sind statische Definitionen bezüglich der Rahmenlinie und der Füllung des Elements.

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Füllart	Definiert die Art der Füllung für die Füllfarbe: • Ausgefüllt: Die Füllfarbe ist sichtbar • Unsichtbar: Die Füllfarbe ist nicht sichtbar
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.
Innere Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um seinen Rotationspunkt gedreht wird; positive Werte = mathematisch positiv = Uhrzeigersinn. Im Gegensatz zu "Rotation" (s.o.) rotiert das Element selbst. Der Rotationspunkt (Zentrum ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Relative Bewegung

Das Element kann relativ zu seiner fixen Position verschoben werden. Die obere-linke, und untere-rechte Kante des Elements werden um den durch eine Integer-Variable definierten Wert (Pixel) in X- bzw. Y-Richtung verschoben. Im Gegensatz zur absoluten Bewegung wird eine relative Position definiert, d.h. der Abstand zur ursprünglichen Position. Das Element kann dadurch in der Form verändert werden. Positive Werte verschieben die horizontalen Kanten nach unten bzw. die vertikalen nach rechts.

Bewegung links-oben • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in Y-Richtung verschoben wird.
Bewegung rechts-unten • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in Y-Richtung verschoben wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := ‘Arial’;)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart‘. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach „16#“ dargestellt. Farben, dessen Definition mit „16#00“ beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements im Alarmzustand bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehensvariablen

Anzuwenden bei einer dynamischen Definition des Aussehens der Kontur und der Füllung des Elements. Statische Definition werden in "Aussehen" festgelegt.

Linienstärke	Angabe einer Variablen vom Typ Integer, die die Linienstärke des Elements in Pixel definiert. Das überschreibt den fixen Wert, der in "Aussehen" festgelegt wurde.
Füllart	Angabe einer Variablen vom Typ DWORD, die die Füllung des Elements definiert. Die Farbe, die bei Farbvariablen angegeben ist, kann angezeigt oder ignoriert werden: • Variablenwert = 0: gefüllt • Variablenwert > 0: unsichtbar, das heißt keine Füllung ist sichtbar
Linienart	Angabe einer Variablen vom Typ DWORD, die die Kontur der Außenlinie festlegt. Die folgenden Werte entsprechen folgenden Linienarten: • 0: Durchgezogen • 1: Strich • 2: Punkt • 3: Strich Punkt • 4: Strich Punkt Punkt • 8: Unsichtbar. Das heißt, das Element wird ohne Außenlinie dargestellt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed

- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3.2 Linie

Das Element ‚Linie' ist eine einfache Linie, die durch zwei definierte Punkt charakteristisch beschrieben wird.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier sind die für das Visualisierungselement charakteristischen Positionen (X/Y-Koordinaten) jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das entsprechende Element im Visualisierungseditor [▶ 394] geändert.

Die Anzahl der Punkte, durch die das Element definiert ist, kann erhöht oder verringert werden. Die Auswahl einer der Punkte im Visualisierungseditor bei gedrückter Steuerungstaste hat zur Folge, dass dieser Punkt dupliziert wird. Dadurch wird automatisch ein Eintrag für einen weiteren Punkt in den Eigenschaften hinzugefügt. Um einen der Punkte zu löschen, muss dieser Punkt bei gleichzeitig gedrückter Steuerungs- und Umschalttaste selektiert werden. Der entsprechende Eintrag wird dann in den Eigenschaften gelöscht.

[0] X / Y ..... [n] X / Y	X/Y-Positionen (in Pixel) der einzelnen Elementpunkte [0] ist die Position des Starkpunktes. Die folgenden Punkte werden fortlaufend nummeriert.
X	Horizontale Position in Pixel X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel Y=0 ist der obere Fensterrand.

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die

Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehen

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.
Innere Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um seinen Rotationspunkt gedreht wird; positive Werte = mathematisch positiv = Uhrzeigersinn. Im Gegensatz zu "Rotation" (s.o.) rotiert das Element selbst. Der Rotationspunkt (Zentrum ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Relative Bewegung

Das Element kann relativ zu seiner fixen Position verschoben werden. Die obere-linke, und untere-rechte Kante des Elements werden um den durch eine Integer-Variable definierten Wert (Pixel) in X- bzw. Y-Richtung verschoben. Im Gegensatz zur absoluten Bewegung wird eine relative Position definiert, d.h. der Abstand zur ursprünglichen Position. Das Element kann dadurch in der Form verändert werden. Positive Werte verschieben die horizontalen Kanten nach unten bzw. die vertikalen nach rechts.

Bewegung Punkt [n] • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die der Punkt[0]/ Punkt[1] in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die der Punkt[0]/ Punkt[1] in Y-Richtung verschoben wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := ‘Arial’;)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart‘. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach "16#" dargestellt. Farben, dessen Definition mit "16#00" beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in der Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements im Alarmzustand bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in den Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Linienartvariable

Dynamische Definition der Linienart mittels einer Variablen.

Ganzzahlwert	Angabe einer Variablen mit ganzzahligem Datentypen. Der Wert definiert die Linienart. Folgende Variablenwerte werden unterstützt: • 0 = durchgezogen • 1 = Striche • 2 = Punkte • 3 = Strich Punkt • 4 = Strich Punkt Punkt • 8 = unsichtbar: Die Linie ist unsichtbar. Dies überschreibt den fixen Wert, der in der Kategorie "Aussehen" festgelegt ist.

Linienstärkenvariable

Dynamische Definition der Dicke eines Linienelements mittels einer Variablen.

Ganzzahlwert	Angabe einer Variablen vom Typ Integer, die die Linienstärke des Elements in Pixel definiert. Dies überschreibt den fixen Wert, der in der Kategorie "Aussehen" festgelegt ist. Zu berücksichtigen ist, dass 0 das Gleiche wie 1 kodiert und die Linienstärke dabei auf 1 Pixel gesetzt wird.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

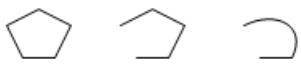
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3.3 Polygon, Linienzug oder Bézierkurve

Polygon, Linienzug und Bézierkurve sind vom gleichen Elementtyp. Sie können nur durch Ändern der Eigenschaft Elementtyp gegenseitig konvertiert werden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier sind die für das Visualisierungselement charakteristischen Positionen (X/Y-Koordinaten) jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das entsprechende Element im Visualisierungseditor [▶ 394] geändert.

Die Anzahl der Punkte, durch die das Element definiert ist, kann erhöht oder verringert werden. Die Auswahl einer der Punkte im Visualisierungseditor bei gedrückter Steuerungstaste hat zur Folge, dass dieser Punkt dupliziert wird. Dadurch wird automatisch ein Eintrag für einen weiteren Punkt in den Eigenschaften hinzugefügt. Um einen der Punkte zu löschen, muss dieser Punkt bei gleichzeitig gedrückter Steuerungs- und Umschalttaste selektiert werden. Der entsprechende Eintrag wird dann in den Eigenschaften gelöscht.

[0] X / Y ..... [n] X / Y	X/Y-Positionen (in Pixel) der einzelnen Elementpunkte [0] ist die Position des Starkpunktes. Die folgenden Punkte werden fortlaufend nummeriert.
X	Horizontale Position in Pixel X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel Y=0 ist der obere Fensterrand.

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Normalzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Farbverlauf verwenden	Standardmäßig ist die Checkbox deaktiviert. Falls sie aktiviert wird, wird das zugehörige Element mit Farbverlauf gezeichnet.
Farbverlaufsauswahl	Der Farbverlaufeditor [▶ 424] öffnet sich.

Farbverlauf und Transparenz werden unter Windows CE nicht unterstützt.

Aussehen

Die Einstellungen unter Aussehen sind statische Definitionen bezüglich der Rahmenlinie und der Füllung des Elements.

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Füllart	Definiert die Art der Füllung für die Füllfarbe: • Ausgefüllt: Die Füllfarbe ist sichtbar • Unsichtbar: Die Füllfarbe ist nicht sichtbar
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.
Innere Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um seinen Rotationspunkt gedreht wird; positive Werte = mathematisch positiv = Uhrzeigersinn. Im Gegensatz zu "Rotation" (s.o.) rotiert das Element selbst. Der Rotationspunkt (Zentrum ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Dynamische Punkte

Eine dynamische Definition für die Punkte, die das Element definieren.

Array von Punkten	Angabe einer Variablen, die auf ein Array der Struktur VisuElems.VisuStructPoint zeigt. Die VisuElems.VisuStructPoint-Komponenten iX und iY enthalten die X/ Y-Koordination eines Punktes. Das Array enthält alle Punkte des Elements und die Struktur kann dynamische durch eine Projektvariable gefüllt werden. Die Anzahl der Elementpunkte muss explizit in der folgenden Einstellung ‚Anzahl Punkte' deklariert werden. Beispiel: pPoints : POINTER TO ARRAY[1..100] OF VisuElems.VisuStructPoint;
Anzahl Punkte	Angabe einer Variablen vom Typ Integer, die die Anzahl der Punkte des Elements definiert. Beispiel: nCount : INT := 24; Das Element wird durch 24 einzelne Punkte definiert. Diese Festlegung ist notwendig, weil die Definition der einzelnen Punkte über einen Zeiger erfolgt und dies nicht erlaubt, die Anzahl zu kontrollieren.

Um die dynamische Definition von Punkten zu ermöglichen, muss entweder die PLC HMI [▶ 635], die PLC HMI Web [▶ 640] oder beide aktiviert worden sein.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := ‘Arial’;)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart‘. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Aussehensvariablen

Anzuwenden bei einer dynamischen Definition des Aussehens der Kontur und der Füllung des Elements. Statische Definition werden in "Aussehen" festgelegt.

Linienstärke	Angabe einer Variablen vom Typ Integer, die die Linienstärke des Elements in Pixel definiert. Das überschreibt den fixen Wert, der in "Aussehen" festgelegt wurde.
Füllart	Angabe einer Variablen vom Typ DWORD, die die Füllung des Elements definiert. Die Farbe, die bei Farbvariablen angegeben ist, kann angezeigt oder ignoriert werden: • Variablenwert = 0: gefüllt • Variablenwert > 0: unsichtbar, das heißt keine Füllung ist sichtbar
Linienart	Angabe einer Variablen vom Typ DWORD, die die Kontur der Außenlinie festlegt. Die folgenden Werte entsprechen folgenden Linienarten: • 0: Durchgezogen • 1: Strich • 2: Punkt • 3: Strich Punkt • 4: Strich Punkt Punkt • 8: Unsichtbar. Das heißt, das Element wird ohne Außenlinie dargestellt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

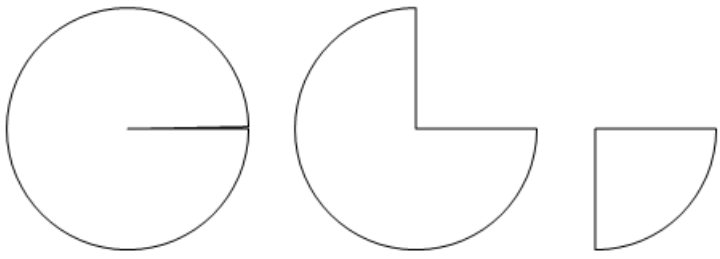
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3.4 Kreissektor

Das Element "Kreissektor" stellt grundsätzlich einen Vollkreis mit der Mittelposition an dem Punkt dar, an dem das Element auf die Visualisierung gesetzt wird. Der anzuzeigende Ausschnitt kann in den Einstellungen [▶ 526] angepasst werden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche … die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Anfang	Anfang der Kreissektorlinie
Ende	Ende der Kreissektorlinie
Variable für Anfang	Variable, die den Anfang der Kreissektorlinie dynamisch definieren soll. Für Hilfe vom Eingabeassistenten ist die Schaltfläche … zu betätigen.
Variable für Ende	Variable, die das Ende der Kreissektorlinie dynamisch definieren soll. Für Hilfe vom Eingabeassistenten ist die Schaltfläche … zu betätigen.
Nur Kreisbogen anzeigen	Falls diese Option aktiviert ist, wird der Kreissektor ohne Radiuslinie für Anfang und Ende dargestellt.

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die

Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Normalzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmzustand • Rahmenfarbe • Füllfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Farbverlauf verwenden	Standardmäßig ist die Checkbox deaktiviert. Falls sie aktiviert wird, wird das zugehörige Element mit Farbverlauf gezeichnet.
Farbverlaufsauswahl	Der Farbverlaufeditor [▶ 424] öffnet sich.

Farbverlauf und Transparenz werden unter Windows CE nicht unterstützt.

Aussehen

Die Einstellungen unter Aussehen sind statische Definitionen bezüglich der Rahmenlinie und der Füllung des Elements.

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Füllart	Definiert die Art der Füllung für die Füllfarbe: • Ausgefüllt: Die Füllfarbe ist sichtbar • Unsichtbar: Die Füllfarbe ist nicht sichtbar
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.
Innere Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um seinen Rotationspunkt gedreht wird; positive Werte = mathematisch positiv = Uhrzeigersinn. Im Gegensatz zu "Rotation" (s.o.) rotiert das Element selbst. Der Rotationspunkt (Zentrum ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := 'Arial';)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart'. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach „16#" dargestellt. Farben, dessen Definition mit „16#00" beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)

- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand • Rahmenfarbe • Füllfarbe	Angabe von Variablen vom Typ DWORD, die die Rahmen- und Füllfarbe des Elements im Alarmzustand bestimmen. Sie überschreiben die Werte, die aktuell in "Farben" definiert sind. Die Werte in den Projektvariablen werden verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Aussehensvariablen

Anzuwenden bei einer dynamischen Definition des Aussehens der Kontur und der Füllung des Elements. Statische Definition werden in "Aussehen" festgelegt.

Linienstärke	Angabe einer Variablen vom Typ Integer, die die Linienstärke des Elements in Pixel definiert. Das überschreibt den fixen Wert, der in "Aussehen" festgelegt wurde.
Füllart	Angabe einer Variablen vom Typ DWORD, die die Füllung des Elements definiert. Die Farbe, die bei Farbvariablen angegeben ist, kann angezeigt oder ignoriert werden: • Variablenwert = 0: gefüllt • Variablenwert > 0: unsichtbar, das heißt keine Füllung ist sichtbar
Linienart	Angabe einer Variablen vom Typ DWORD, die die Kontur der Außenlinie festlegt. Die folgenden Werte entsprechen folgenden Linienarten: • 0: Durchgezogen • 1: Strich • 2: Punkt • 3: Strich Punkt • 4: Strich Punkt Punkt • 8: Unsichtbar. Das heißt, das Element wird ohne Außenlinie dargestellt.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3.5 Bild

Mit diesem Element kann in der Visualisierung ein Bild eingefügt werden. Ein Bild wird dem Bildelement zugewiesen, indem die entsprechende statische ID in Kombination mit dem Namen der Bildersammlung [▶ 153] angegeben wird. Mithilfe der Einstellung "Bild-ID-Variable [▶ 537]" kann das darzustellende Bild auch dynamisch definiert werden.

Ein Hintergrundbild für die Visualisierungsseite kann über einen Hintergrunddialog [▶ 395] im Visualisierungseditor eingestellt werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Statische ID	Bezeichner der Bilddatei für eine statische Definition. Geben Sie die ID der Bilddatei ein, wie sie in der entsprechenden Bildersammlung definiert ist (String). Der Name der Sammlung sollte dabei vorangestellt werden, um den Eintrag eindeutig zu machen. (Dies ist nicht notwendig, wenn die Bilddatei im GlobalImagePool verwaltet wird, da diese Sammlung immer als erste durchsucht wird.) Beispiel: IP_ImagePool.ButtonImage Mit Klick auf die Schaltfläche  öffnet sich der Dialog "Eingabehilfe [▶ 543]" und listet alle verfügbaren Bildersammlungen mit dazugehörigen Bildern auf.
Rahmen zeichnen	Wenn diese Option aktiviert ist, wird die Bilddatei mit einem Rahmen dargestellt.
Abschneiden	Wenn diese Option zusammen mit Skalierungsart "Unskaliert" aktiviert ist, wird nur der Teil des Bildes, der in das Element passt, dargestellt.
Transparenz	Ist die Option aktiviert, wird die Farbe, die in der Eigenschaft "Transparenzfarbe" angegeben ist, transparent dargestellt.
Transparenzfarbe	Die Schaltfläche  öffnet den Farbenauswahldialog zur Auswahl einer Farbe, die transparent dargestellt werden soll, wenn die Option "Transparenz" aktiviert ist.
Skalierungsart	Hier definieren, wie die Bilddatei auf Änderungen der Elementrahmengrö0e reagieren soll: • Isotropisch Das Bild behält seine Proportionen. Das heißt, das Verhältnis von Höhe zu Breite bleibt erhalten, auch wenn die Höhe und Breite des Elementrahmens separat verändert werden. • Anisotropisch Das Bild passt sich der Größe des Elementrahmens an. D.h. Höhe und Breite können unabhängig voneinander verändert werden. • Unskaliert Das Bild behält die Originalgröße, auch wenn sich die Größe des Elementrahmens ändert. Zu berücksichtigen ist auch, ob auch Option "Abschneiden" aktiviert ist. Bei dieser Einstellung wird bei jeder Neuzuweisung einer ImageID die Elementgröße auf die Bildgröße automatisch angepasst.
Horizontale Ausrichtung	Diese Einstellung ist nur verfügbar, wenn die Option "Experte" des Eigenschafteneditors [▶ 403] aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Sie dient dazu, die horizontale Ausrichtung zum Rahmen des Elements zu definieren. • Links • Zentriert • Rechts
Vertikale Ausrichtung	Diese Einstellung ist nur verfügbar, wenn die Option "Experte" des Eigenschafteneditors [▶ 403] aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Sie dient dazu, die vertikale Ausrichtung zum Rahmen des Elements zu definieren. • Oben • Zentriert • Unten

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche [...] geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehen

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Bild-ID-Variable

Bild-ID	Projektvariable vom Typ String, die die Bild-ID enthält, mit der das Element darzustellen ist. Die ID ist in der Bildersammlung [▶ 153] definiert. Der Name der Sammlung sollte dabei vorangestellt werden, um den Eintrag eindeutig zu definieren. Nur bei Bilddateien, die im "GlobalImagePool" verwaltet werden, muss keine Sammlung angegeben werden, da diese Sammlung immer als erste durchsucht wird.

Dynamisches Bild

Mithilfe dieser Eigenschaft kann eine Bilddatei zur Laufzeit neu eingelesen werden. Auf diese Weise ist es zum Beispiel möglich, Bilder einer Kamera zu aktualisieren.

Die Bilddatei, welche neu geladen werden soll, muss

- im Falle der PLC HMI mit einem Build <4022.28 im Client-Verzeichnis des Geräts, auf dem der PLC HMI Client gestartet ist, ausgetauscht werden (*C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi\[Port_X]\Visu*).
- im Falle der PLC HMI mit einem Build >=4022.28 und der PLC HMI Web im Boot-Verzeichnis des Geräts, auf dem der Steuerungscode ausgeführt wird, ausgetauscht werden (*C:\TwinCAT\3.1\Boot\Plc\[Port_X]\Visu*).
- im Falle der PLC HMI und der PLC HMI Web mit einem Build >=TC3.1.4026.0 im Boot-Verzeichnis des Geräts, auf dem der Steuerungscode ausgeführt wird, ausgetauscht werden (*C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc\[Port_X]\Visu*).

Bitmap-Version	Variable eines ganzzahligen Datentyps, welche die Version des Bildes enthält. Wenn der Wert der Variable sich ändert, liest die Visualisierung das in der Eigenschaft "Bild-ID" referenzierte Bild neu ein und zeigt es an.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.
Innere Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um seinen Rotationspunkt gedreht wird; positive Werte = mathematisch positiv = Uhrzeigersinn. Im Gegensatz zu "Rotation" (s.o.) rotiert das Element selbst. Der Rotationspunkt (Zentrum ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Relative Bewegung

Das Element kann relativ zu seiner fixen Position verschoben werden. Die obere-linke, und untere-rechte Kante des Elements werden um den durch eine Integer-Variable definierten Wert (Pixel) in X- bzw. Y-Richtung verschoben. Im Gegensatz zur absoluten Bewegung wird eine relative Position definiert, d.h. der Abstand zur ursprünglichen Position. Das Element kann dadurch in der Form verändert werden. Positive Werte verschieben die horizontalen Kanten nach unten bzw. die vertikalen nach rechts.

Bewegung links-oben • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in Y-Richtung verschoben wird.
Bewegung rechts-unten • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in Y-Richtung verschoben wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := 'Arial';)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart'. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach "16#" dargestellt. Farben, dessen Definition mit "16#00" beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in der Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements im Alarmzustand bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in den Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

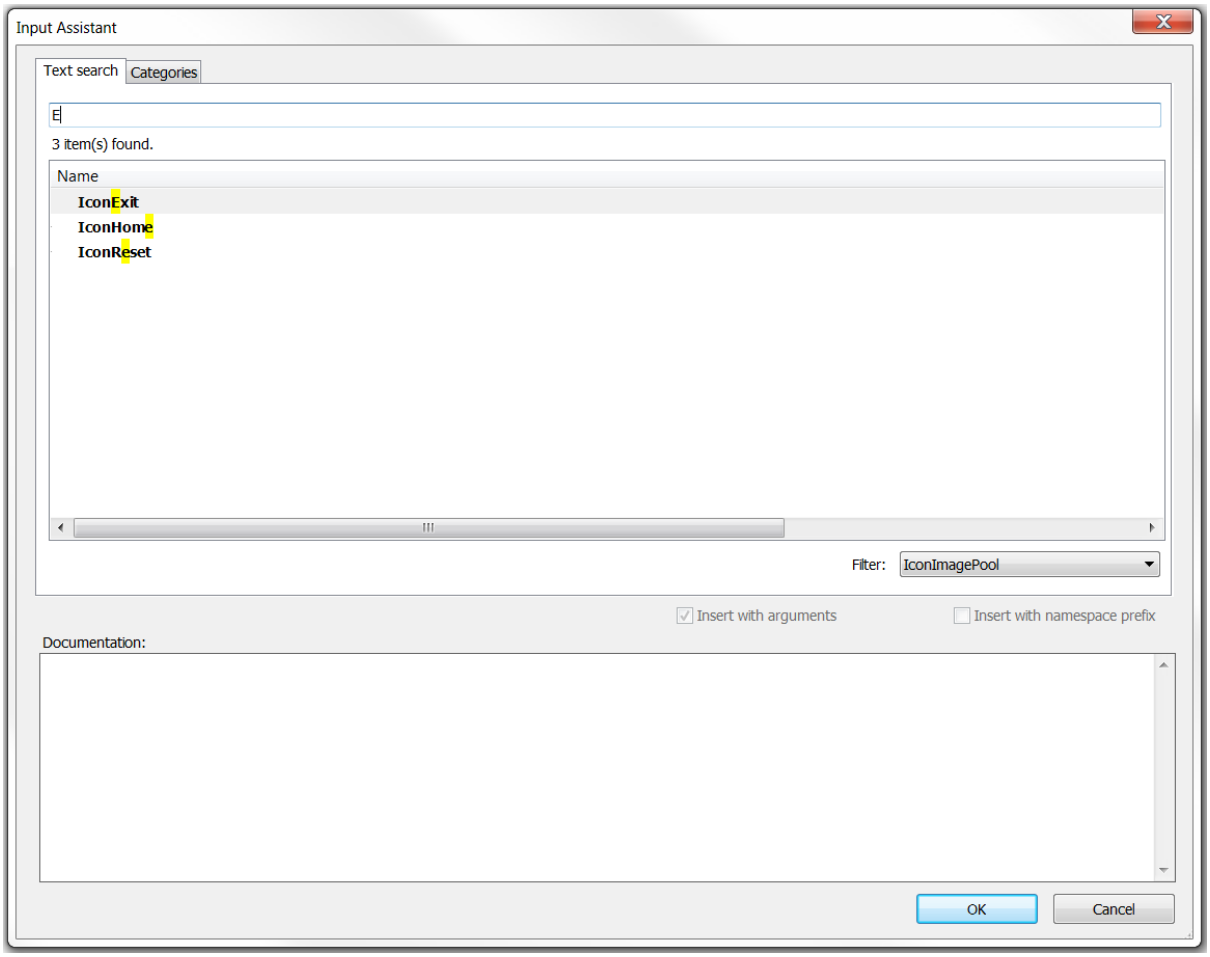
Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3.5.1 Eingabehilfe

Die Eingabehilfe für Bilder aus Bildersammlungen [▶ 153] öffnet automatisch, wenn ein Visualisierungselement vom Typ Bild [▶ 534] erstmals auf einer Visualisierungsseite eingefügt wird. Sie erreichen diesen Dialog auch, wenn Sie in den Eigenschaften eines Bildelements in das Wertefeld "Statische ID [▶ 535]" auswählen und dann auf die Schaltfläche ... klicken.

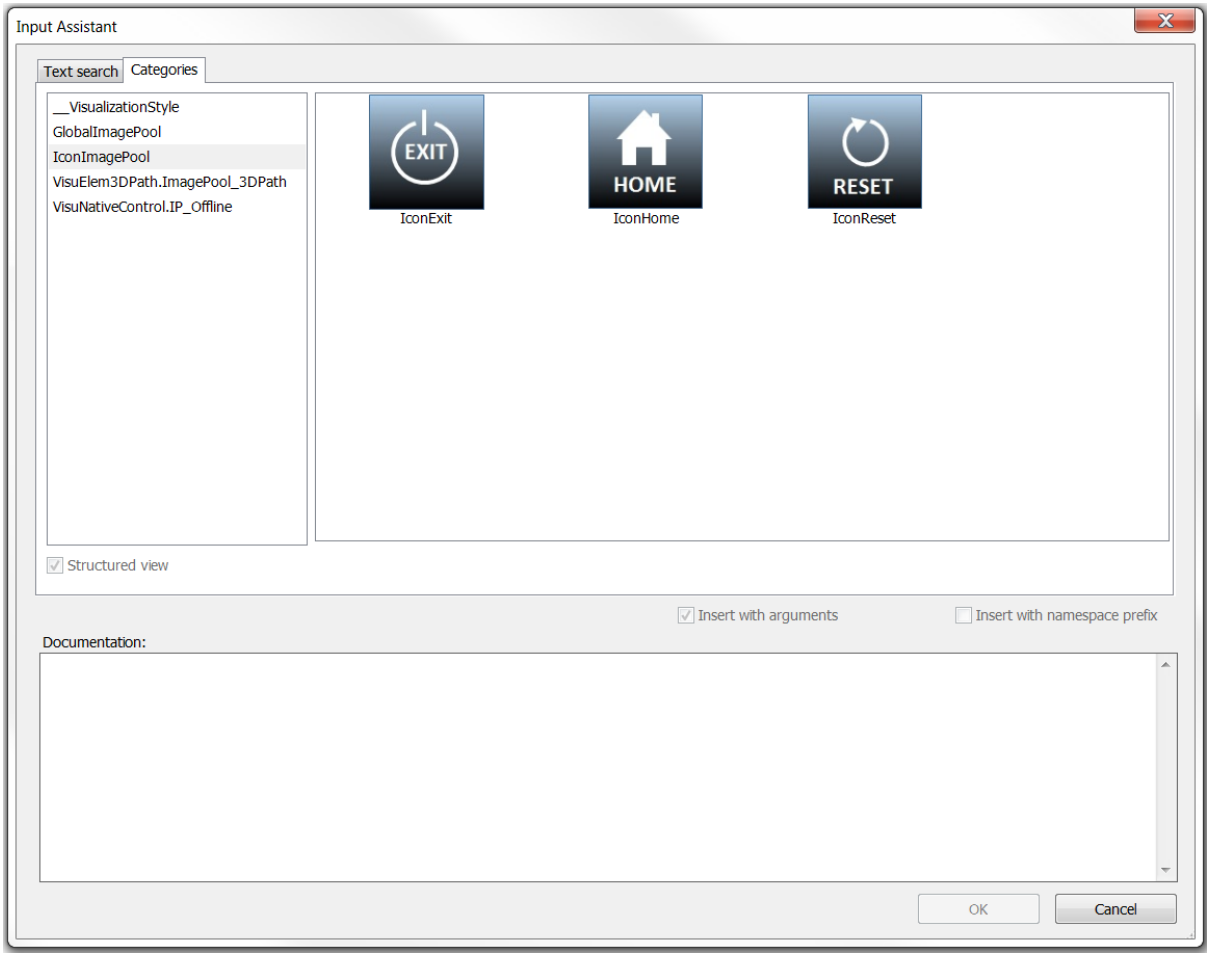
Es stehen in diesem Dialog zwei Möglichkeiten zur Verfügung, ein Bild aus den im SPS-Projekt vorhandenen Bildersammlungen auszuwählen:

- Textsuche



Geben Sie den Namen oder einen Teil des Namens der gesuchten Bilddatei in den oberen Zeileneditor ein. Es werden alle Bilddateinamen, die eine Übereinstimmung mit der Eingabe haben, angezeigt. Sie können zusätzlich einen Filter setzen, um ausschließlich in einer Bildersammlung zu suchen.

- Kategorien



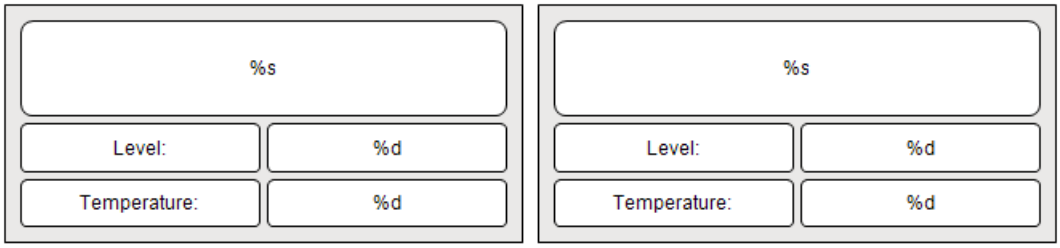
In der linken Spalte sind alle Bildersammlungen des Projekts aufgelistet. Durch das Selektieren einer Sammlung werden rechts deren Bilder in der Vorschau dargestellt.

Wählen Sie auf einer der beiden Wege ein Bild aus, selektieren Sie es und beenden den Dialog mit OK. Sie können alternativ auch einen Doppelklick auf dem gewünschten Bild ausführen. Nach dem Schließen des Dialogs ist in der Eigenschaft "Statische ID [▶ 535]" das Bild mit <Name Bildersammlung>.<Name Bild> vollständig referenziert angegeben.

15.8.3.6 Frame

Ein Frame-Element wird verwendet

- Als Referenz zu einer anderen Visualisierung [▶ 644]
- Zum Anbieten einer Schnittstelle für Platzhalter [▶ 644]
- Zum Umschalten zwischen verschiedenen Visualisierungsseiten innerhalb eines Frames [▶ 644]



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Abschneiden	Wenn diese Option zusammen mit Skalierungsart "Unskaliert" aktiviert ist, wird nur der Teil der Visualisierung, der in den Rahmen passt, dargestellt.
Rahmen zeichnen	Wenn diese Option aktiviert ist, wird die referenzierte Visualisierung mit einem Rahmen angezeigt.
Skalierungsart	Hier spezifizieren, wie der Frame auf Größenänderungen der Visualisierung reagieren soll: • Isotropisch Der Frame behält seine Proportionen. Damit kann das Verhältnis von Höhe und Breite der Visualisierung unabhängig modifiziert werden. • Anisotropisch Der Frame richtet sich nach der Größe der Visualisierung, so dass Höhe und Breite der referenzierten Visualisierung unabhängig modifiziert werden können. • Unskaliert Die Originalgröße des Frames wird ungeachtet der Visualisierungsgröße beibehalten. Falls die Option ‚Abschneiden' ebenfalls aktiviert ist, wird nur der passende Teil angezeigt. • Unskaliert und scrollbar Verwenden Sie diese Option, dann wird die referenzierte Visualisierung ohne Skalierung angezeigt. Ist sie größer als der Fensterbereich des Frames, dann wird der Frame mit Scrollbalken versehen, um den angezeigten Bereich der Visualisierung verschieben zu können. Wenn Sie die Position des Scrollbalkens mit einer Variablen setzen wollen, verwenden Sie die Eigenschaften "Variable Scrollposition horizontal" oder "Variable Scrollposition vertikal".
Deaktivieren des Hintergrundzeichnens	Um die Performance der Visualisierung zu optimieren, werden die nicht-animierten Elemente des Frame-Elements als Hintergrund-Bitmap gezeichnet. Dies könnte dazu führen, dass die Elemente nicht in der erwarteten Reihenfolge dargestellt werden. Um dieses Verhalten zu vermeiden, kann die Funktion deaktiviert werden.

Scrollbalken-Einstellungen

Die Scrollbalken-Einstellungen sind nur sichtbar, wenn bei der Skalierungsart "Unskaliert und scrollbar" eingetragen ist. Es ist sehr empfehlenswert, die Variablen Client-spezifisch zu verwenden. Ändern sich die Variablen oder wird ein Scrollbalken mithilfe der Maus verschoben, dann wirkt sich die Änderung in diesem Fall nur auf den Frame des betroffenen Clients aus. Andernfalls werden alle Clients aktualisiert.

Variable Scrollposition horizontal	Variable, die die Position des horizontalen Scrollbalkens enthält.
Variable Scrollposition vertikal	Variable, die die Position des vertikalen Scrollbalkens enthält.

Clientspezifische Variable für die Scrollposition:

```
PROGRAM MAIN
VAR
    aScrollPositionsHorizontal : ARRAY[0..20] OF INT;
    aScrollPositionsVertical : ARRAY[0..20] OF INT;
END_VAR
```

Festgelegte Eigenschaften für das Frame-Element:

Variable Scrollposition horizontal	MAIN.aScrollPositionsHorizontal[CURRENTCLIENTID]
Variable Scrollposition vertikal	MAIN.aScrollPositionsVertical[CURRENTCLIENTID]

Referenzierte Visualisierungen

Hier kann der Dialog "Frame-Auswahl [▶ 555]" geöffnet werden, mit dessen Hilfe die zu referenzierenden Visualisierungsseiten ausgewählt werden können. Nach der Auswahl einer oder mehrerer Visualisierungsseiten werden diese darunter gegebenenfalls mit ihren Platzhaltern [▶ 644] aufgelistet. Bei Änderungen der Platzhalter wird automatisch der Dialog "Aktualisierung der Frameparameter [▶ 555]" für alle Instanzen geöffnet.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Zentrum

Mit Bearbeiten des Wertes wird gleichzeitig das zugehörige Element ⊕ im Visualisierungseditor [▶ 394] verschoben.

X	Horizontale Position des Drehpunktes des Elements in Pixel
Y	Vertikale Position des Drehpunktes des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbe	Auszuwählen ist eine Rahmen- und Füllfarbe für den Normalzustand. Im Fall, dass die Farbumschlagsvariable als FALSE definiert ist, ist das Element im Normalzustand.
Alarmfarbe	Auszuwählen ist eine Rahmen- und Füllfarbe, mit der das Element im Alarmzustand dargestellt wird. Dies ist der Fall, wenn die Farbumschlagsvariable als TRUE definiert ist.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehen

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Texte

Diese Eigenschaften werden für eine statische Definition der Elementbeschriftung verwendet. Jede kann eine Formatierungssequenz [▶ 649], wie zum Beispiel %s, enthalten. Im Onlinebetrieb wird die Sequenz mit dem Inhalt der Variablen definiert in "Textvariablen" ersetzt.

Text	Geben Sie einen Text ein. Er wird verwendet, um das Element zu beschriften. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert. Hinweis: Der Text kann auch direkt eingegeben werden. Wenn das Element im Visualisierungseditor selektiert ist, kann durch Drücken der Leertaste ein Eingabefeld geöffnet werden.
Tooltip	Geben Sie einen Text ein. Er wird als Tooltip des Elements verwendet und erscheint in der Visualisierung nur im Onlinebetrieb, wenn der Cursor auf einem Element platziert ist. Der Text kann eine Formatierungssequenz, wie zum Beispiel %s, enthalten. Die zugehörige Variable ist in "Textvariablen" definiert.

Texteigenschaften

Diese Eigenschaften werden für eine statische Definition der Schriftart verwendet. Eine dynamische Definition der Schrift ist in der Kategorie "Schriftartvariablen" möglich.

Horizontale Ausrichtung	Definiert die horizontale Ausrichtung des Textes durch eine Auswahl aus: • Links • Zentriert • Rechts
Vertikale Ausrichtung	Definiert die vertikale Ausrichtung des Textes durch eine Auswahl aus: • Oben • Zentriert • Unten
Textformat	Definiert die Darstellung eines Textes, der zu lang ist, um ihn vollständig im Element anzeigen zu können: • Default – Der Text ragt aus dem Element heraus. • Zeilenschaltung – Der Text wird automatisch umgebrochen. • Auslassungspunkte – Der Text wird soweit angezeigt, wie es möglich ist und dann mit "…" abgekürzt.
Schriftart	Definiert die Schriftart aus einer Auswahl aus vordefinierten Schriftarten: • Standard • Überschrift • Große Überschrift • Title • Anmerkung Über Schaltfläche … öffnet sich der Dialog für benutzerdefinierte Eigenschaften der Schriftart.
Farbe Schriftart	Definiert die Schriftartfarbe für das Element. Entweder aus der Auswahlliste oder durch Setzen im sich öffnenden Dialog, wenn … geklickt wird.

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)
Rotation	Die hier eingetragene Integer-Variable definiert den Winkel (Winkelgrade), um den das Element um den Rotationspunkt rotiert. positive Werte = mathematisch positiv = Uhrzeigersinn Hinweis: Das Element selbst dreht sich, im Gegensatz zum Verhalten bei "innere Rotation" (s.u.), nicht. Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann mit gedrückt gehaltener Maustaste verschoben werden.
Skalierung	Die hier eingetragene Integer-Variable definiert den aktuellen Skalierungsfaktor (Prozent). Die Elementgröße wird linear entsprechend dieses Wertes verändert. Implizit wird der Wert durch 1000 dividiert, so dass es nicht nötig ist, REAL-Variablen zu verwenden um das Element zu verkleinern. Die Skalierung bezieht sich immer auf den Rotationspunkt (Zentrum). Der Rotationspunkt ⊕ wird bei einem Mausklick auf das Element sichtbar. Er kann gedrückt gehaltener Maustaste verschoben werden.

Relative Bewegung

Das Element kann relativ zu seiner fixen Position verschoben werden. Die obere-linke, und untere-rechte Kante des Elements werden um den durch eine Integer-Variable definierten Wert (Pixel) in X- bzw. Y-Richtung verschoben. Im Gegensatz zur absoluten Bewegung wird eine relative Position definiert, d.h. der Abstand zur ursprünglichen Position. Das Element kann dadurch in der Form verändert werden. Positive Werte verschieben die horizontalen Kanten nach unten bzw. die vertikalen nach rechts.

Bewegung links-oben • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die linke obere Ecke in Y-Richtung verschoben wird.
Bewegung rechts-unten • X • Y	• X: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in X-Richtung verschoben wird. • Y: Integer-Variable, deren Wert die Anzahl der Pixel angibt, um die die rechte untere Ecke in Y-Richtung verschoben wird.

Textvariablen

Sie können Text, der in einer Variablen gespeichert ist, anzeigen. Fügen Sie dazu zuerst eine Formatierungssequenz in dem Text hinzu, der unter der Eigenschaft "Texte" definiert ist. Weisen Sie dann hier eine Variable hinzu. Im Onlinebetrieb wird die Formatierungssequenz [▶ 649] durch den Inhalt der Variablen ersetzt.

Beispiel %f:

Geben Sie "Ergebnis: %2.5f" in der Eigenschaft "Texte" ein. Geben Sie "fValue" in der Eigenschaft "Textvariablen" ein. Es muss eine definierte IEC-Variable sein. Dann wird im Onlinebetrieb das Element mit "Ergebnis: 12.12345" beschriftet, wenn fValue = 12.1234567 ist.

Textvariable	Angabe einer Variablen (Typ ist Standarddatentyp), die die Information enthält, die sie anzeigen wollen. Der Typ muss konform mit der Formatierungssequenz in der Einstellung "Texte" sein.
Tooltipvariable	Angabe einer Variablen vom Typ String, die den Tooltip-Text enthält, der angezeigt werden soll. Die Eingabe in der Eigenschaft "Texte" muss eine Formatierungssequenz enthalten.

Dynamische Texte

Diese Parameter dienen der Definition von dynamischen Texten, die aus Textlisten [▶ 145] stammen. Dies erlaubt unter anderem eine Sprachumschaltung [▶ 648].

Als weitere Möglichkeit einer dynamischen Textdefinition kann der Text über eine String-Variable geliefert werden. (Siehe Kategorie "Textvariablen")

Textliste	Name der Textliste, wie im Projektbaum verwendet, als String Beispiel: 'TL_ErrorList'
Textindex	Index (ID) des Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegebene werden.
Tooltipindex	Index (ID) des Tooltip-Textes, wie in der Textliste definiert, als String. Er kann direkt statisch oder als String-Variable angegeben werden.

Schriftartvariablen

Diese Variablen werden bei dynamischen Schriftdefinitionen des Elementtextes über Projektvariablen verwendet. Statische Definitionen werden unter "Texteigenschaften" konfiguriert.

Schriftname	Angabe einer Variablen vom Typ String, die den Schriftnamen enthält, der für die Beschriftung des Elements verwendet werden soll. (Namensangabe wie im Standard-Schrifttypen-Dialog) Beispiel: MAIN.sFont (sFont := ‘Arial’;)
Größe	Angabe einer Variablen vom Typ INT, die die Größe des Elementtextes in Pixel enthält, so wie im Standarddialog ‚Schriftart‘. Beispiel: MAIN.nHeight (nHeight := 16;)
Flags	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Font-Darstellung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 1: kursiv • 2: fett • 4: unterstrichen • 8: gelöscht Beispiel: MAIN.nFlag (nFlag := 6;) Der Text wird fett und unterstrichen dargestellt.
Zeichensatz	Der Zeichensatz, der für den Font verwendet werden soll, kann über die Standard-Zeichensatz-Nummer definiert werden. Mit einer DWORD-Variablen kann diese Nummer angegeben werden (siehe auch die Definition im Standard-Font-Dialog)
Farbe	Angabe einer Variablen vom Typ DWORD, die die Farbe des Element-Textes definiert.
Flags für die Textausrichtung	Angabe einer Variablen vom Typ DWORD, die über die unten genannten Flag-Werte die Textausrichtung festlegt. Eine kombinierte Definition kann erreicht werden, indem die jeweiligen Flag-Werte addiert werden und die Summe angegeben wird. • 0: oben links • 1: horizontal zentriert • 2: rechts • 4: vertikal zentriert • 8: unten **Beispiel** MAIN.nFlag (nFlag := 5;) Der Text wird horizontal und vertikal zentriert dargestellt.

Farbvariablen

Die Farbvariablen werden für eine dynamische Definition der Elementfarben mittels Projektvariablen vom Typ DWORD angewandt. Eine Farbe wird dabei durch eine Hexadezimalzahl definiert, die sich aus Rot, Grün und Blau (RGB) Anteilen zusammensetzt. Zusätzlich wird mithilfe der Variablen die Transparenz der Farbe (FF: voll deckend - 00: voll transparent) festgelegt. Das DWORD ist wie folgt aufgebaut: 16#TTRRGGBB

Der Aufbau der Hexadezimalzahl unterscheidet sich zu TwinCAT 2. In TwinCAT 3 ist es möglich zusätzlich zu den RGB Anteilen auch die Transparenz der Farbe mit der Hexadezimalzahl zu definieren. Die Transparenz wird durch die ersten beiden Ziffern nach "16#" dargestellt. Farben, dessen Definition mit "16#00" beginnt, sind nicht sichtbar, weil sie voll transparent dargestellt werden.

Beispiel:

nFillColor := 16#FF8FE03F;

- FF: Transparenz (voll deckend)
- 8F: rot
- E0: grün
- 3F: blau

Farbumschlag	Angabe einer booleschen Variablen, die das Umschalten der Elementfarbe zwischen "Normalzustand" (Variable = FALSE) und "Alarmzustand" (Variable = TRUE) steuert.
Normalzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in der Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" FALSE ist.
Alarmzustand	Angabe einer Variablen vom Typ DWORD, die die Farbe des Elements im Alarmzustand bestimmt. Sie überschreibt den Wert, der aktuell in "Farben" definiert ist. Der Wert in den Projektvariablen wird verwendet, wenn die Variable definiert in "Farbumschlag" TRUE ist.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehensvariablen

Anzuwenden bei einer dynamischen Definition des Aussehens der Kontur und der Füllung des Elements. Statische Definition werden in "Aussehen" festgelegt.

Linienstärke	Angabe einer Variablen vom Typ Integer, die die Linienstärke des Elements in Pixel definiert. Das überschreibt den fixen Wert, der in "Aussehen" festgelegt wurde.
Füllart	Angabe einer Variablen vom Typ DWORD, die die Füllung des Elements definiert. Die Farbe, die bei Farbvariablen angegeben ist, kann angezeigt oder ignoriert werden: • Variablenwert = 0: gefüllt • Variablenwert > 0: unsichtbar, das heißt keine Füllung ist sichtbar
Linienart	Angabe einer Variablen vom Typ DWORD, die die Kontur der Außenlinie festlegt. Die folgenden Werte entsprechen folgenden Linienarten: • 0: Durchgezogen • 1: Strich • 2: Punkt • 3: Strich Punkt • 4: Strich Punkt Punkt • 8: Unsichtbar. Das heißt, das Element wird ohne Außenlinie dargestellt.

Umschaltvariable

Mit dieser Eigenschaft können Visualisierungen eines Frames umgeschaltet werden.

Variable	Integer-Variable, deren Wert die ID der anzuzeigenden Visualisierung enthält. Die ID einer Visualisierung wird durch deren Reihenfolge in der Liste der zugewiesenen Visualisierungen in der Frame-Auswahl [▶ 649] bestimmt. Zum Beispiel ergibt der erste Eintrag in dieser Liste die ID 0 und der zweite Eintrag die ID 1. Die Visualisierungen, welche einem Frame zugewiesen sind, können mit einer Variablen umgeschaltet werden. Der Wert (ID) der Visualisierung wird durch die Reihenfolge dieses Elementes in der Liste der zugewiesenen Visualisierungen im Dialog "Konfiguration der Framevisualisierungen" bestimmt. Der erste Eintrag in dieser Liste ergibt für Ganzzahlwert den Wert 0, der zweite Eintrag den Wert 1 usw.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Eingabekonfiguration

Hier kann definiert werden, welche Folgeaktionen ausgeführt werden, wenn der Benutzer im Onlinebetrieb eine Eingabe im Element vornimmt. Solange noch keine Folgeaktion definiert ist, erscheint "Konfigurieren…" im Eigenschaftenfeld. Mit einem Klick auf "Konfigurieren…" wird der Dialog Eingabekonfiguration [▶ 425] geöffnet, der erlaubt Folgeaktionen zuzuweisen. Jede Eingabeaktion kann eine beliebige zugeordnete Anzahl an Folgeaktionen besitzen.

Es stehen die folgenden Eingabeereignisse für ein Element zur Verfügung:

- OnDialogClosed
- OnMouseClick
- OnMouseDown
- OnMouseEnter
- OnMouseLeave
- OnMouseMove
- OnMouseUp

OnDialogClosed	Dieses Ereignis tritt ein, wenn innerhalb der Visualisierung eines der für Benutzereingaben vorher geöffneten Dialogfenster geschlossen wird. Hinweis: Diese Eigenschaft ist nicht auf das Element, für das es konfiguriert ist, beschränkt, sondern innerhalb der gesamten Visualisierung gültig. Es reagiert also auf jede Dialog-Schließen-Aktion. Momentan gibt es noch keine Möglichkeit, eine solche Eigenschaft für die gesamte Visualisierung zu definieren und daher muss sie einem ihrer Elemente zugewiesen werden.
OnMouseClick	Dieses Mausereignis tritt ein, wenn der Cursor auf ein Element zeigt und ein vollständiger Mausklick (drücken und freigeben der Maustaste) auf diesem Element ausgeführt wird.
OnMouseDown	Dieses Mausereignis tritt ein, wenn die Maustaste gedrückt wird, während der Cursor auf ein Element zeigt. Es ist nicht relevant, wo auf der Visualisierung die Maustaste wieder freigegeben wird.
OnMouseEnter	Dieses Mausereignis tritt ein, wenn der Cursor auf das Element gezogen wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseLeave	Dieses Mausereignis tritt ein, wenn der Cursor das Element verlässt. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseMove	Dieses Mausereignis tritt ein, wenn der Cursor innerhalb des Elements bewegt wird. Es ist nicht relevant, ob die Maustaste gedrückt oder freigegeben ist.
OnMouseUp	Dieses Mausereignis tritt ein, wenn die Maustaste auf dem Element losgelassen wird. Das Drücken der Maustaste geschah zuvor außerhalb des Elements.

Eingabekonfiguration – Tastaturkürzel

Mit einem Tastenkürzel kann eine Taste, auch mit Zusatztasten, definiert und mit einer bestimmten Nachfolgeaktion (z.B. MouseDown, MouseUp) verknüpft werden, die bei einem Ereignis der Taste (KeyDown, KeyUp) ausgeführt werden soll. Standardmäßig wird bei KeyDown (Taste drücken) die MouseDown-Aktion ausgeführt und bei KeyUp (Taste freigeben) die MouseUp-Aktion. Das kann nützlich sein, wenn eine Visualisierung sowohl über Maus- als auch Tastatureingaben bedient werden soll, weil die Eingabeaktionen dann nur einmal konfiguriert werden müssen. Diese Tastenkonfiguration für ein Element wird auch in den Tastaturkonfigurationen [▶ 411] der Visualisierung verwaltet. Änderungen werden immer zwischen diesem und dem Elementeigenschafteneditor synchronisiert.

Taste	Zuweisen einer Taste. Eine Auswahlliste stellt alle aktuell unterstützten Tasten bereit wie zum Beispiel M.
Ereignis(se)	Definition des Ereignisses, das ausgeführt werden soll, wenn die Taste beziehungsweise Taste und Modifizierer verwendet werden. Mögliche Werte, die in einer Auswahlliste bereitstehen: • Keine Aktion • MouseDown-Aktion, wenn die Taste gedrückt wird • MouseUp-Aktion, wenn die Taste freigegeben wird • MouseDown/MouseUp-Aktion, wenn die Taste gedrückt/freigegeben wird
Umschalten	Wenn diese Option aktiviert wird, muss die Taste mit der Umschalt-Taste kombiniert verwendet werden.
Steuerung	Wenn diese Option aktiviert wird, muss die Taste mit der Strg-Taste kombiniert verwendet werden.
Alt	Wenn diese Option aktiviert wird, muss die Taste mit der Alt-Taste kombiniert verwendet werden.

Eingabekonfiguration – Tasten

Mit "Tasten" kann konfiguriert werden, dass beim Ereignis "Tasten" der Wert einer booleschen Projektvariablen abhängig vom Mausverhalten gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert TRUE ist, wenn die Maustaste gedrückt wird, während der Cursor auf das Element zeigt. Ihr Wert wird wieder FALSE, sobald die Maustaste freigegeben wird oder der Cursor das Element verlässt.
FALSE Tasten	Wenn diese Option aktiviert ist, wird das oben beschrieben Tastenverhalten für die eingetragene Variable umgekehrt. Das heißt, wenn die Maustaste gedrückt wird, wird der Variablenwert auf FALSE gesetzt. Wenn die Taste wieder freigegeben wird, wird der Variablenwert auf TRUE gesetzt.
Beim Betreten tasten, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variablen' beschreiben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen, wird die Variable FALSE gesetzt. Sie wird allerdings automatisch wieder TRUE gesetzt, wenn in den Elementbereich zurückgekehrt wird, ohne die Maus zwischendurch wieder freizugeben. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus "gefangen" ist, auch wenn der Elementbereich verlassen wird.

Eingabekonfiguration – Umschalten

Mit "Umschalten" kann konfiguriert werden, dass bei diesem Ereignis der Wert einer booleschen Projektvariable abhängig vom Mausverhalt gesetzt wird.

Variable	Angabe einer booleschen Variablen, deren Wert bei jedem Mausklick auf das Element zwischen TRUE und FALSE umgeschaltet wird. Wird der Cursor bei gedrückter Maustaste aus dem Element herausbewegt, dann erfolgt kein Umschalten. Damit kann die begonnene Umschalteingabe abgebrochen werden.
Umschalten beim Loslassen, falls Maus gefangen	Ist diese Option aktiviert, verhält sich der Variablenwert wie unter ‚Variable' beschrieben, solange sich der Cursor innerhalb des Elementbereichs befindet. Wird bei gedrückter Maustaste der Elementbereich verlassen und dann die Maustaste losgelassen, wird die Variable trotzdem umgeschaltet. Es wird also berücksichtigt, dass, solange die Maustaste gedrückt ist, die Maus „gefangen" ist, auch wenn der Elementbereich verlassen wird.

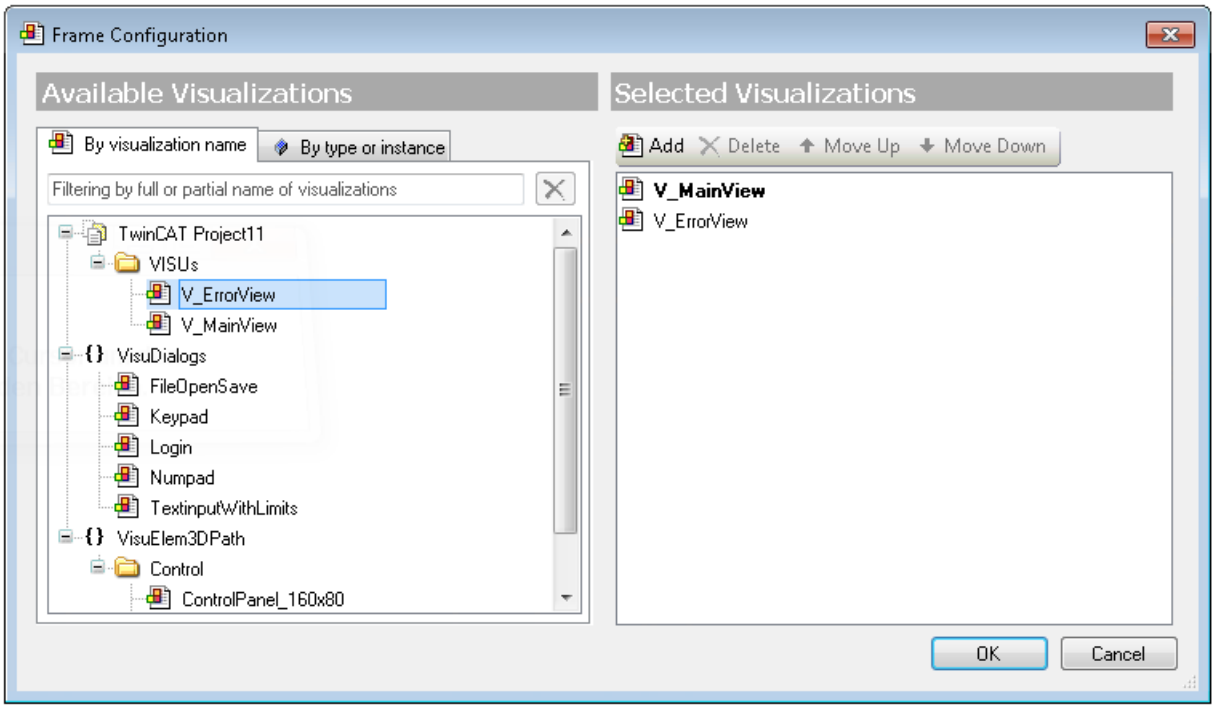
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.3.6.1 Frame-Auswahl

Dieser Dialog wird dazu verwendet, um ein Frame [▶ 544]-Element zu konfigurieren. Mit seiner Hilfe können eine oder mehrere Visualisierungsseiten [▶ 422], die über das Frame-Element referenziert werden sollen, ausgewählt werden.

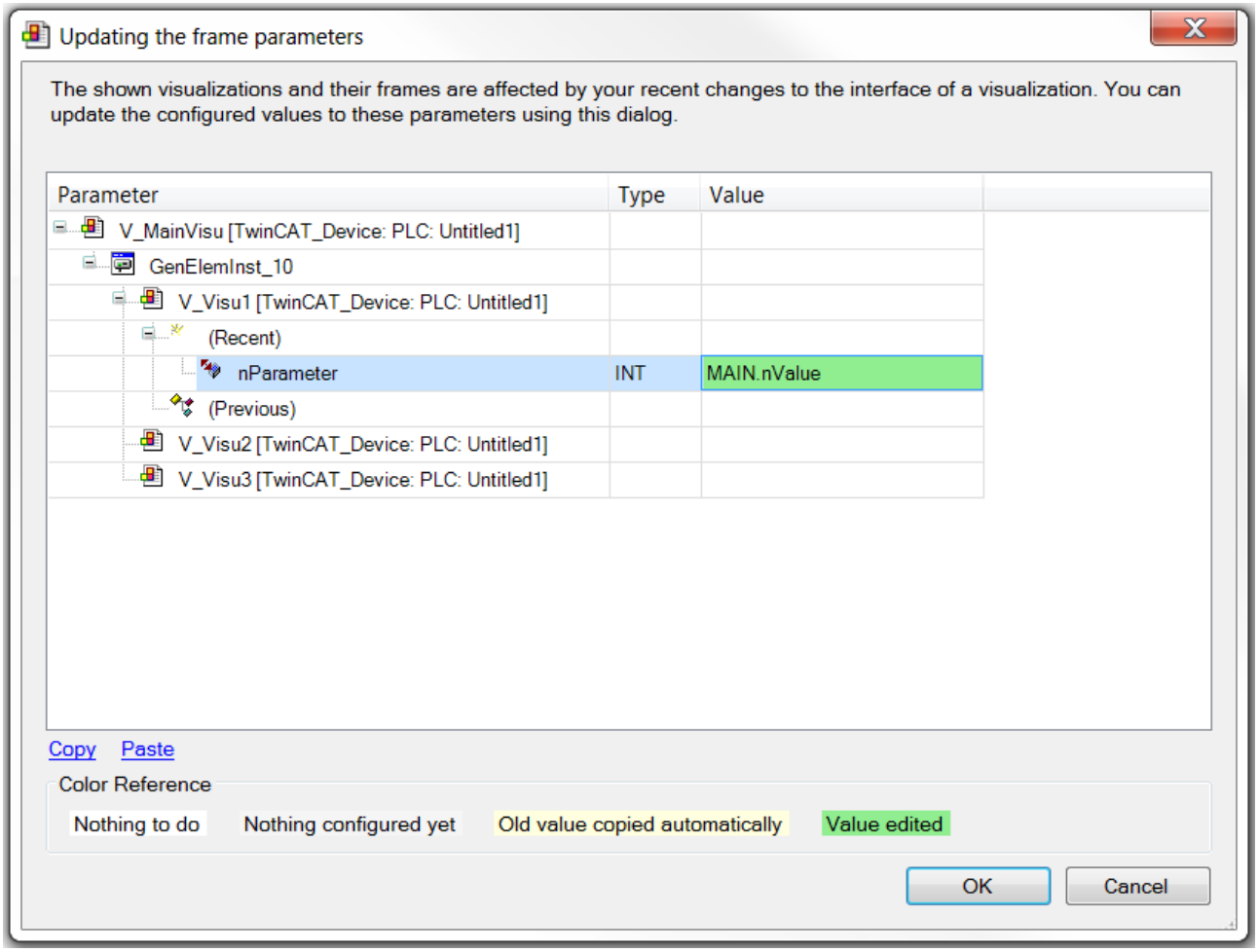


Auf der linken Seite des Dialogs können Sie die im Projekt verfügbaren Visualisierungsseiten bzw. -objekte sehen. Wählen Sie diejenigen aus, die im Frame referenziert werden sollen. Durch einen Doppelklick beziehungsweise über die Schaltfläche "Hinzufügen" können sie zur Liste der ausgewählten Visualisierungsseiten auf der rechten Seite des Dialogs hinzugefügt werden. Ausgewählte Visualisierungen können durch einen Doppelklick beziehungsweise durch die Schaltfläche "Löschen" aus der Liste entfernt werden. Eine Mehrfachauswahl von Visualisierungsseiten ist sowohl beim Hinzufügen als auch beim Löschen möglich. Die Reihenfolge innerhalb der Liste kann durch die Schaltflächen "Nach oben" und "Nach unten" verändert werden.

Die Reihenfolge der ausgewählten Visualisierungsobjekte von oben nach unten ist bestimmt durch automatisch erzeugte implizite Indexnummern der Visualisierungen. Die oberste erhält die „0“, die folgenden „1“, „2“, usw. Die Indexnummern werden für die Konfiguration der Umschaltfunktion [▶ 433] eines anderen Elements benötigt. Initial wird die Visualisierungsseite mit dem Index „0“ dargestellt.

15.8.3.6.2 Aktualisierung der Frameparameter

Der Dialog wird sichtbar, wenn eine Änderung der Parameter im Schnittstellen-Editor [▶ 395] einer Visualisierungsseite [▶ 422] vorgenommen wird, die referenziert worden ist. Mit seiner Hilfe können Frame-Parameter [▶ 398] hinzugefügt oder aktualisiert werden.



Wie der Dialog zu verwenden ist, ist im Schnittstellen-Editor [▶ 395] beschrieben.

15.8.4 Lampen/ Schalter/ Bilder

15.8.4.1 Bildwechsler

Das Element Bildwechsler stellt ein Bild aus drei referenzierten Bildern [▶ 557] dar. Wenn zur Laufzeit der Visualisierung eine Mausaktion auftritt, wechselt das dargestellte Bild. Die Folgen von Mausaktionen sind über das Elementverhalten einstellbar. Weitere Informationen zu den Integrationsmöglichkeiten von Bildern in der Visualisierung finden Sie im Abschnitt "Bilder [▶ 652]" in den "Anwendungstipps".

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.

- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Zuweisung einer booleschen Variablen, deren Zustand entsprechend der Benutzereingabe wechselt. Abhängig von Elementverhalten ist die Variable TRUE, solange die Maustaste gedrückt ist (Bildtaster), oder der Wert ändert sich mit jedem Mausklick (Bildumschalter).

Bildeinstellungen

Um die Einträge für die Bildernamen einheitlich zu halten, sollte der Name der Bildersammlung als Präfix angegeben werden. Dies ist nicht notwendig, wenn die Bilder im GlobalImagePool verwaltet werden, da diese sowieso als erste durchsucht wird.

Bild an	ID einer Bitmap aus einer Bildersammlung. Das Bild wird im An-Zustand angezeigt.
Bild aus	ID einer Bitmap aus einer Bildersammlung. Das Bild wird im Aus-Zustand angezeigt.
Bild gedrückt	ID einer Bitmap aus einer Bildersammlung. Die Visualisierung zeigt zur Laufzeit das referenzierte Bild an, sobald das Element die Mausaktion "Rechte Maustaste gedrückt" empfängt. Voraussetzung: Elementverhalten ist Bildumschalter.
Transparenz	Wenn diese Option aktiviert ist, ist das Bild für einen Teilbereich voll-transparent gezeichnet, der durch die Transparenzfarbe definiert wird.
Transparenzfarbe	Hier kann die Farbe eingestellt werden, die im Bild voll-transparent dargestellt ist. Beispiel: Wenn der Hintergrund des Bildes weiß ist und auch die Transparenzfarbe weiß ist, dann wird der Hintergrund voll-transparent gezeichnet. Voraussetzung: Transparenz ist aktiviert.
Skalierungsart	Hier ist zu spezifizieren, wie das Element auf die Änderung der Framegröße zu reagieren hat: • Isotropisch: Höhe und Breite des Bildes werden mit dem Frame verändert. Ihre ursprünglichen Proportionen werden aber beibehalten. • Anisotropisch: Das Bild füllt, unabhängig von seinen Proportionen, den ganzen Frame aus. • Unskaliert: Die Bildgröße ist festgelegt und wird nicht verändert, wenn der Frame seine Größe ändert.
Horizontale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschafteneditor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die horizontale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Links • Zentriert • Rechts
Vertikale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschaften-Editor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die vertikale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Oben • Zentriert • Unten

Elementverhalten	Hier kann das Elementverhalten ausgewählt werden: • Bildumschalter: Der Zustand des Elements und damit Bild und Variable toggeln mit jedem Mausklick. • Bildtaster: ‚Bild an' wird angezeigt und die Variable ist TRUE, solange die Maustaste gedrückt gehalten wird. In diesem Zustand wird das Bild, das in "Bild gedrückt" referenziert ist, nicht verwendet.
FALSE Tasten	Wenn diese Option aktiviert ist, wird die Variable durch das Drücken der Maustaste auf FALSE gesetzt. Andernfalls wird die Variable auf TRUE gesetzt. Voraussetzung: In Elementverhalten ist Bildwechsler eingestellt.

Texte

Tooltip	Hier kann der Text eingestellt werden, der als Tooltip des Elements verwendet werden soll. Er erscheint in der Visualisierung nur zur Laufzeit.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.4.2 Lampe

Die Lampe leuchtet auf, wenn die zugewiesene Variable gesetzt wird. Die Farbe der Lampe kann in ihren Einstellungen [▶ 561] geändert werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche … die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Boolesche Variable, deren Wert den Zustand der Lampe schaltet. Bei TRUE wird die Lampe eingeschaltet und leuchtend dargestellt.

Bildeinstellung

Skalierungsart	Hier ist zu spezifizieren, wie das Element auf die Änderung der Framegröße zu reagieren hat: • Isotropisch: Höhe und Breite des Bildes werden mit dem Frame verändert. Ihre ursprünglichen Proportionen werden aber beibehalten. • Anisotropisch: Das Bild füllt, unabhängig von seinen Proportionen, den ganzen Frame aus. • Unskaliert: Die Bildgröße ist festgelegt und wird nicht verändert, wenn der Frame seine Größe ändert.
Horizontale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschafteneditor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die horizontale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Links • Zentriert • Rechts
Vertikale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschaften-Editor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die vertikale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Oben • Zentriert • Unten

Texte

Tooltip	Hier kann der Text eingestellt werden, der als Tooltip des Elements verwendet werden soll. Er erscheint in der Visualisierung nur zur Laufzeit.

Hintergrund

Bild	Hier ist die Farbe für die Lampe auszuwählen: • Gelb • Rot • Grün • Blau • Grau

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.4.3 Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter

Der Schalter wird verwendet, um den Wert einer booleschen Variablen zu setzen.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Boolesche Variable, deren Wert entsprechend der Benutzereingabe wechselt. Abhängig vom Elementverhalten ist die Variable TRUE, solange die Maustaste gedrückt ist (Taster), oder der Wert ändert sich mit jedem Mausklick (Umschalter).

Bildeinstellung

Skalierungsart	Hier ist zu spezifizieren, wie das Element auf die Änderung der Framegröße zu reagieren hat: • Isotropisch: Höhe und Breite des Bildes werden mit dem Frame verändert. Ihre ursprünglichen Proportionen werden aber beibehalten. • Anisotropisch: Das Bild füllt, unabhängig von seinen Proportionen, den ganzen Frame aus. • Unskaliert: Die Bildgröße ist festgelegt und wird nicht verändert, wenn der Frame seine Größe ändert.
Horizontale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschafteneditor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die horizontale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Links • Zentriert • Rechts
Vertikale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschaften-Editor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die vertikale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Oben • Zentriert • Unten

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Texte

Tooltip	Hier kann der Text eingestellt werden, der als Tooltip des Elements verwendet werden soll. Er erscheint in der Visualisierung nur zur Laufzeit.

Hintergrund

Bild	Hier ist die Farbe für die Lampe auszuwählen: • Gelb • Rot • Grün • Blau • Grau

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.4.4 Drehschalter

Der Drehschalter wird verwendet, um den Wert einer booleschen Variablen zu setzen.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche … die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Variable	Boolesche Variable, deren Wert entsprechend der Benutzereingabe wechselt. Abhängig vom Elementverhalten ist die Variable TRUE, solange die Maustaste gedrückt ist (Taster), oder der Wert ändert sich mit jedem Mausklick (Umschalter).

Bildeinstellungen

Skalierungsart	Hier ist zu spezifizieren, wie das Element auf die Änderung der Framegröße zu reagieren hat: • Isotropisch: Höhe und Breite des Bildes werden mit dem Frame verändert. Ihre ursprünglichen Proportionen werden aber beibehalten. • Anisotropisch: Das Bild füllt, unabhängig von seinen Proportionen, den ganzen Frame aus. • Unskaliert: Die Bildgröße ist festgelegt und wird nicht verändert, wenn der Frame seine Größe ändert.
Horizontale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschafteneditor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die horizontale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Links • Zentriert • Rechts
Vertikale Ausrichtung	Nur verfügbar, wenn die Option "Experte" für den Eigenschaften-Editor aktiviert ist und wenn die Skalierungsart des Bildes "Isotropisch" ist. Dient dazu, die vertikale Ausrichtung zu einem anderen Element (wie in der Basis-Visualisierung definiert) explizit aufrechtzuerhalten, wenn die Visualisierung innerhalb eines skalierten Frames verwendet wird. Siehe oben: Einstellung "Isotropisch" • Oben • Zentriert • Unten

Elementverhalten	Hier kann das Elementverhalten ausgewählt werden: • Bildumschalter: Der Zustand des Elements und damit Bild und Variable toggeln mit jedem Mausklick. • Bildtaster: ‚Bild an' wird angezeigt und die Variable ist TRUE, solange die Maustaste gedrückt gehalten wird. In diesem Zustand wird das Bild, das in "Bild gedrückt" referenziert ist, nicht verwendet.
FALSE Tasten	Wenn diese Option aktiviert ist, wird die Variable durch das Drücken der Maustaste auf FALSE gesetzt. Andernfalls wird die Variable auf TRUE gesetzt. Voraussetzung: In Elementverhalten ist Bildwechsler eingestellt.
Ausrichtung	• Oben: Der Schalter ist nach oben gerichtet. • Seitlich: Der Schalter ist seitlich gerichtet.
Farbwechsel	Ist die Option nicht aktiviert, dann leuchtet der Schalter auf, wenn er auf eingeschaltet ist. Ist die Option aktiviert, dann leuchtet der Schalter nicht auf, auch wenn er auf eingeschaltet ist.

Texte

Tooltip	Hier kann der Text eingestellt werden, der als Tooltip des Elements verwendet werden soll. Er erscheint in der Visualisierung nur zur Laufzeit.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.
Eingabe deaktiviert	Angabe einer booleschen Variablen. Wenn diese TRUE liefert, bleiben Eingaben auf das Element ohne Effekt. Außerdem wird das Element selbst in der Visualisierung ausgegraut, um zu kennzeichnen, dass keine Benutzereingaben möglich sind. In dem Fall, dass die Visualisierung die Benutzerverwaltung [▶ 412] verwendet, werden die Elemente für Benutzergruppen mit Zugriffsrecht "nur sichtbar" ausgegraut.

Hintergrund

Bild	Hier ist die Farbe für die Lampe auszuwählen: • Gelb • Rot • Grün • Blau • Grau

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.5 Messgeräte

15.8.5.1 Balkenanzeige

Mit diesem Element kann eine Balkenanzeige zur Visualisierung hinzugefügt werden. Die Balkenanzeige hat ein vordefiniertes Design, bei dem die Hintergrundfarbe [▶ 568] geändert werden kann. Wahlweise kann dieses Design durch ein eigenes Hintergrundbild ersetzt werden. Die Ausrichtung der Anzeige kann von horizontal in vertikal geändert [▶ 568] werden und der Balken kann in Farbbereiche [▶ 570] unterteilt werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche … die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Variable	Numerische Variable, deren Wert als Balkenlänge dargestellt wird.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Hintergrund

Wird kein eigenes Hintergrundbild genutzt, stehen folgende Eigenschaften zur Verfügung:

Farbe Hintergrund	Auszuwählen ist eine Farbe für den Balken: • Gelb • Rot • Grün • Blau • Grau

Wird ein eigenes Hintergrundbild definiert, stehen diese Eigenschaften zur Verfügung:

Bild	Hier kann ein Bild aus einem ImagePool zugewiesen werden, indem der Name der Bilddatei oder dessen ID angegeben wird.
Transparenzfarbe	Für Bilder mit transparentem Hintergrund kann eine Farbe ausgewählt werden, die transparent dargestellt werden soll. Die Schaltfläche  öffnet den Farbenauswahldialog.

Balken

Diagrammart	Die Drop-down-Liste stellt verschiedene Möglichkeiten zur Verfügung, wie Balken und Skale platziert werden können: • Skala neben Balken • Skala im Balken • Balken in Skala • Keine Skala
Ausrichtung	Der Balken kann horizontal oder vertikal ausgerichtet sein. Die Ausrichtung entsteht aus dem Verhältnis von Breite zu Höhe und kann hier nicht editiert werden. Diese Änderung erfolgt im Visualisierungseditor [▶ 394], indem ein Eckpunkt es Elements mit der Maus "gegriffen" und horizontal oder vertikal gezogen wird.
Laufrichtung	Ist die Ausrichtung horizontal, kann gewählt werden zwischen: • Links nach rechts • Rechts nach links Ist die Ausrichtung vertikal, kann gewählt werden zwischen: • Von unten nach oben • Von oben nach unten

Skala

Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala. Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenlinienstärke	Dicke der Skalenlinie in Pixel
Skalenfarbe	Die Farbe kann über die Selektionsliste bzw. über die Schaltfläche [...] aus dem Standarddialog zur Farbauswahl eingestellt werden.
Skala in 3D	Wenn diese Option aktiviert ist, wird die Skala 3-dimensional dargestellt.
Elementrahmen	Wenn diese Option aktiviert ist, wird ein Rahmen um die Balkenanzeige gezogen.

Beschriftung

Einheit	Der eingegebene Text beschriftet das Element. Er wird unterhalb des Mittelpunkts der Skalierung angezeigt. Auf diese Weise kann beispielsweise die Skalierungseinheit angegeben werden.
Schriftart	Hier ist die Schriftart für Einheit und Skalierung festzulegen: • Standard • Überschrift • Große • Titel • Anmerkung Mit Klick auf die Schaltfläche [...] öffnet ein Dialog für benutzerdefinierte Einstellungen der Schrifteigenschaften.
Skalenformat (C-Syntax)	Benutzen Sie die C-Syntax zur Angabe der Formatierung der Skalierungsbeschriftung. So führt die Eingabe des Strings "%3.2f s" in diesem Feld zur Anzeige der Skalierungsbeschriftungen mit 3 Stellen, davon 2 Nachkommastellen, gefolgt vom Buchstaben "s".
Max. Breite der Beschriftungen	Wert, der die maximale Breite der Skalierungsbeschriftung vorgibt– In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Höhe der Beschriftungen	Wert, der die Höhe der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Farbe Schriftart	Die Farbe für die Schrift kann über Selektionsliste bzw. über Schaltfläche [...] aus dem Standarddialog zur Farbauswahl eingestellt werden.

Positionierung

Horizontale Verschiebung	Abstand in Pixel der Skala bzw. des Balkens zum horizontalen Elementrahmen
Vertikale Verschiebung	Abstand in Pixel der Skala bzw. des Balkens zum vertikalen Elementrahmen
Horizontale Skalierung	Faktor, um den die Skala bzw. der Balken in horizontaler Richtung vergrößert (negativer Wert) oder verkleinert (positiver Wert) wird
Vertikale Skalierung	Faktor, um den die Skala bzw. der Balken in vertikaler Richtung vergrößert (negativer Wert) oder verkleinert (positiver Wert) wird

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die

Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbe des Graphen	Farbe für den Balken
Balkenhintergrund	Wenn diese Option aktiviert ist, wird die Hintergrundfarbe der Balkenanzeige in schwarz geändert. Ansonsten ist sie weiß.
Rahmenfarbe	Farbe des Rahmens, der die Balkenanzeige umgibt, sofern das Kontrollkästchen Elementrahmen aktiviert ist
Farbbereiche verwenden	Wenn diese Option aktiviert ist, werden die Farbbereiche, so wie sie unter Farbbereiche definiert sind, angezeigt.
Gesamte Farbe umschalten	Wenn diese Option aktiviert ist, ändert der gesamte Balken seine Farbe, wenn der Variablenwert den Anfangs-/ Endwert der Skala unter- oder überschreitet.
Farbverlauf für Balken verwenden	Wenn diese Option aktiviert ist, wird der Balken mit Farbverlauf angezeigt.
Farbbereichsmarkierung	Hier kann ausgewählt werden, in welche Richtung die Farbbereiche zeigen. Es stehen die folgenden Einstellungen zur Verfügung: • Keine Markierung • Markierung vorwärts • Markierung rückwärts
Farbbereiche • Bereiche ◦ [<n>]	Mit einem Klick auf die Schaltfläche  wird ein neuer Farbbereich erzeugt. Für jeden Farbbereich wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf "Löschen" wird der zugehörige Farbbereich mit seinen Einstellungen gelöscht.

Bereich [<n>]

Bereichsanfang	Anfang des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Bereichsende	Ende des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Farbe	Farbe des Balkenbereichs

Farbverlauf und Transparenz werden unter Windows CE nicht unterstützt.

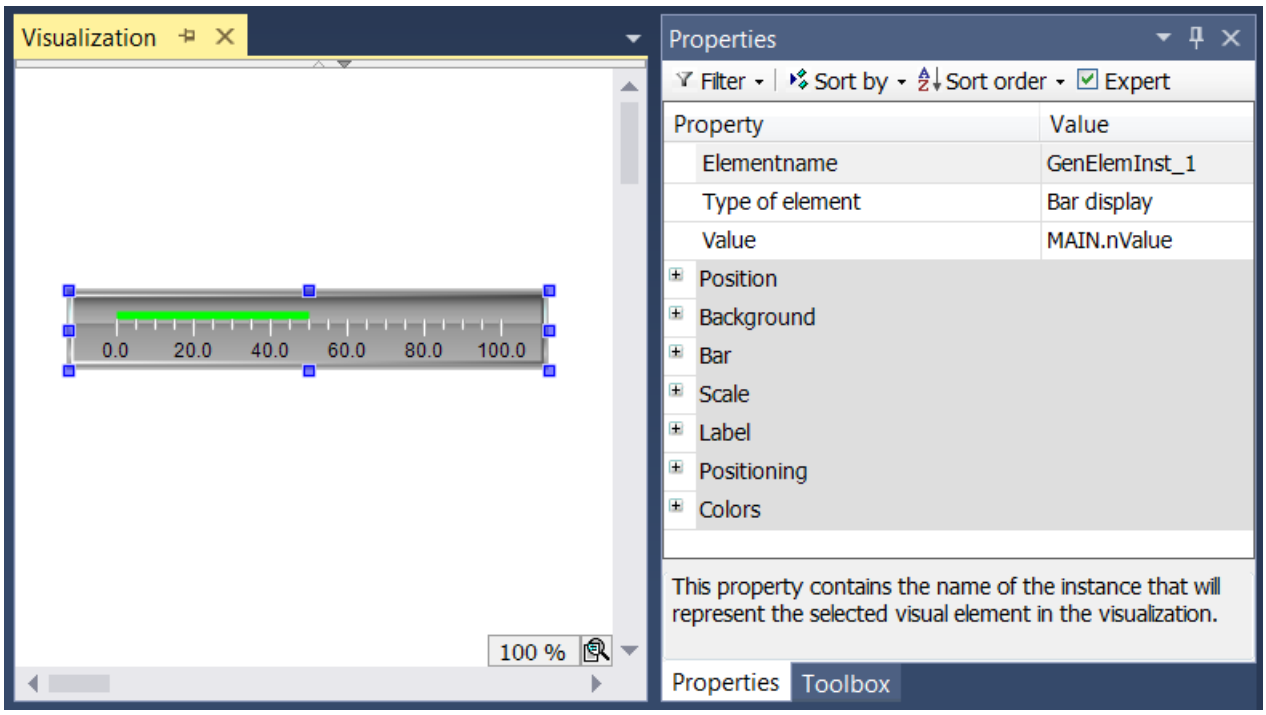
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

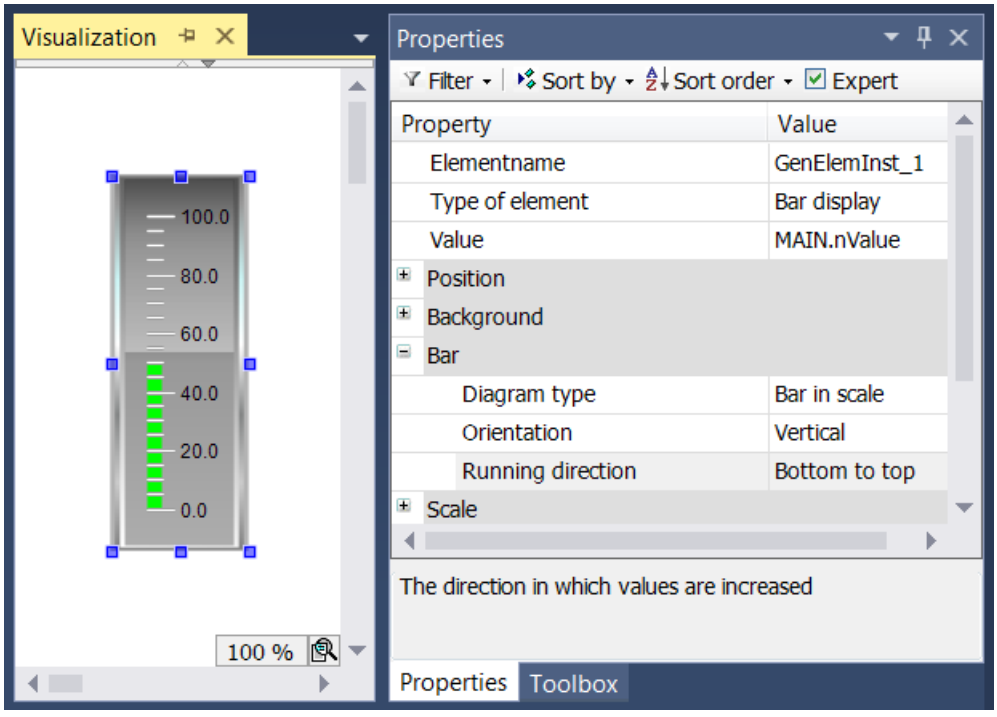
15.8.5.1.1 Konfiguration einer Balkenanzeige

Im Folgenden wird ein Beispiel für die Konfiguration eines Zeigerinstruments erläutert.



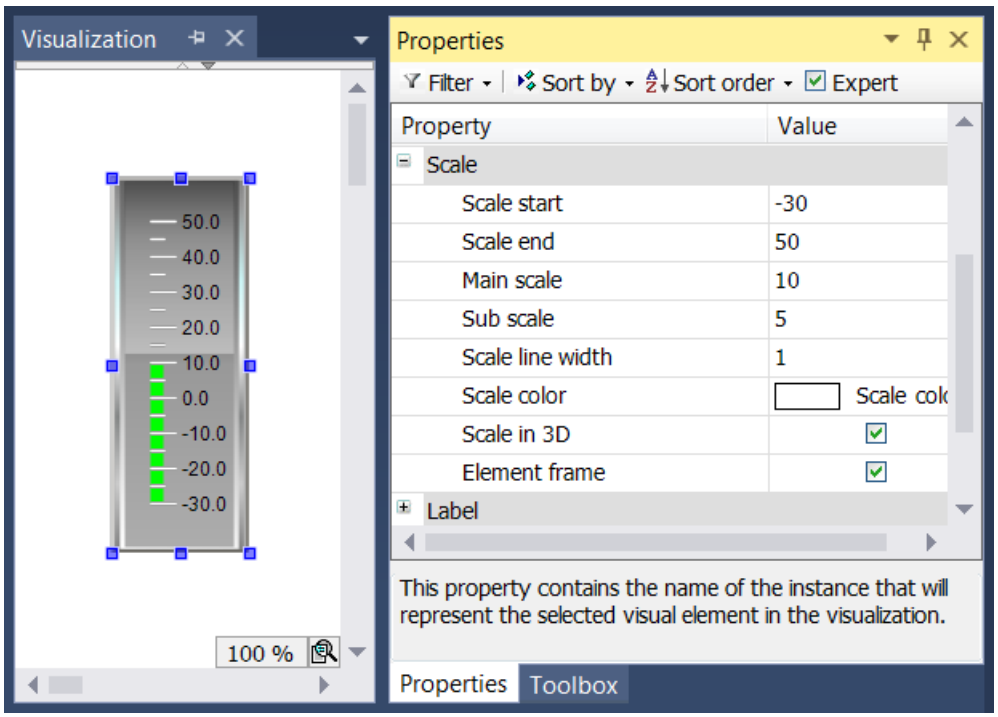
Die zu verknüpfende Eingangsvariable – im Beispiel eine Variable namens "nValue" – muss innerhalb der Elementeigenschaften der "Balkenanzeige" angegeben werden. Nach einem Klick in das Eingabefeld der Eigenschaft "Variable" steht Ihnen die Schaltfläche [...] zur Verfügung, welche benutzt werden kann, um im Projekt nach der Variable zu browsen.

Die Orientierung und Platzierung des Balkens in Bezug auf die Skalierung kann innerhalb des Abschnitts "Balken" eingestellt werden.

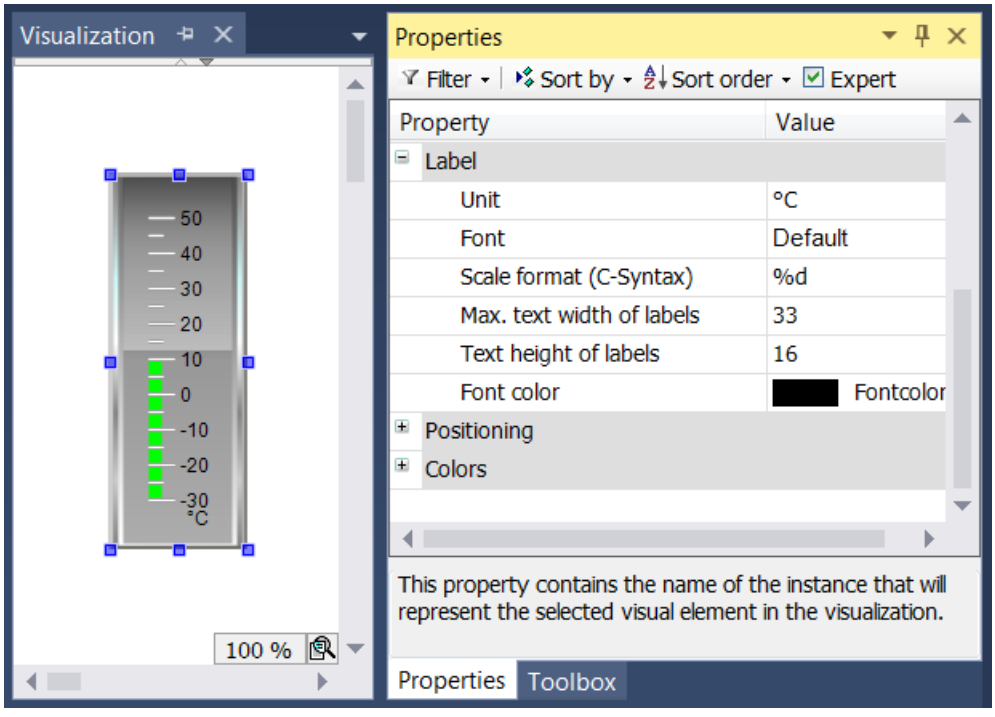


Im Beispiel wurde die Orientierung des Balkens von "Horizontal" in "Vertikal" geändert, indem das Seitenverhältnis von Breite zu Höhe geändert worden ist. Diese Einstellung ändert auch die möglichen Angaben für die "Laufrichtung". Anstelle von "Links nach rechts" und "Rechts nach links" kann nun zwischen "Unten nach oben" und "Oben nach unten" gewählt werden. Die Position des Balkens in Bezug zur Skalierung kann durch Einstellung der "Diagrammart" bestimmt werden.

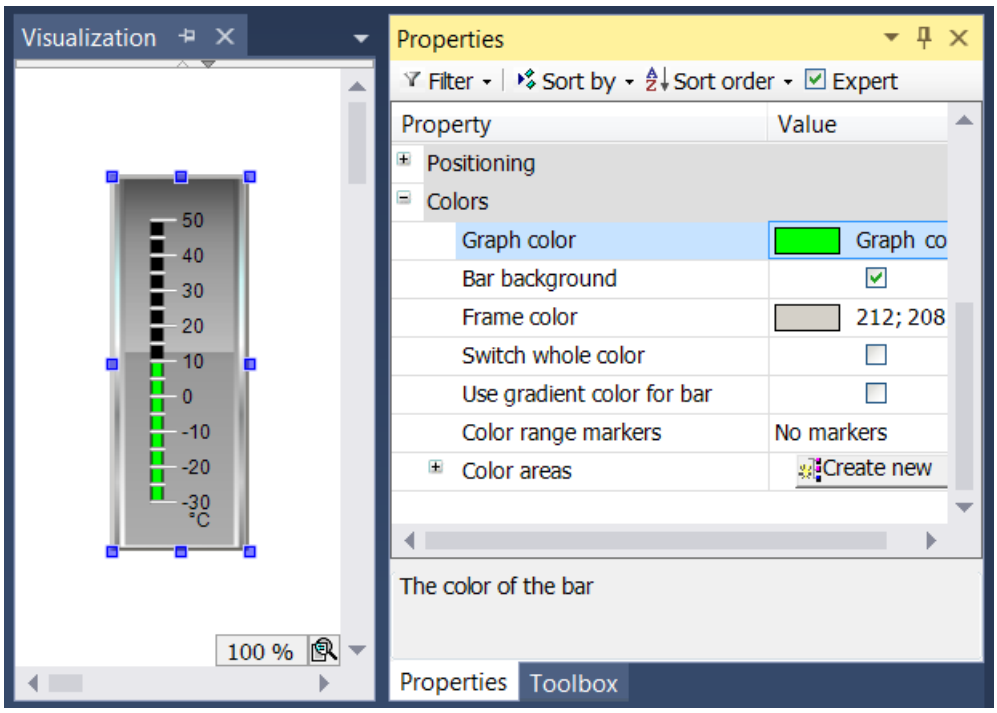
Der Abschnitt "Skala" ist zur Festlegung des Wertebereichs der Skala und der groben bzw. feineren Unterteilung der Skalierung vorgesehen.



Der Wertebereich der Skala wird von unten durch den Wert in "Skalenanfang" und von oben durch den Wert in "Skalenende" begrenzt. Der Wert von "Skalenstart" muss geringer sein als der von "Skalenende". Beide Distanzangaben in "Hauptskala" und "Unterskala" können auf 0 gesetzt werden, um eine Anzeige der Skalierungsstriche auszuschalten. Wird der Wert der Grobskalierung auf 0 gesetzt, so werden unabhängig vom Wert der Feinskalierung keinerlei Skalierungsstriche gezeichnet. Wird die Feinskalierung auf 0 gesetzt, werden nur die Skalierungsstriche der groben Unterteilung gezeichnet. Da wir im Beispiel einen Rahmen um das Element gezeichnet haben möchten, wird das Kontrollkästchen "Elementrahmen" aktiviert.



Der Abschnitt "Beschriftung" dient zur Beschriftung des Balkens. Der Eintrag "Einheit" ist zur Angabe der Einheit gedacht und wird zentriert unterhalb des Balkens angezeigt. Nach der Wahl einer geeigneten Schrift(-farbe) können Sie das Format der Beschriftung anpassen. Die Angabe der Formatierung für Zahlenwerte muss entsprechend der C-Syntax vorgenommen werden. Benutzen Sie "%d" für ganzzahlige Werte und "%.Xf" für Gleitpunktzahlen, wobei "X" durch die gewünschte Anzahl an Nachkommastellen ersetzt werden muss.



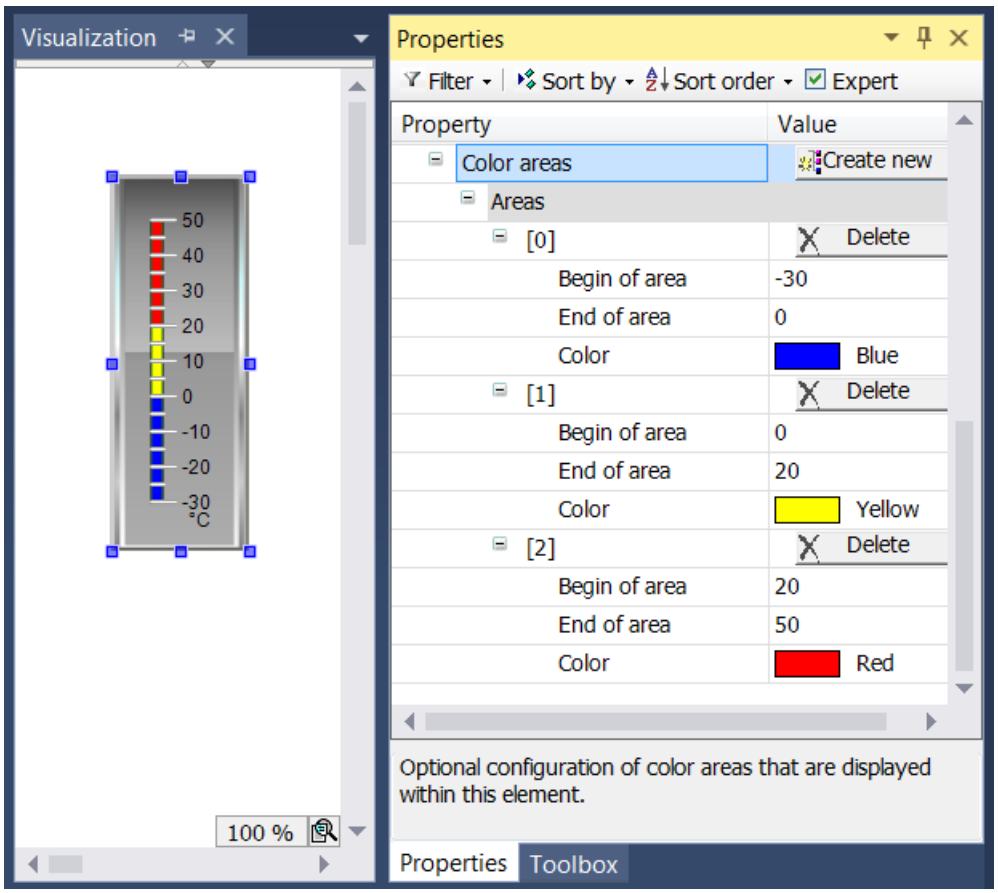
Schließlich kann die Einfärbung des Elements innerhalb des Abschnitts "Farben" gesetzt werden. Zuerst kann die Farbe des Balkens selbst bestimmt werden ("Farbe des Graphen"). Standardmäßig wird kein Balkenhintergrund, das ist der Teil der Balkenzeile, die aktuell nicht vom Balken ausgefüllt ist, gezeichnet. Bei Aktivierung des Kontrollkästchens "Balkenhintergrund" wird der nicht ausgefüllte Teil des Balkens schwarz hinterlegt.

Innerhalb des Unterbereichs "Farbbereiche" kann die Skala in Teilbereichen unterteilt werden. Jedem Teilbereich kann eine bestimmte Farbe zugewiesen werden, indem für sie mithilfe der Schaltfläche

ein Farbbereich erzeugt wird. Die Farbbereiche werden aufsteigend nummeriert. Jeder Farbbereich wird innerhalb der Elementeigenschaften mit eigenen Eingabefeldern ausgestattet.

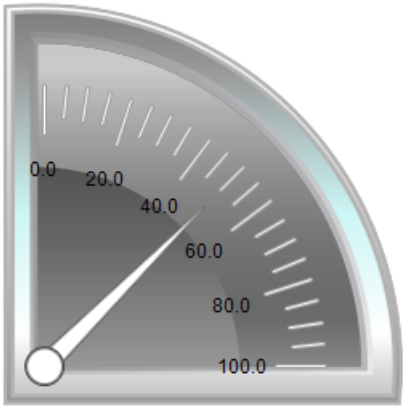
In "Bereichsanfang" und "Bereichsende" können die Grenzen des Unterbereichs festgelegt werden. Seine Einfärbung kann aus dem Aufklappmenü ausgewählt werden. Ein einmal angelegter Farbbereich kann durch einen Klick auf die entsprechende Schaltfläche

gelöscht werden.



15.8.5.2 Zeigerinstrument 90°

Mit dem Zeigerinstrument kann zum Beispiel ein Drehzahlmesser der Visualisierung hinzugefügt werden. Die Nadel positioniert sich nach dem Wert der zugewiesenen Variablen. Das Element hat ein vorkonfiguriertes Design, in dem die Hintergrundfarbe [▶ 575] gesetzt werden kann. Wahlweise kann dieses Design durch ein eigenes Hintergrundbild [▶ 575] ersetzt werden. Die Skala kann in Farbbereiche [▶ 577] unterteilt werden. Ein Beispiel für die Konfiguration ist im Abschnitt "Konfiguration eines Zeigerinstruments [▶ 578]" zu finden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Wert	Numerische Variable, deren Wert als Ausschlag der Instrumentennadel dargestellt wird.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Hintergrund

Wird kein eigenes Hintergrundbild genutzt, stehen folgende Eigenschaften zur Verfügung:

Farbe Hintergrund	Auszuwählen ist eine Farbe für den Balken: • Gelb • Rot • Grün • Blau • Grau

Wird ein eigenes Hintergrundbild definiert, stehen diese Eigenschaften zur Verfügung:

Bild	Hier kann ein Bild aus einem ImagePool zugewiesen werden, indem der Name der Bilddatei oder dessen ID angegeben wird.
Transparenzfarbe	Für Bilder mit transparentem Hintergrund kann eine Farbe ausgewählt werden, die transparent dargestellt werden soll. Die Schaltfläche  öffnet den Farbenauswahldialog.

Zeiger

Zeigertyp	Es gibt eine Auswahl an verschiedenen Zeigertypen: • Normaler Zeiger • Dünner Zeiger • Breiter Zeiger • Dünne Nadel • Dünner 3D-Zeiger • Dünne 3D-Nadel
Farbe	Farbe, mit der der Zeiger angezeigt wird
Winkelbereich	Hier kann ein 90° Ausschnitt ausgewählt werden: • Oben rechts • Oben links • Unten links • Unten rechts
Zusätzlicher Zeiger	Wenn diese Option aktiviert ist, ist ein zusätzlicher Zeiger auf der Skala gegenüber der Nadel sichtbar.

Skala

Unterskalaposition	Die Unterskala kann am äußeren oder inneren Radius des Skalenrings angezeigt werden: • Außen • Innen
Skalentyp	Die Skala kann angezeigt werden als: • Linien • Punkte • Quadrate
Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala – Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenlinienstärke	Dicke der Skalenlinie in Pixel
Skalenfarbe	Die Farbe kann über die Selektionsliste bzw. über die Schaltfläche  aus dem Standarddialog zur Farbauswahl eingestellt werden.
Skala in 3D	Wenn diese Option aktiviert ist, wird die Skala 3-dimensional dargestellt.
Skala anzeigen	Wenn diese Option aktiviert ist, wird die Skala dargestellt.
Innerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring innen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.
Äußerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring außen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.

Beschriftung

Beschriftung	Hier kann eingestellt werden, ob die Skalenwerte auf der Außenseite oder Innenseite der Skala platziert werden sollen.
Einheit	Der eingegebene Text beschriftet das Element. Er wird unterhalb des Mittelpunkts der Skalierung angezeigt. Auf diese Weise kann beispielsweise die Skalierungseinheit angegeben werden.
Schriftart	Hier ist die Schriftart für Einheit und Skalierung festzulegen: • Standard • Überschrift • Große • Titel • Anmerkung Mit Klick auf die Schaltfläche … öffnet ein Dialog für benutzerdefinierte Einstellungen der Schrifteigenschaften.
Skalenformat (C-Syntax)	Benutzen Sie die C-Syntax zur Angabe der Formatierung der Skalierungsbeschriftung. So führt die Eingabe des Strings "%3.2f s" in diesem Feld zur Anzeige der Skalierungsbeschriftungen mit 3 Stellen, davon 2 Nachkommastellen, gefolgt vom Buchstaben "s".
Max. Breite der Beschriftungen	Wert, der die maximale Breite der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Höhe der Beschriftungen	Wert, der die Höhe der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Farbe Schriftart	Die Farbe für die Schrift kann über Selektionsliste bzw. über Schaltfläche … aus dem Standarddialog zur Farbauswahl eingestellt werden.

Positionierung

Zeigerabstand	Länge des Zeigers in Pixel
Beschriftungsverschiebung	Abstand in Pixel, um die Beschriftung zu positionieren
Einheitenverschiebung	Vertikaler Abstand in Pixel, um den Text (eingegeben in Beschriftung→ Einheit) zu positionieren

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche … geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbereiche

Dauerhafte Farbbereiche	Wenn diese Option aktiviert ist, sind die Farbbereiche dauerhaft sichtbar. Die Auswirkungen dieser Option sind nur im Onlinebetrieb sichtbar.
[<n>]	Mit einem Klick auf die Schaltfläche Create new wird ein neuer Farbbereich erzeugt. Für jeden Farbbereich wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf ‚Löschen' wird der zugehörige Farbbereich mit seinen Einstellungen gelöscht.

Bereich [<n>]

Bereichsanfang	Anfang des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Bereichsende	Ende des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Farbe	Farbe des Balkenbereichs

Transparenz wird unter Windows CE nicht unterstützt.

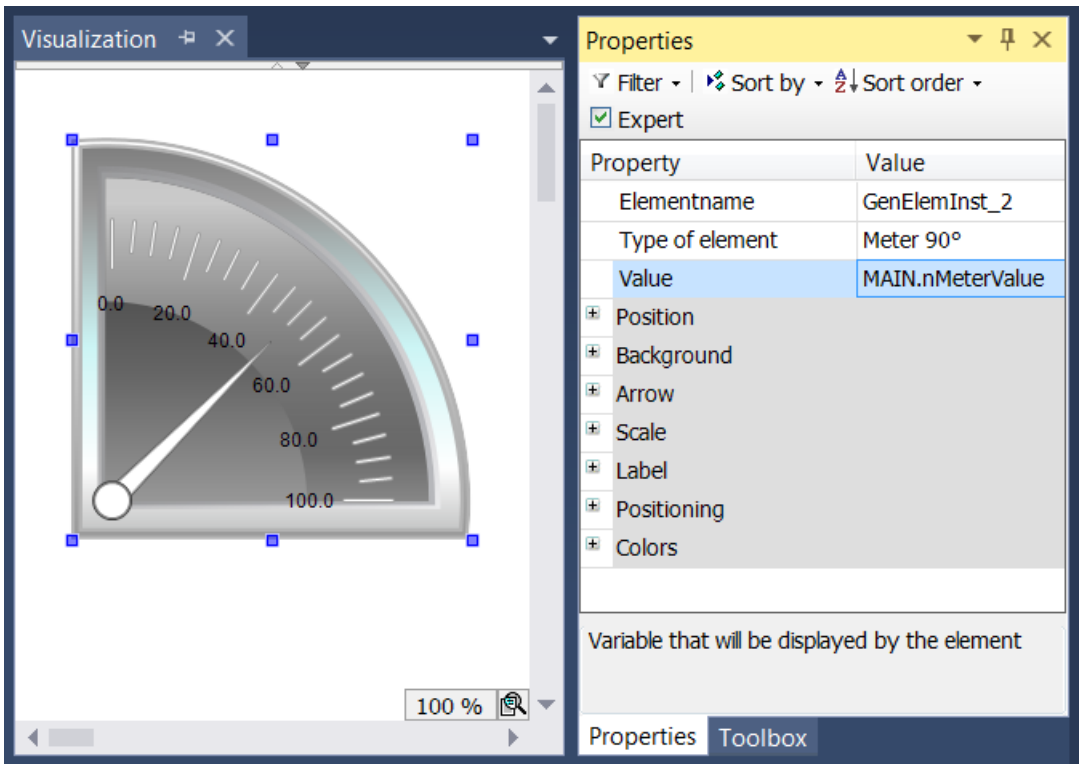
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.5.2.1 Konfiguration eines Zeigerinstruments

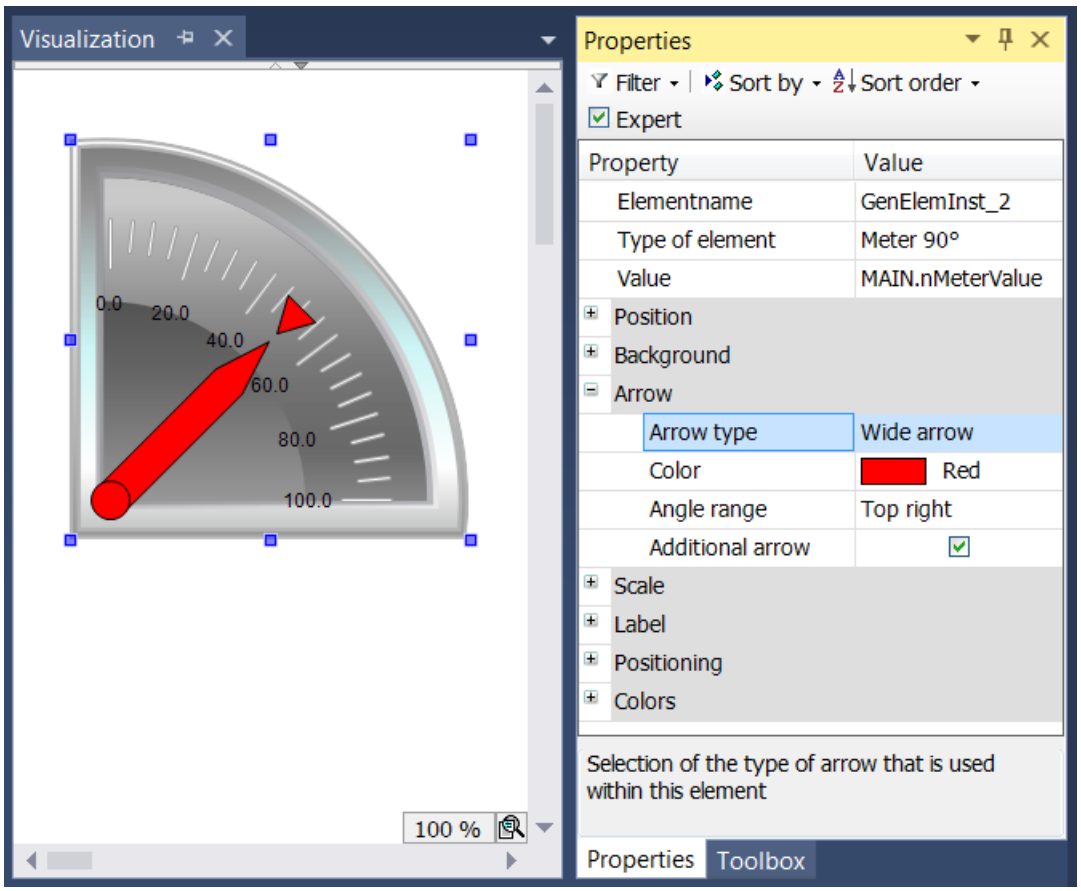
Im Folgenden wird ein Beispiel für die Konfiguration eines Zeigerinstruments erläutert. Dieses Beispiel ist anwendbar für alle verfügbaren Zeigerinstrumente.



Die zu verknüpfende Eingangsvariable – im Beispiel eine Variable namens "nMeterValue" – soll innerhalb der Elementeigenschaften des Zeigerinstrument-Elements unter "Wert" angegeben werden. Nach einem

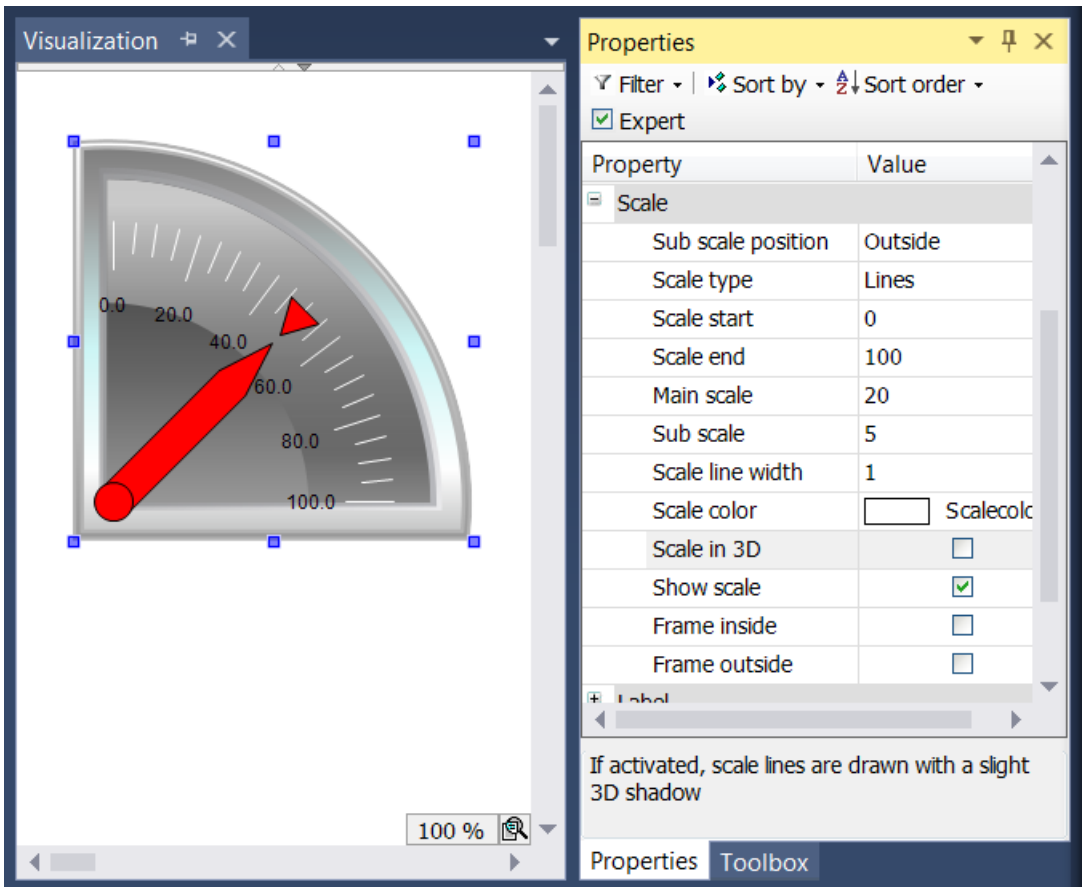
Klick in das Eingabefeld steht die Schaltfläche [...] zur Verfügung. Sie kann betätigt werden, um das Projekt nach der Eingangsvariablen zu durchsuchen. Achten Sie darauf, die Eingangsvariable durch Angabe ihres vollständigen Pfads im Projektbaum zu spezifizieren.

Die Orientierung und Größe der Skalenanzeige sowie die Farbe und das Aussehen des Pfeils können im Abschnitt Zeiger der Elementeigenschaften festgelegt werden.



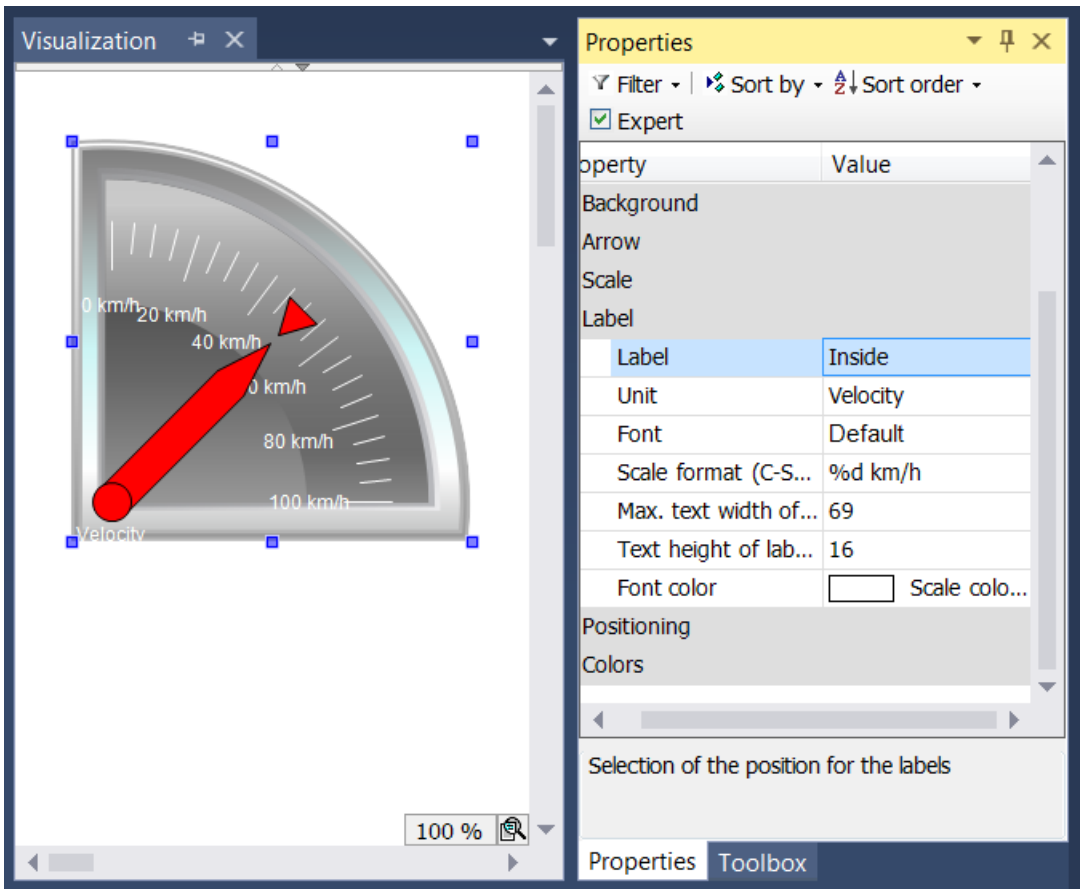
Für das Element mit dem Namen Zeigerinstrument können zusätzlich der "Zeiger Anfang" und das "Zeiger Ende" eingestellt werden. Sie geben den Winkel (in Grad) an, den der linke bzw. rechte Rand der Skala mit der horizontalen Geraden einschließt. Dieser Winkel ist mathematisch, d.h. entgegen dem Uhrzeigersinn orientiert, so dass der Wert im Feld "Zeiger Anfang" immer größer sein muss als der Wert von "Zeiger Ende". Das Paar aus Anfangs- und Endwert ist periodisch mit 360 Grad.

Im Abschnitt Skala kann der Zahlenbereich für die Skala und ein Grob- und Feinskalierung bestimmt werden.



Der Wert vom Skalenanfang wird am linken Rand der Skala eingetragen. Daher muss er kleiner sein als der Wert von Skalenende, der am rechten Rand der Skala eingefügt wird. Im Gegensatz zu der groben Skalierung kann die feinere Unterteilung (Unterskala) weggelassen werden, indem Sie den Wert für Ihre Abstände auf 0 setzen. In diesem Fall würden also keine feinen Skalenstriche angezeigt werden. Da das Kontrollkästchen "Innerer Rahmen" und "Äußerer Rahmen" im Beispiel deaktiviert wurde, sind der innere und äußere Kreisbogen der Skala ausgeblendet.

Nachdem die Skalierung festgesetzt wurde, soll nun im Abschnitt "Beschriftung" die Beschriftung der Skala formatiert werden.

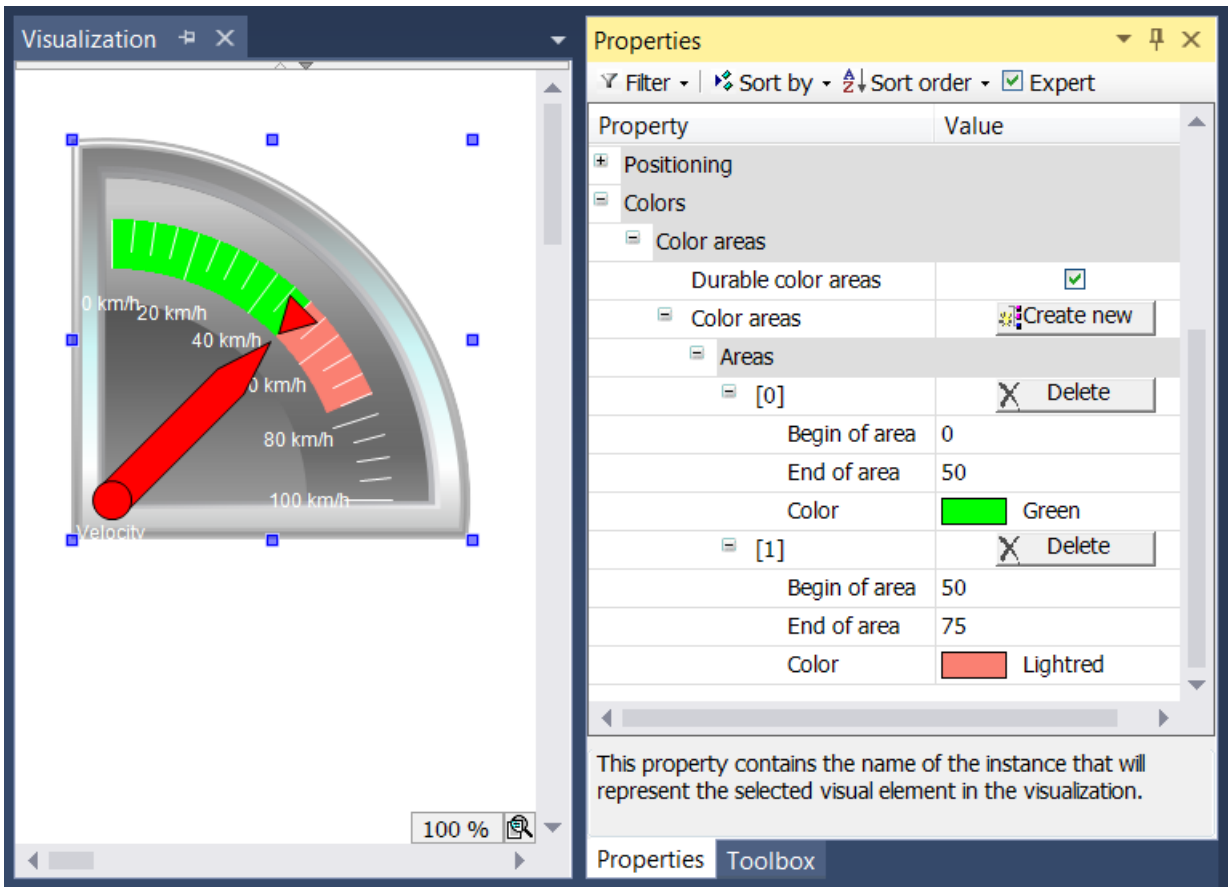


Durch Abänderung der Beschriftung von "Außen" in "Innen" wird die Anzeige der Skalierungsmarken in das Kreisinnere verlagert. Der Eintrag im Feld Einheit erscheint unterhalb des Fußpunktes des Pfeils. Nach Auswahl einer geeigneten Schrift(farbe), wird die Formatierung der Skalierungsmarken angepasst. Der Zahlenwert der Skalierung ist gemäß der Syntax der Programmiersprache C zu formatieren. Benutzen Sie "%d" für ganze Zahlen und "%.Xf" für Gleitkommazahlen, wobei "X" durch die gewünschte Anzahl an Nachkommastellen zu ersetzen ist. Die Werte in den folgenden beiden Eingabefeldern werden entsprechend Ihrer vorhergehenden Einstellungen in diesem Abschnitt eingetragen. Sie müssen die Werte nur dann verändern, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.

Schließlich können im Abschnitt Farben bestimmte Bereiche der Skala eingefärbt werden, indem

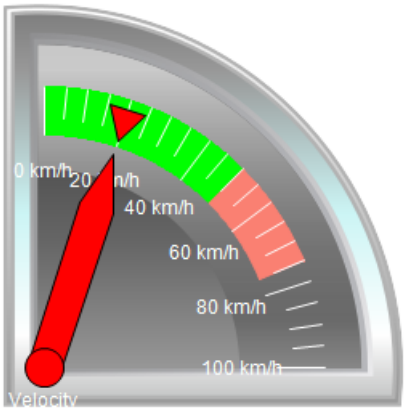
Farbbereiche mithilfe der Schaltfläche 

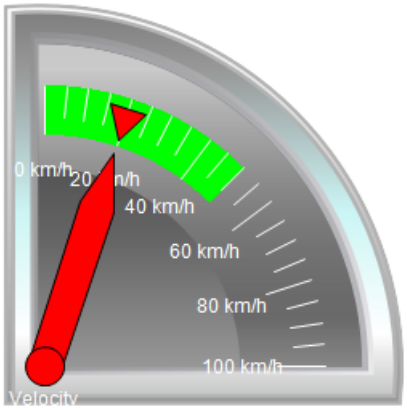
angelegt werden. Die Farbbereiche werden in aufsteigender Ordnung durchnummeriert. Jeder Farbbereich wird innerhalb der Elementeigenschaften mit eigenen Eingabefeldern ausgestattet.



Die Felder "Bereichsanfang" und "Bereichsende" stehen zur Spezifizierung des Unterbereichs der Skalierung zur Verfügung. Die Einfärbung kann aus einem Aufklappmenü ausgewählt werden und mithilfe der Schaltfläche Löschen wieder gelöscht werden.

Der Effekt des Kontrollkästchens "Dauerhafte Farbbereiche" ist nur zur Laufzeit ersichtlich. Im ersten Fall ist das Kontrollkästchen ausgewählt, im zweiten Fall nicht.

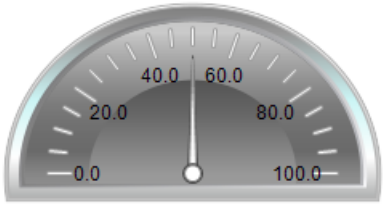




Während das untere Zeigerinstrument nur den Farbbereich anzeigt, der den Pfeil enthält, zeigt das obere Element alle angelegten Farbbereiche an, da das Kontrollkästchen "Dauerhafte Farbbereiche" aktiviert wurde.

15.8.5.3 Zeigerinstrument 180°

Mit dem Zeigerinstrument kann zum Beispiel ein Drehzahlmesser der Visualisierung hinzugefügt werden. Die Nadel positioniert sich nach dem Wert der zugewiesenen Variablen. Das Element hat ein vorkonfiguriertes Design, in dem eine Hintergrundfarbe [▶ 584] gesetzt werden kann. Wahlweise kann dieses Design durch ein eigenes Hintergrundbild [▶ 584] ersetzt werden. Die Skala kann in Farbbereiche [▶ 586] unterteilt werden. Ein Beispiel für die Konfiguration ist im Abschnitt "Konfiguration eines Zeigerinstruments [▶ 587]" zu finden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche [...] die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Wert	Numerische Variable, deren Wert als Ausschlag der Instrumentennadel dargestellt wird.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Hintergrund

Wird kein eigenes Hintergrundbild genutzt, stehen folgende Eigenschaften zur Verfügung:

Farbe Hintergrund	Auszuwählen ist eine Farbe für den Balken: • Gelb • Rot • Grün • Blau • Grau

Wird ein eigenes Hintergrundbild definiert, stehen diese Eigenschaften zur Verfügung:

Bild	Hier kann ein Bild aus einem ImagePool zugewiesen werden, indem der Name der Bilddatei oder dessen ID angegeben wird.
Transparenzfarbe	Für Bilder mit transparentem Hintergrund kann eine Farbe ausgewählt werden, die transparent dargestellt werden soll. Die Schaltfläche  öffnet den Farbenauswahldialog.

Zeiger

Zeigertyp	Es gibt eine Auswahl an verschiedenen Zeigertypen: • Normaler Zeiger • Dünner Zeiger • Breiter Zeiger • Dünne Nadel • Dünner 3D-Zeiger • Dünne 3D-Nadel
Farbe	Farbe, mit der der Zeiger angezeigt wird
Winkelbereich	Hier kann ein 180° Ausschnitt ausgewählt werden: • Oben • Unten • Links • Rechts
Zusätzlicher Zeiger	Wenn diese Option aktiviert ist, ist ein zusätzlicher Zeiger auf der Skala gegenüber der Nadel sichtbar.

Skala

Unterskalaposition	Die Unterskala kann am äußeren oder inneren Radius des Skalenrings angezeigt werden: • Außen • Innen
Skalentyp	Die Skala kann angezeigt werden als: • Linien • Punkte • Quadrate
Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala – Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenlinienstärke	Dicke der Skalenlinie in Pixel
Skalenfarbe	Die Farbe kann über die Selektionsliste bzw. über die Schaltfläche  aus dem Standarddialog zur Farbauswahl eingestellt werden.
Skala in 3D	Wenn diese Option aktiviert ist, wird die Skala 3-dimensional dargestellt.
Skala anzeigen	Wenn diese Option aktiviert ist, wird die Skala dargestellt.
Innerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring innen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.
Äußerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring außen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.

Beschriftung

Beschriftung	Hier kann eingestellt werden, ob die Skalenwerte auf der Außenseite oder Innenseite der Skala platziert werden sollen.
Einheit	Der eingegebene Text beschriftet das Element. Er wird unterhalb des Mittelpunkts der Skalierung angezeigt. Auf diese Weise kann beispielsweise die Skalierungseinheit angegeben werden.
Schriftart	Hier ist die Schriftart für Einheit und Skalierung festzulegen: • Standard • Überschrift • Große • Titel • Anmerkung Mit Klick auf die Schaltfläche [...] öffnet ein Dialog für benutzerdefinierte Einstellungen der Schrifteigenschaften.
Skalenformat (C-Syntax)	Benutzen Sie die C-Syntax zur Angabe der Formatierung der Skalierungsbeschriftung. So führt die Eingabe des Strings "%3.2f s" in diesem Feld zur Anzeige der Skalierungsbeschriftungen mit 3 Stellen, davon 2 Nachkommastellen, gefolgt vom Buchstaben "s".
Max. Breite der Beschriftungen	Wert, der die maximale Breite der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Höhe der Beschriftungen	Wert, der die Höhe der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Farbe Schriftart	Die Farbe für die Schrift kann über Selektionsliste bzw. über Schaltfläche [...] aus dem Standarddialog zur Farbauswahl eingestellt werden.

Positionierung

Zeigerabstand	Länge des Zeigers in Pixel
Beschriftungsverschiebung	Abstand in Pixel, um die Beschriftung zu positionieren
Einheitenverschiebung	Vertikaler Abstand in Pixel, um den Text (eingegeben in Beschriftung→ Einheit) zu positionieren

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche [...] geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbereiche

Dauerhafte Farbbereiche	Wenn diese Option aktiviert ist, sind die Farbbereiche dauerhaft sichtbar. Die Auswirkungen dieser Option sind nur im Onlinebetrieb sichtbar.
[<n>]	Mit einem Klick auf die Schaltfläche Create new wird ein neuer Farbbereich erzeugt. Für jeden Farbbereich wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf ‚Löschen' wird der zugehörige Farbbereich mit seinen Einstellungen gelöscht.

Bereich [<n>]

Bereichsanfang	Anfang des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Bereichsende	Ende des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Farbe	Farbe des Balkenbereichs

Transparenz wird unter Windows CE nicht unterstützt.

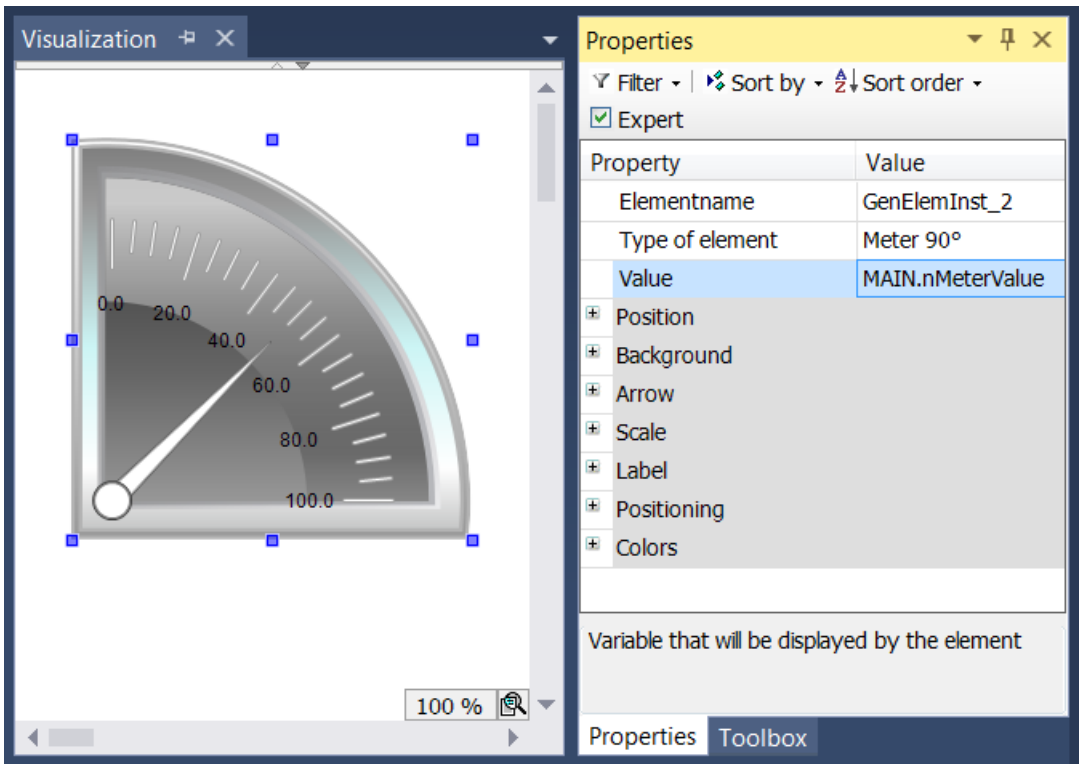
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.5.3.1 Konfiguration eines Zeigerinstruments

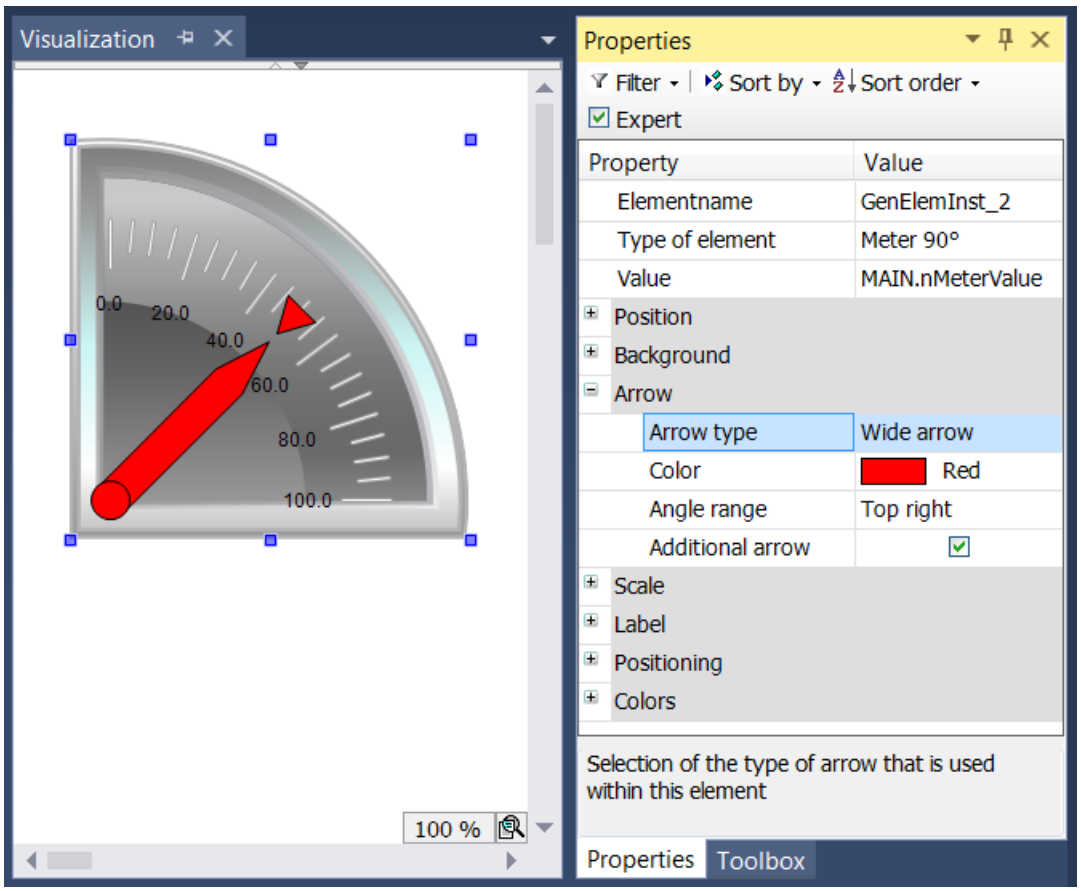
Im Folgenden wird ein Beispiel für die Konfiguration eines Zeigerinstruments erläutert. Dieses Beispiel ist anwendbar für alle verfügbaren Zeigerinstrumente.



Die zu verknüpfende Eingangsvariable – im Beispiel eine Variable namens "nMeterValue" – soll innerhalb der Elementeigenschaften des Zeigerinstrument-Elements unter "Wert" angegeben werden. Nach einem

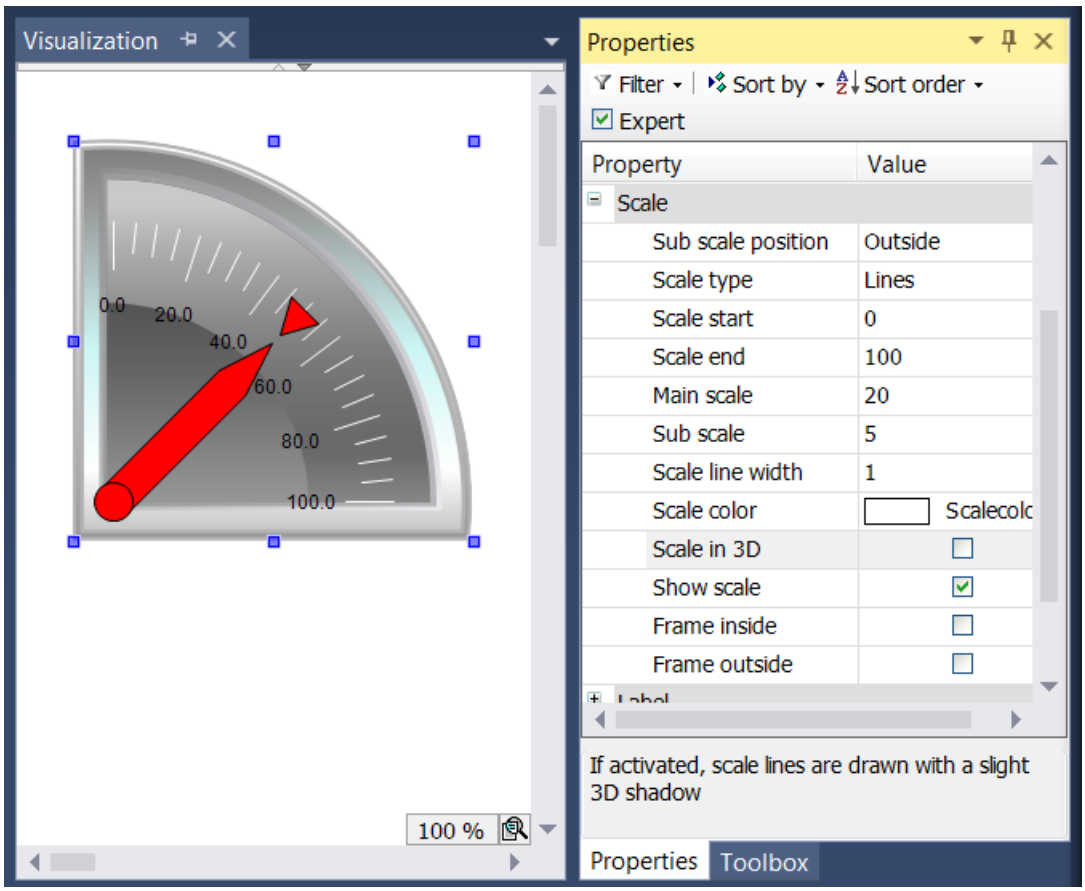
Klick in das Eingabefeld steht die Schaltfläche [...] zur Verfügung. Sie kann betätigt werden, um das Projekt nach der Eingangsvariablen zu durchsuchen. Achten Sie darauf, die Eingangsvariable durch Angabe ihres vollständigen Pfads im Projektbaum zu spezifizieren.

Die Orientierung und Größe der Skalenanzeige sowie die Farbe und das Aussehen des Pfeils können im Abschnitt Zeiger der Elementeigenschaften festgelegt werden.



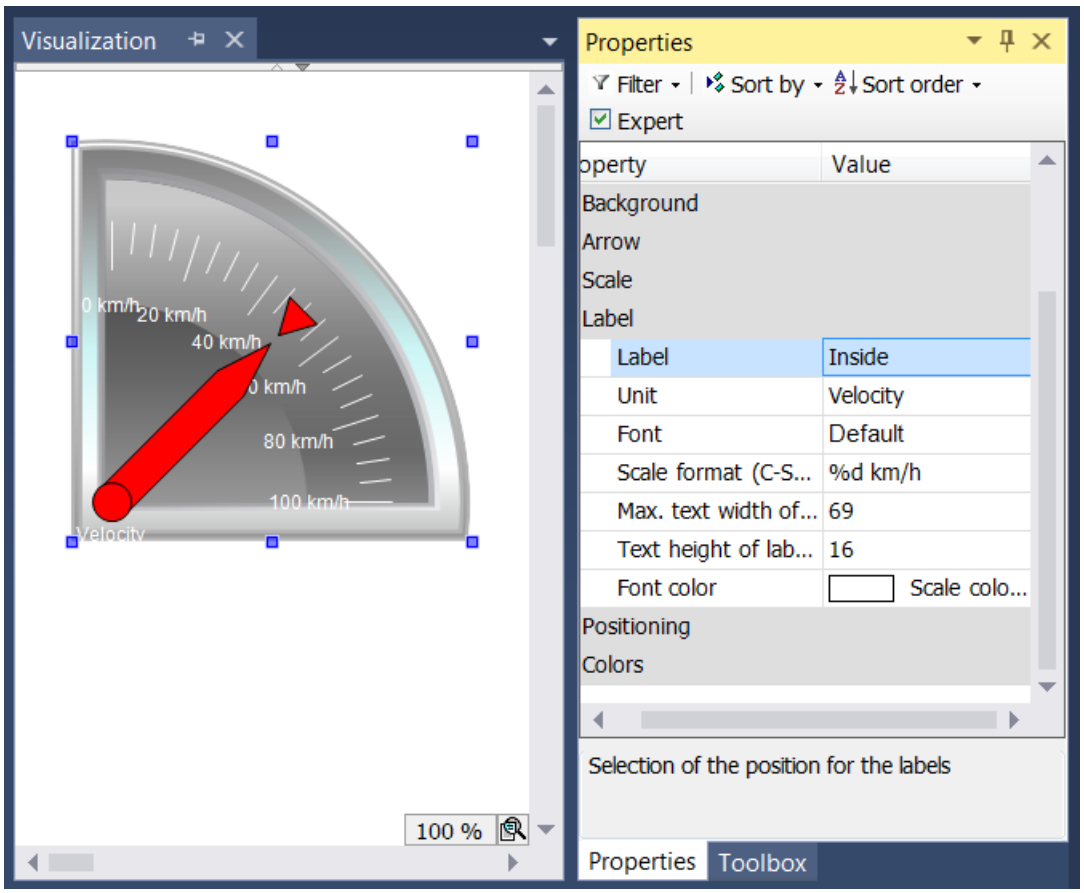
Für das Element mit dem Namen Zeigerinstrument können zusätzlich der "Zeiger Anfang" und das "Zeiger Ende" eingestellt werden. Sie geben den Winkel (in Grad) an, den der linke bzw. rechte Rand der Skala mit der horizontalen Geraden einschließt. Dieser Winkel ist mathematisch, d.h. entgegen dem Uhrzeigersinn orientiert, so dass der Wert im Feld "Zeiger Anfang" immer größer sein muss als der Wert von "Zeiger Ende". Das Paar aus Anfangs- und Endwert ist periodisch mit 360 Grad.

Im Abschnitt Skala kann der Zahlenbereich für die Skala und ein Grob- und Feinskalierung bestimmt werden.



Der Wert vom Skalenanfang wird am linken Rand der Skala eingetragen. Daher muss er kleiner sein als der Wert von Skalenende, der am rechten Rand der Skala eingefügt wird. Im Gegensatz zu der groben Skalierung kann die feinere Unterteilung (Unterskala) weggelassen werden, indem Sie den Wert für Ihre Abstände auf 0 setzen. In diesem Fall würden also keine feinen Skalenstriche angezeigt werden. Da das Kontrollkästchen "Innerer Rahmen" und "Äußerer Rahmen" im Beispiel deaktiviert wurde, sind der innere und äußere Kreisbogen der Skala ausgeblendet.

Nachdem die Skalierung festgesetzt wurde, soll nun im Abschnitt "Beschriftung" die Beschriftung der Skala formatiert werden.

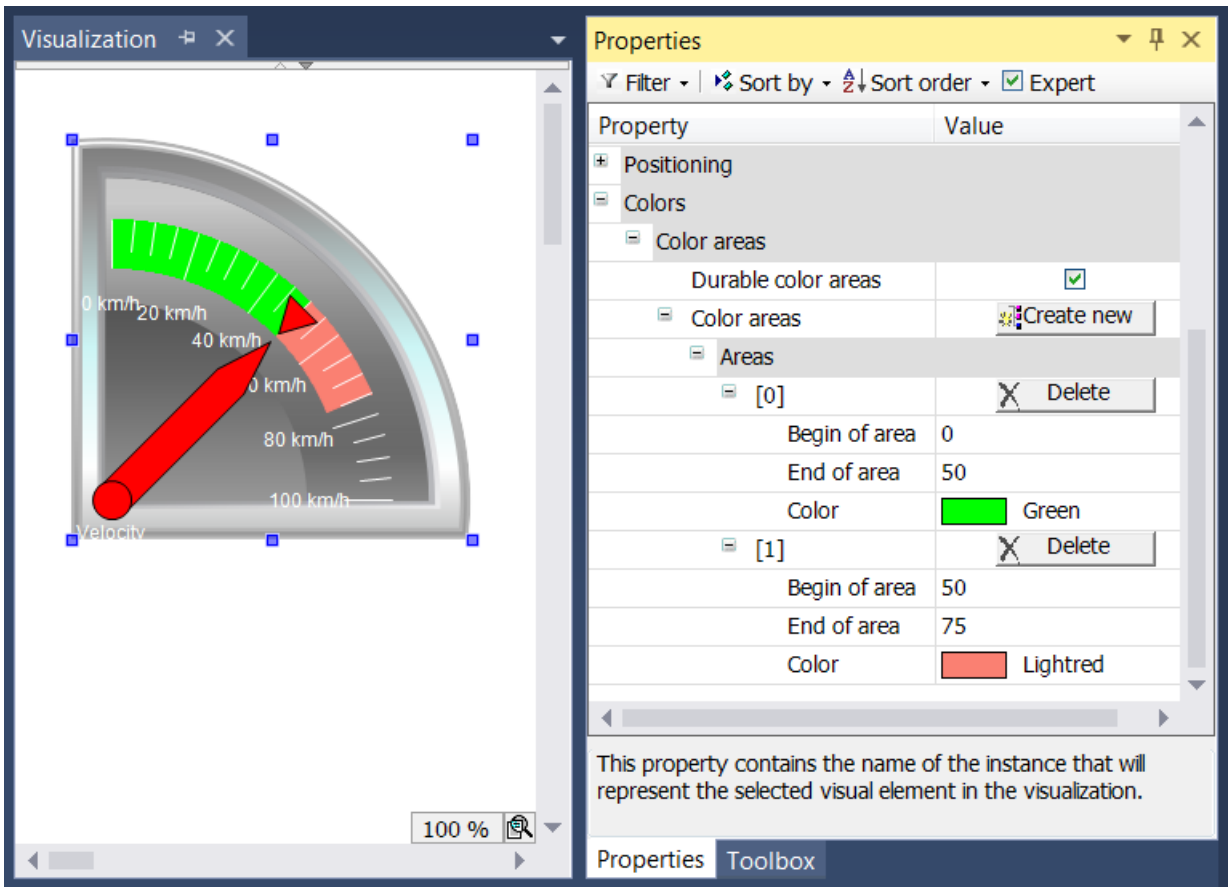


Durch Abänderung der Beschriftung von "Außen" in "Innen" wird die Anzeige der Skalierungsmarken in das Kreisinnere verlagert. Der Eintrag im Feld Einheit erscheint unterhalb des Fußpunktes des Pfeils. Nach Auswahl einer geeigneten Schrift(farbe), wird die Formatierung der Skalierungsmarken angepasst. Der Zahlenwert der Skalierung ist gemäß der Syntax der Programmiersprache C zu formatieren. Benutzen Sie "%d" für ganze Zahlen und "%.Xf" für Gleitkommazahlen, wobei "X" durch die gewünschte Anzahl an Nachkommastellen zu ersetzen ist. Die Werte in den folgenden beiden Eingabefeldern werden entsprechend Ihrer vorhergehenden Einstellungen in diesem Abschnitt eingetragen. Sie müssen die Werte nur dann verändern, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.

Schließlich können im Abschnitt Farben bestimmte Bereiche der Skala eingefärbt werden, indem

Farbbereiche mithilfe der Schaltfläche 

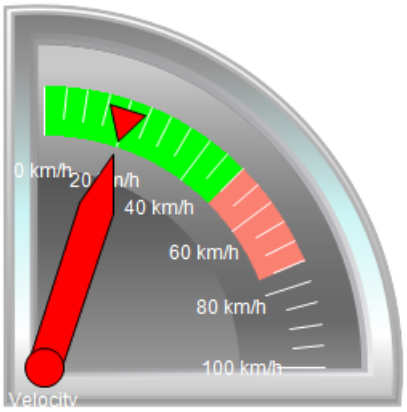
angelegt werden. Die Farbbereiche werden in aufsteigender Ordnung durchnummeriert. Jeder Farbbereich wird innerhalb der Elementeigenschaften mit eigenen Eingabefeldern ausgestattet.

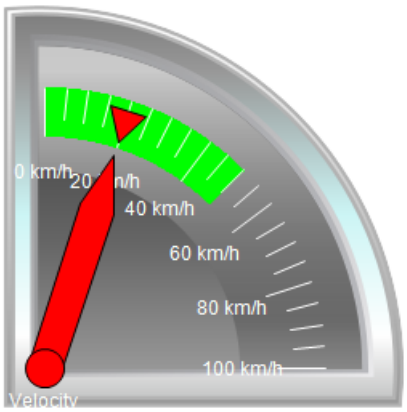


Die Felder "Bereichsanfang" und "Bereichsende" stehen zur Spezifizierung des Unterbereichs der Skalierung zur Verfügung. Die Einfärbung kann aus einem Aufklappmenü ausgewählt werden und mithilfe der Schaltfläche 

wieder gelöscht werden.

Der Effekt des Kontrollkästchens "Dauerhafte Farbbereiche" ist nur zur Laufzeit ersichtlich. Im ersten Fall ist das Kontrollkästchen ausgewählt, im zweiten Fall nicht.

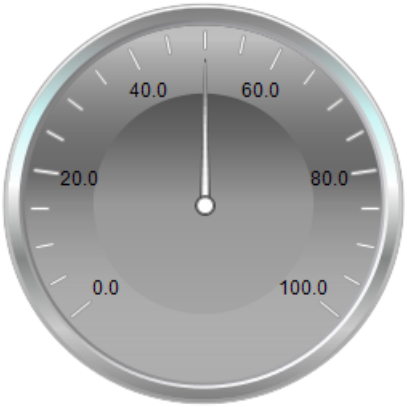




Während das untere Zeigerinstrument nur den Farbbereich anzeigt, der den Pfeil enthält, zeigt das obere Element alle angelegten Farbbereiche an, da das Kontrollkästchen "Dauerhafte Farbbereiche" aktiviert wurde.

15.8.5.4 Zeigerinstrument

Mit dem Zeigerinstrument kann zum Beispiel ein Drehzahlmesser der Visualisierung hinzugefügt werden. Die Nadel positioniert sich nach dem Wert der zugewiesenen Variablen. Das Element hat ein vorkonfiguriertes Design, in dem die Hintergrundfarbe [▶ 593] gesetzt werden kann. Wahlweise kann dieses Design durch ein eigenes Hintergrundbild [▶ 593] ersetzt werden. Die Skala kann in Farbbereiche [▶ 595] unterteilt werden. Ein Beispiel für die Konfiguration ist im Abschnitt "Konfiguration eines Zeigerinstruments [▶ 596]" zu finden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.

- öffnen Sie mit der Schaltfläche [...] die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Wert	Numerische Variable, deren Wert als Ausschlag der Instrumentennadel dargestellt wird.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Hintergrund

Wird kein eigenes Hintergrundbild genutzt, stehen folgende Eigenschaften zur Verfügung:

Farbe Hintergrund	Auszuwählen ist eine Farbe für den Balken: • Gelb • Rot • Grün • Blau • Grau

Wird ein eigenes Hintergrundbild definiert, stehen diese Eigenschaften zur Verfügung:

Bild	Hier kann ein Bild aus einem ImagePool zugewiesen werden, indem der Name der Bilddatei oder dessen ID angegeben wird.
Transparenzfarbe	Für Bilder mit transparentem Hintergrund kann eine Farbe ausgewählt werden, die transparent dargestellt werden soll. Die Schaltfläche [...] öffnet den Farbenauswahldialog.

Zeiger

Zeigertyp	Es gibt eine Auswahl an verschiedenen Zeigertypen: • Normaler Zeiger • Dünner Zeiger • Breiter Zeiger • Dünne Nadel • Dünner 3D-Zeiger • Dünne 3D-Nadel
Farbe	Farbe, mit der der Zeiger angezeigt wird
Zeiger Anfang	Der hier anzugebene Wert wird als Winkel in Grad interpretiert, der sich vom Nullpunkt der Skala bis zum Anfang des Skalenbereichs im Gegenuhrzeigersinn aufspannt. Nullpunkt der Skala liegt bei "3 Uhr". Der Winkel hat einen Wertebereich von 0° bis 360°. Zentrum der Drehung ist die X/Y-Position.
Zeiger Ende	Der hier anzugebene Wert wird als Winkel in Grad interpretiert, der sich vom Nullpunkt der Skala bis zum Ende des Skalenbereichs im Gegenuhrzeigersinn aufspannt. Nullpunkt der Skala liegt bei "3 Uhr". Eingabe eines negativen Werts ist erlaubt. Zentrum der Drehung ist die X/Y-Position. Zeiger Ende muss so gesetzt werden, dass es im Uhrzeigersinn betrachtet rechts von Zeiger Anfang zu liegen kommt. Der Winkel hat einen Wertebereich von 0° bis maximal 350°.
Zusätzlicher Zeiger	Wenn diese Option aktiviert ist, ist ein zusätzlicher Zeiger auf der Skala gegenüber der Nadel sichtbar.

Skala

Unterskalaposition	Die Unterskala kann am äußeren oder inneren Radius des Skalenrings angezeigt werden: • Außen • Innen
Skalentyp	Die Skala kann angezeigt werden als: • Linien • Punkte • Quadrate
Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala – Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenlinienstärke	Dicke der Skalenlinie in Pixel
Skalenfarbe	Die Farbe kann über die Selektionsliste bzw. über die Schaltfläche  aus dem Standarddialog zur Farbauswahl eingestellt werden.
Skala in 3D	Wenn diese Option aktiviert ist, wird die Skala 3-dimensional dargestellt.
Skala anzeigen	Wenn diese Option aktiviert ist, wird die Skala dargestellt.
Innerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring innen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.
Äußerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring außen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.

Beschriftung

Beschriftung	Hier kann eingestellt werden, ob die Skalenwerte auf der Außenseite oder Innenseite der Skala platziert werden sollen.
Einheit	Der eingegebene Text beschriftet das Element. Er wird unterhalb des Mittelpunkts der Skalierung angezeigt. Auf diese Weise kann beispielsweise die Skalierungseinheit angegeben werden.
Schriftart	Hier ist die Schriftart für Einheit und Skalierung festzulegen: • Standard • Überschrift • Große • Titel • Anmerkung Mit Klick auf die Schaltfläche … öffnet ein Dialog für benutzerdefinierte Einstellungen der Schrifteigenschaften.
Skalenformat (C-Syntax)	Benutzen Sie die C-Syntax zur Angabe der Formatierung der Skalierungsbeschriftung. So führt die Eingabe des Strings "%3.2f s" in diesem Feld zur Anzeige der Skalierungsbeschriftungen mit 3 Stellen, davon 2 Nachkommastellen, gefolgt vom Buchstaben "s".
Max. Breite der Beschriftungen	Wert, der die maximale Breite der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Höhe der Beschriftungen	Wert, der die Höhe der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Farbe Schriftart	Die Farbe für die Schrift kann über Selektionsliste bzw. über Schaltfläche … aus dem Standarddialog zur Farbauswahl eingestellt werden.

Positionierung

Zeigerabstand	Länge des Zeigers in Pixel
Skalenabstand	Abstand der Skalenstriche zum Zentrum in Pixel. Diese Eigenschaft ist nur verfügbar, wenn ein eigenes Hintergrundbild definiert worden ist.
Skalenlänge	Länge der Skalenstriche in Pixel. Diese Eigenschaft ist nur verfügbar, wenn ein eigenes Hintergrundbild definiert worden ist.
Beschriftungsverschiebung	Abstand in Pixel, um die Beschriftung zu positionieren
Einheitenverschiebung	Vertikaler Abstand in Pixel, um den Text (eingegeben in Beschriftung → Einheit) zu positionieren
Ursprungsverschiebung	Versatz des Elements. Er kann verwendet werden, um eine exakte Positionierung relativ zum Hintergrundbild zu erreichen.

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche … geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbereiche

Dauerhafte Farbbereiche	Wenn diese Option aktiviert ist, sind die Farbbereiche dauerhaft sichtbar. Die Auswirkungen dieser Option sind nur im Onlinebetrieb sichtbar.
[<n>]	Mit einem Klick auf die Schaltfläche Create new wird ein neuer Farbbereich erzeugt. Für jeden Farbbereich wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf ‚Löschen' wird der zugehörige Farbbereich mit seinen Einstellungen gelöscht.

Bereich [<n>]

Bereichsanfang	Anfang des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Bereichsende	Ende des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Farbe	Farbe des Balkenbereichs

Transparenz wird unter Windows CE nicht unterstützt.

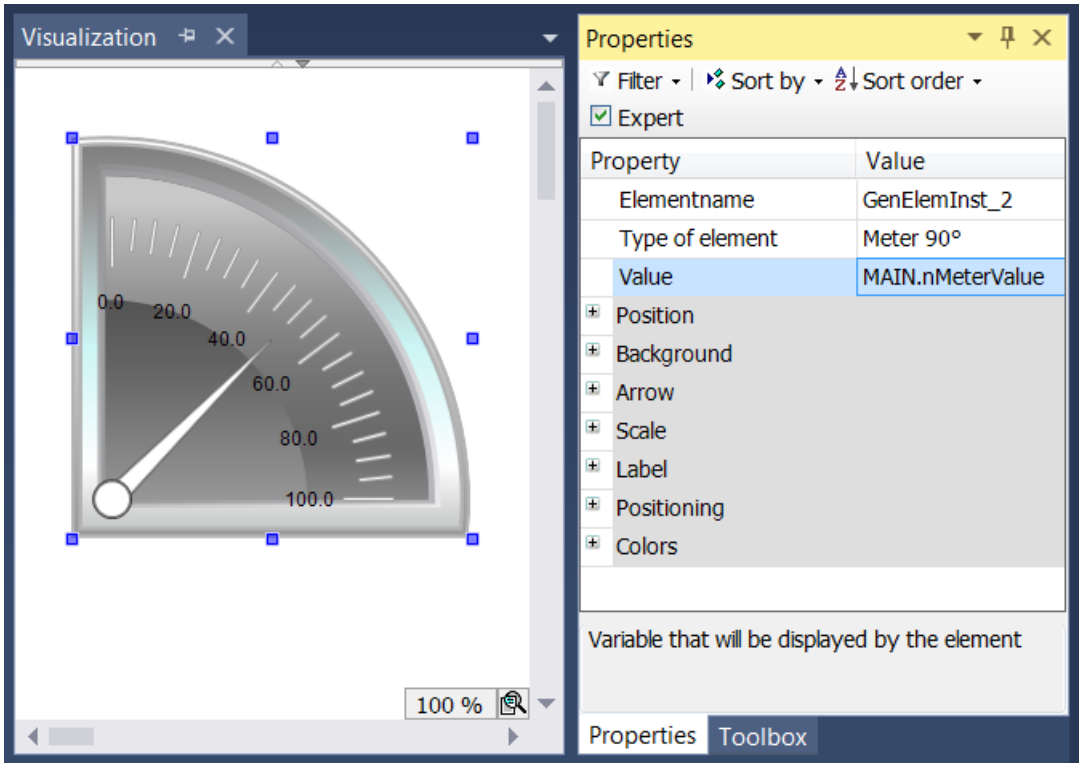
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.5.4.1 Konfiguration eines Zeigerinstruments

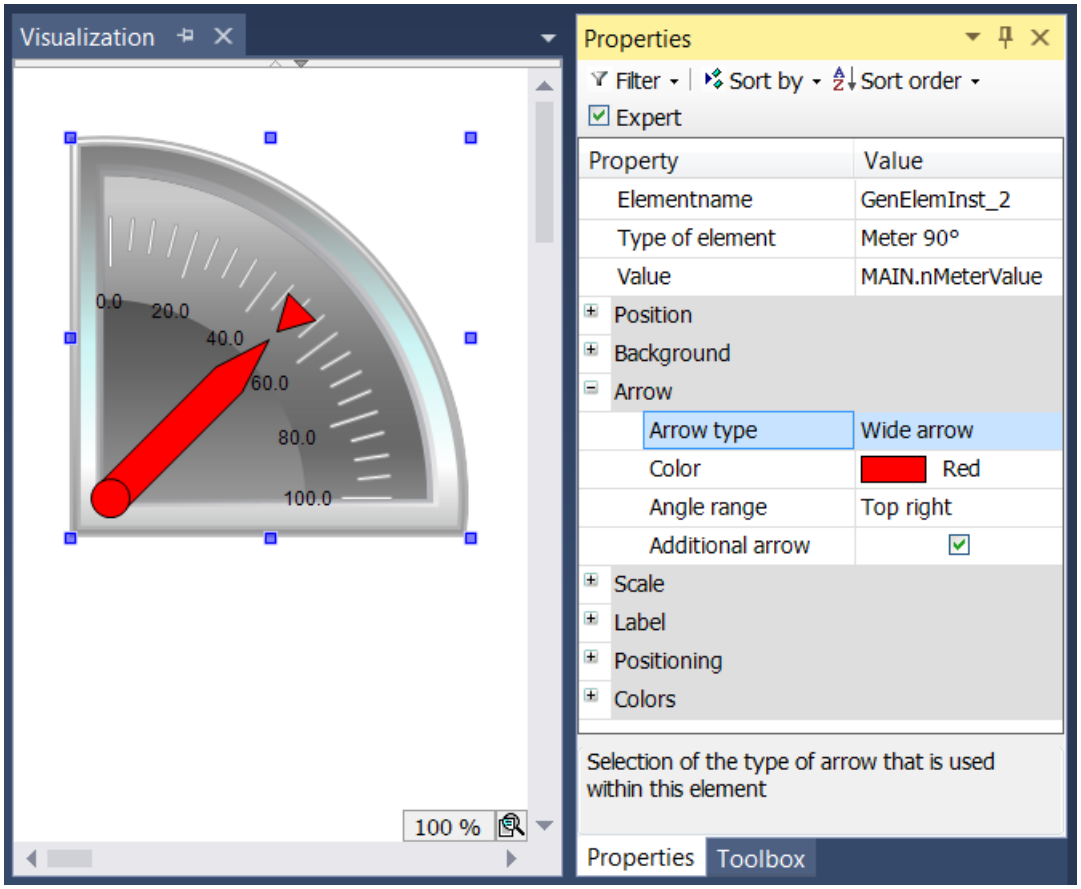
Im Folgenden wird ein Beispiel für die Konfiguration eines Zeigerinstruments erläutert. Dieses Beispiel ist anwendbar für alle verfügbaren Zeigerinstrumente.



Die zu verknüpfende Eingangsvariable – im Beispiel eine Variable namens "nMeterValue" – soll innerhalb der Elementeigenschaften des Zeigerinstrument-Elements unter "Wert" angegeben werden. Nach einem

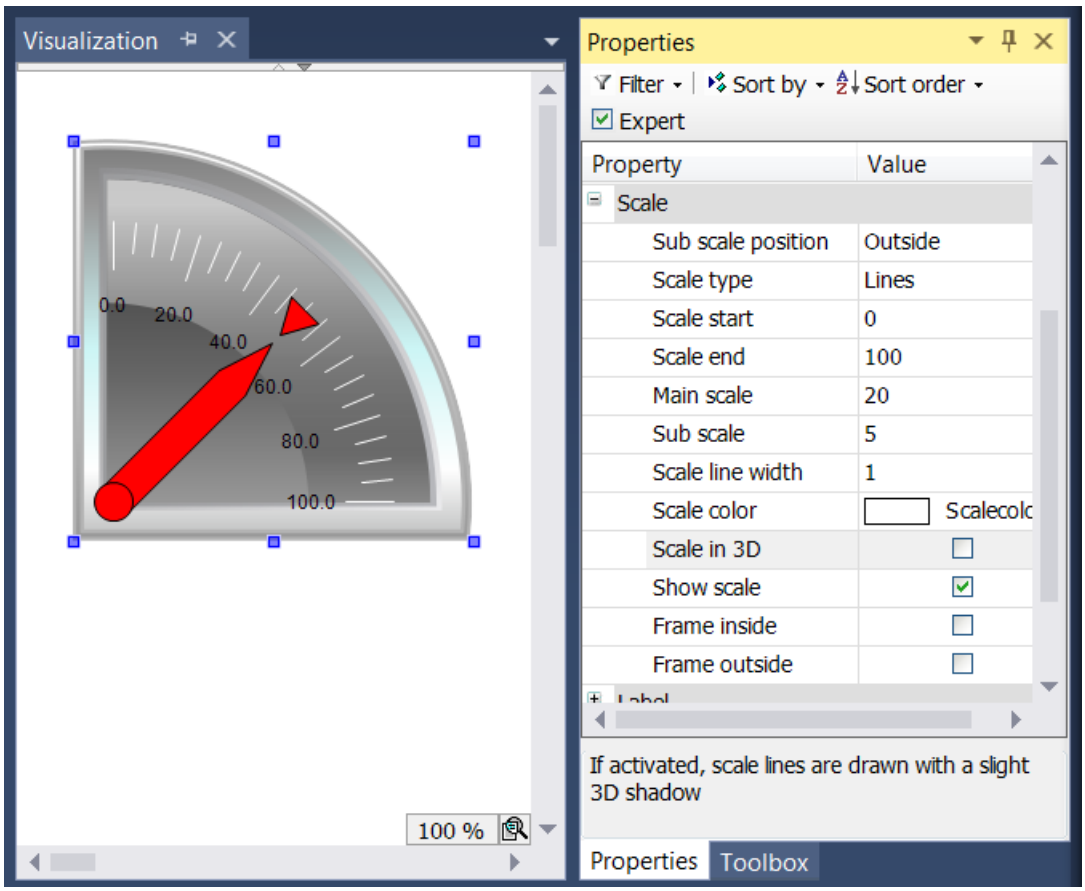
Klick in das Eingabefeld steht die Schaltfläche ... zur Verfügung. Sie kann betätigt werden, um das Projekt nach der Eingangsvariablen zu durchsuchen. Achten Sie darauf, die Eingangsvariable durch Angabe ihres vollständigen Pfads im Projektbaum zu spezifizieren.

Die Orientierung und Größe der Skalenanzeige sowie die Farbe und das Aussehen des Pfeils können im Abschnitt Zeiger der Elementeigenschaften festgelegt werden.



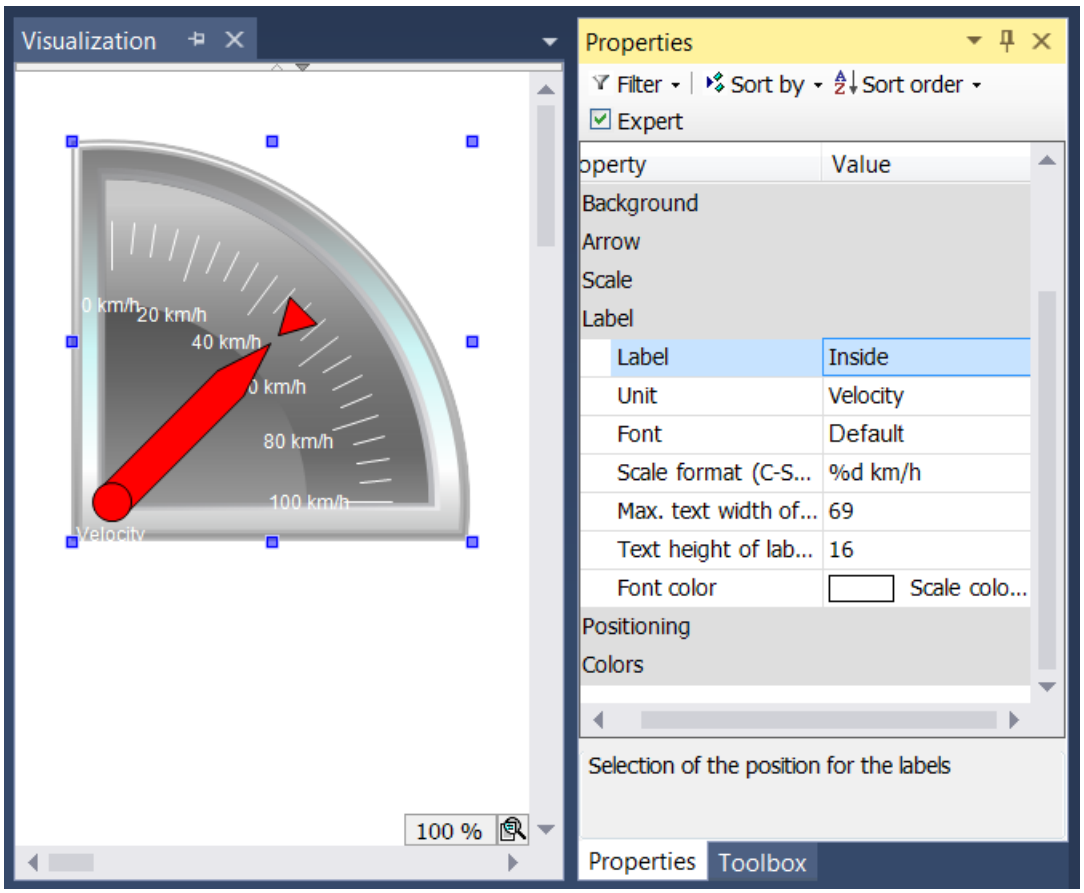
Für das Element mit dem Namen Zeigerinstrument können zusätzlich der "Zeiger Anfang" und das "Zeiger Ende" eingestellt werden. Sie geben den Winkel (in Grad) an, den der linke bzw. rechte Rand der Skala mit der horizontalen Geraden einschließt. Dieser Winkel ist mathematisch, d.h. entgegen dem Uhrzeigersinn orientiert, so dass der Wert im Feld "Zeiger Anfang" immer größer sein muss als der Wert von "Zeiger Ende". Das Paar aus Anfangs- und Endwert ist periodisch mit 360 Grad.

Im Abschnitt Skala kann der Zahlenbereich für die Skala und ein Grob- und Feinskalierung bestimmt werden.



Der Wert vom Skalenanfang wird am linken Rand der Skala eingetragen. Daher muss er kleiner sein als der Wert von Skalenende, der am rechten Rand der Skala eingefügt wird. Im Gegensatz zu der groben Skalierung kann die feinere Unterteilung (Unterskala) weggelassen werden, indem Sie den Wert für Ihre Abstände auf 0 setzen. In diesem Fall würden also keine feinen Skalenstriche angezeigt werden. Da das Kontrollkästchen "Innerer Rahmen" und "Äußerer Rahmen" im Beispiel deaktiviert wurde, sind der innere und äußere Kreisbogen der Skala ausgeblendet.

Nachdem die Skalierung festgesetzt wurde, soll nun im Abschnitt "Beschriftung" die Beschriftung der Skala formatiert werden.

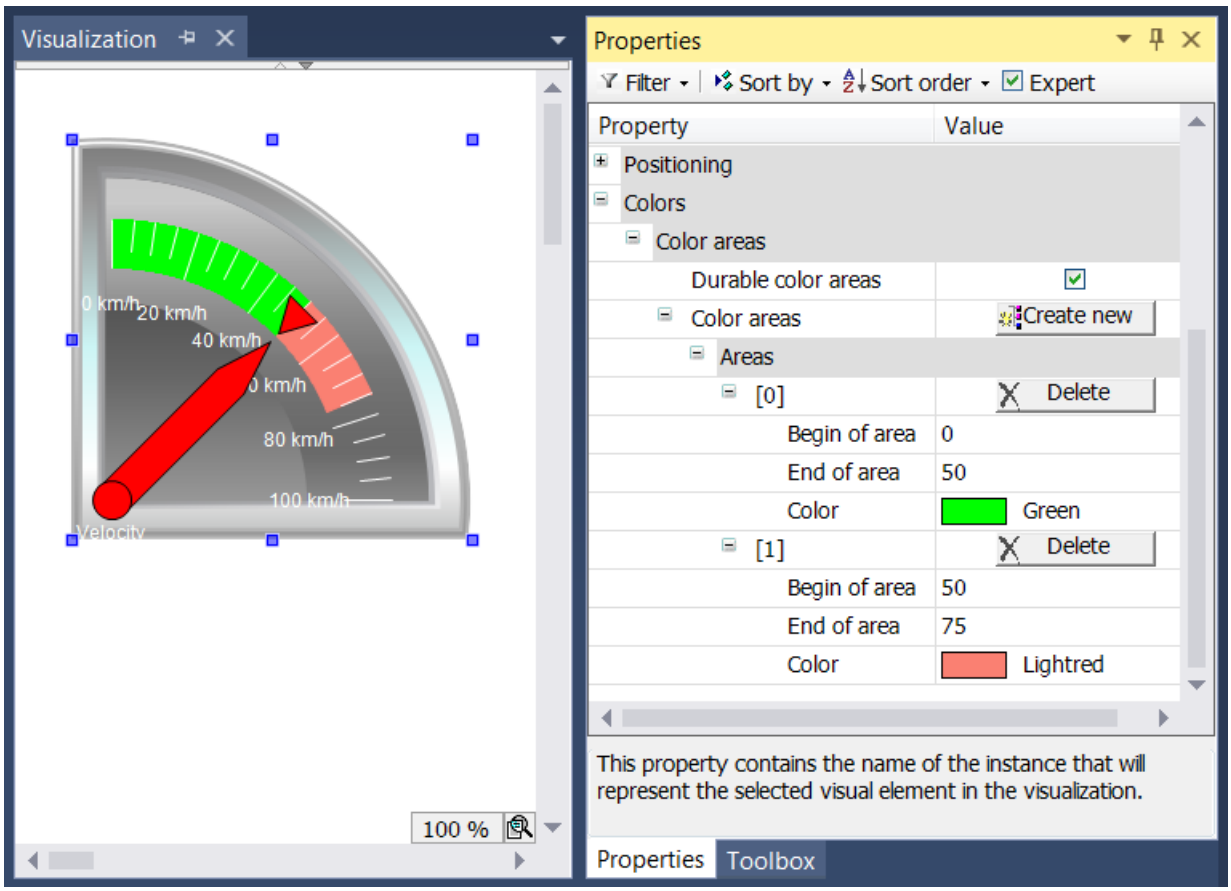


Durch Abänderung der Beschriftung von "Außen" in "Innen" wird die Anzeige der Skalierungsmarken in das Kreisinnere verlagert. Der Eintrag im Feld Einheit erscheint unterhalb des Fußpunktes des Pfeils. Nach Auswahl einer geeigneten Schrift(farbe), wird die Formatierung der Skalierungsmarken angepasst. Der Zahlenwert der Skalierung ist gemäß der Syntax der Programmiersprache C zu formatieren. Benutzen Sie "%d" für ganze Zahlen und "%.Xf" für Gleitkommazahlen, wobei "X" durch die gewünschte Anzahl an Nachkommastellen zu ersetzen ist. Die Werte in den folgenden beiden Eingabefeldern werden entsprechend Ihrer vorhergehenden Einstellungen in diesem Abschnitt eingetragen. Sie müssen die Werte nur dann verändern, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.

Schließlich können im Abschnitt Farben bestimmte Bereiche der Skala eingefärbt werden, indem

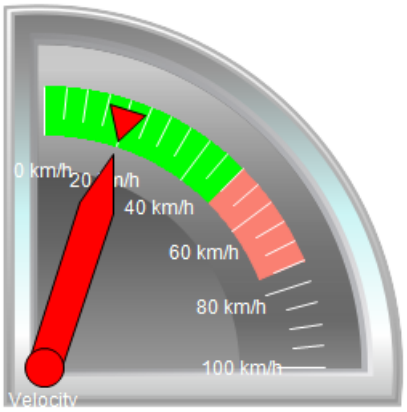
Farbbereiche mithilfe der Schaltfläche 

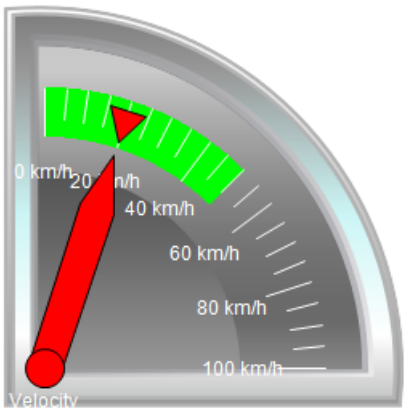
angelegt werden. Die Farbbereiche werden in aufsteigender Ordnung durchnummeriert. Jeder Farbbereich wird innerhalb der Elementeigenschaften mit eigenen Eingabefeldern ausgestattet.



Die Felder "Bereichsanfang" und "Bereichsende" stehen zur Spezifizierung des Unterbereichs der Skalierung zur Verfügung. Die Einfärbung kann aus einem Aufklappmenü ausgewählt werden und mithilfe der Schaltfläche Löschen wieder gelöscht werden.

Der Effekt des Kontrollkästchens "Dauerhafte Farbbereiche" ist nur zur Laufzeit ersichtlich. Im ersten Fall ist das Kontrollkästchen ausgewählt, im zweiten Fall nicht.

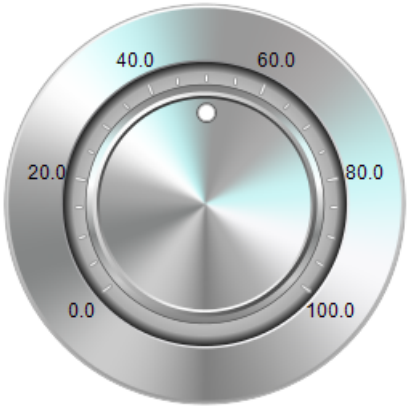




Während das untere Zeigerinstrument nur den Farbbereich anzeigt, der den Pfeil enthält, zeigt das obere Element alle angelegten Farbbereiche an, da das Kontrollkästchen "Dauerhafte Farbbereiche" aktiviert wurde.

15.8.5.5 Potentiometer

Mit diesem Element kann zu einer Visualisierungsseite ein Potentiometer mit vordefiniertem Design hinzugefügt werden. Solch ein Bedienelement zeigt mittels Pfeil bzw. Zeiger einen Variablenwert an, der über die Benutzereingabe verschoben werden kann.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche [...] die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Variable	Numerische Variable, die die Position des Potentiometers enthält.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Hintergrund

Wird kein eigenes Hintergrundbild genutzt, stehen folgende Eigenschaften zur Verfügung:

Farbe Hintergrund	Auszuwählen ist eine Farbe für den Balken: • Gelb • Rot • Grün • Blau • Grau

Wird ein eigenes Hintergrundbild definiert, stehen diese Eigenschaften zur Verfügung:

Bild	Hier kann ein Bild aus einem ImagePool zugewiesen werden, indem der Name der Bilddatei oder dessen ID angegeben wird.
Transparenzfarbe	Für Bilder mit transparentem Hintergrund kann eine Farbe ausgewählt werden, die transparent dargestellt werden soll. Die Schaltfläche  öffnet den Farbenauswahldialog.

Zeiger

Zeigertyp	Es gibt eine Auswahl an verschiedenen Zeigertypen: • Normaler Zeiger • Dünner Zeiger • Breiter Zeiger • Dünne Nadel • Dünner 3D-Zeiger • Dünne 3D-Nadel
Farbe	Farbe, mit der der Zeiger angezeigt wird
Zeiger Anfang	Der hier anzugebene Wert wird als Winkel in Grad interpretiert, der sich vom Nullpunkt der Skala bis zum Anfang des Skalenbereichs im Gegenuhrzeigersinn aufspannt. Nullpunkt der Skala liegt bei "3 Uhr". Der Winkel hat einen Wertebereich von 0° bis 360°. Zentrum der Drehung ist die X/Y-Position.
Zeiger Ende	Der hier anzugebene Wert wird als Winkel in Grad interpretiert, der sich vom Nullpunkt der Skala bis zum Ende des Skalenbereichs im Gegenuhrzeigersinn aufspannt. Nullpunkt der Skala liegt bei "3 Uhr". Eingabe eines negativen Werts ist erlaubt. Zentrum der Drehung ist die X/Y-Position. Zeiger Ende muss so gesetzt werden, dass es im Uhrzeigersinn betrachtet rechts von Zeiger Anfang zu liegen kommt. Der Winkel hat einen Wertebereich von 0° bis maximal 350°.

Skala

Unterskalaposition	Die Unterskala kann am äußeren oder inneren Radius des Skalenrings angezeigt werden: • Außen • Innen
Skalentyp	Die Skala kann angezeigt werden als: • Linien • Punkte • Quadrate
Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala – Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenlinienstärke	Dicke der Skalenlinie in Pixel
Skalenfarbe	Die Farbe kann über die Selektionsliste bzw. über die Schaltfläche  aus dem Standarddialog zur Farbauswahl eingestellt werden.
Skala in 3D	Wenn diese Option aktiviert ist, wird die Skala 3-dimensional dargestellt.
Skala anzeigen	Wenn diese Option aktiviert ist, wird die Skala dargestellt.
Innerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring innen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.
Äußerer Rahmen	Wenn diese Option aktiviert ist, wird der Skalenring außen mit einem Rahmen versehen. Diese Option ist standardmäßig deaktiviert.

Beschriftung

Beschriftung	Hier kann eingestellt werden, ob die Skalenwerte auf der Außenseite oder Innenseite der Skala platziert werden sollen.
Einheit	Der eingegebene Text beschriftet das Element. Er wird unterhalb des Mittelpunkts der Skalierung angezeigt. Auf diese Weise kann beispielsweise die Skalierungseinheit angegeben werden.
Schriftart	Hier ist die Schriftart für Einheit und Skalierung festzulegen: • Standard • Überschrift • Große • Titel • Anmerkung Mit Klick auf die Schaltfläche … öffnet ein Dialog für benutzerdefinierte Einstellungen der Schrifteigenschaften.
Skalenformat (C-Syntax)	Benutzen Sie die C-Syntax zur Angabe der Formatierung der Skalierungsbeschriftung. So führt die Eingabe des Strings "%3.2f s" in diesem Feld zur Anzeige der Skalierungsbeschriftungen mit 3 Stellen, davon 2 Nachkommastellen, gefolgt vom Buchstaben "s".
Max. Breite der Beschriftungen	Wert, der die maximale Breite der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Höhe der Beschriftungen	Wert, der die Höhe der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Farbe Schriftart	Die Farbe für die Schrift kann über Selektionsliste bzw. über Schaltfläche … aus dem Standarddialog zur Farbauswahl eingestellt werden.

Positionierung

Zeigerabstand	Länge des Zeigers in Pixel
Skalenabstand	Abstand der Skalenstriche zum Zentrum in Pixel. Diese Eigenschaft ist nur verfügbar, wenn ein eigenes Hintergrundbild definiert worden ist.
Skalenlänge	Länge der Skalenstriche in Pixel. Diese Eigenschaft ist nur verfügbar, wenn ein eigenes Hintergrundbild definiert worden ist.
Beschriftungsverschiebung	Abstand in Pixel, um die Beschriftung zu positionieren
Einheitenverschiebung	Vertikaler Abstand in Pixel, um den Text (eingegeben in Beschriftung → Einheit) zu positionieren
Ursprungsverschiebung	Versatz des Elements. Er kann verwendet werden, um eine exakte Positionierung relativ zum Hintergrundbild zu erreichen.

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die Schaltfläche … geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbereiche

Dauerhafte Farbbereiche	Wenn diese Option aktiviert ist, sind die Farbbereiche dauerhaft sichtbar. Die Auswirkungen dieser Option sind nur im Onlinebetrieb sichtbar.
[<n>]	Mit einem Klick auf die Schaltfläche  wird ein neuer Farbbereich erzeugt. Für jeden Farbbereich wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf ‚Löschen' wird der zugehörige Farbbereich mit seinen Einstellungen gelöscht.

Bereich [<n>]

Bereichsanfang	Anfang des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Bereichsende	Ende des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Farbe	Farbe des Balkenbereichs

Transparenz wird unter Windows CE nicht unterstützt.

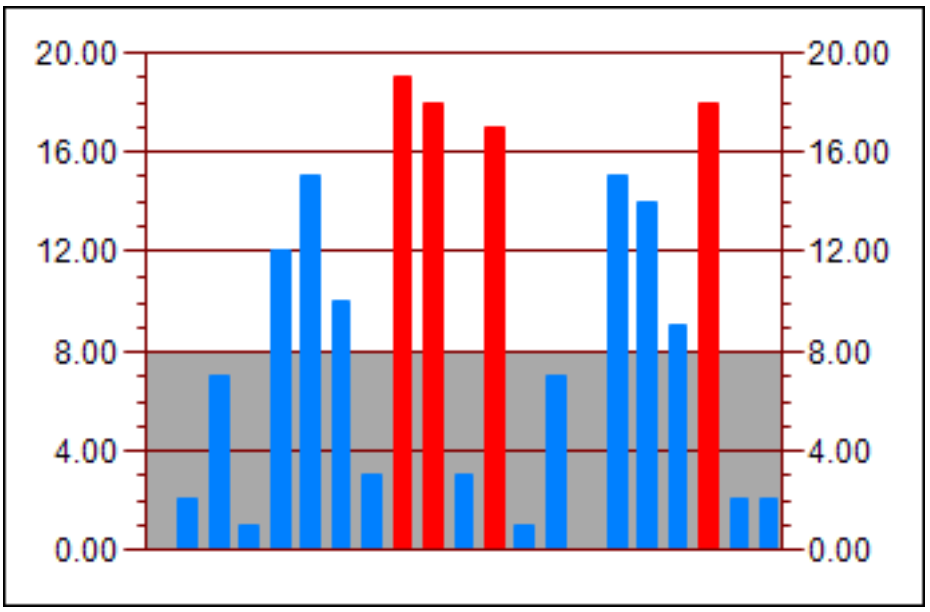
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.5.6 Histogramm

Mit diesem Element kann zu einer Visualisierungsseite ein Histogramm hinzugefügt werden, um Werte eines Arrays darzustellen. Der minimale und maximale Anzeigewert kann angegeben werden. Für bestimmte Wertebereiche können spezielle Farben [▶ 608] definiert werden. Ein Beispiel für die Konfiguration ist im Abschnitt "Konfiguration eines Histogramm [▶ 608]s" zu finden.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Datenarray	Variable vom Typ Array, deren Daten visualisiert werden soll.

Unterbereich des Arrays

Unterbereich verwenden	Wenn diese Option aktiviert ist, wird nur der Unterbereich des zugewiesen Datenarrays verwendet
Startindex	Indiziert den ersten Datensatz des zugewiesenen Datenarrays.
Endindex	Indiziert den letzten Datensatz des zugewiesenen Datenarrays.

Darstellungstyp	Hier kann der Darstellungstyp der Daten im Histogramm ausgewählt werden: • Balken • Linien • Kurve
Linienstärke	Stärke der Kurve, durch die die Daten im Histogramm dargestellt werden - Diese Einstellung ist nur dann verfügbar, wenn im Darstellungstyp "Kurve" ausgewählt worden ist.
Horizontale Linien anzeigen	Wenn diese Option aktiviert ist, werden horizontale Linien an den Hauptskalen gezeichnet.
Relative Balkenbreite	Relative Breite des Balkens bzw. der Linie in Pixeln

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Skala

Skalenanfang	Wert, der die Skalierungsanzeige von unten begrenzt
Skalenende	Wert, der die Skalierungsanzeige von oben begrenzt
Hauptskala	Abstand zwischen zwei Strichen der Grobskala
Unterskala	Abstand zwischen zwei Strichen der Feinskala. Der Wert kann auf 0 gesetzt werden, falls eine weitere Unterteilung der Grobskala nicht gewünscht ist.
Skalenfarbe	Die Farbe kann über die Selektionsliste bzw. über die Schaltfläche  aus dem Standarddialog zur Farbauswahl eingestellt werden.
Grundlinie	Numerischer Wert, um den sich die Grundlinie des Histogramms nach oben verschieben soll.

Beschriftung

Einheit	Der eingegebene Text beschriftet das Element. Er wird unterhalb des Mittelpunkts der Skalierung angezeigt. Auf diese Weise kann beispielsweise die Skalierungseinheit angegeben werden.
Schriftart	Hier ist die Schriftart für Einheit und Skalierung festzulegen: • Standard • Überschrift • Große • Titel • Anmerkung Mit Klick auf die Schaltfläche  öffnet ein Dialog für benutzerdefinierte Einstellungen der Schrifteigenschaften.
Skalenformat (C-Syntax)	Benutzen Sie die C-Syntax zur Angabe der Formatierung der Skalierungsbeschriftung. So führt die Eingabe des Strings "%3.2f s" in diesem Feld zur Anzeige der Skalierungsbeschriftungen mit 3 Stellen, davon 2 Nachkommastellen, gefolgt vom Buchstaben "s".
Max. Breite der Beschriftungen	Wert, der die maximale Breite der Skalierungsbeschriftung vorgibt– In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Höhe der Beschriftungen	Wert, der die Höhe der Skalierungsbeschriftung vorgibt – In der Regel wird dieser Wert automatisch richtig festgesetzt. Benutzen Sie diese Vorgabe nur, falls die automatische Anpassung nicht zum gewünschten Ergebnis führt.
Farbe Schriftart	Die Farbe für die Schrift kann über Selektionsliste bzw. über Schaltfläche  aus dem Standarddialog zur Farbauswahl eingestellt werden.

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die

Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Farbe des Graphen	Farbe für den Graphen
Alarmfarbe • Alarmwert • Alarmbedingung • Alarmfarbe	• Alarmwert Hier ist der Schwellenwert für den Alarm festzulegen. • Alarmbedingung Wenn der aktuelle Wert des Arrayelements die Alarmbedingung erfüllt, dann wird der Alarm gesetzt. Bei ‚Weniger' wird der Alarm ausgelöst, wenn der Wert kleiner als der Schwellenwert ist, und bei "Mehr" wird der Alarm ausgelöst, wenn der Wert größer als der Schwellenwert ist. • Alarmfarbe Hier kann eine Farbe ausgewählt werden, in der der einzelne Balken im Alarmfall dargestellt wird.
Farbbereiche verwenden	Wenn diese Option aktiviert ist, werden die spezifizierten Farbbereiche so wie unter Farbbereiche definiert dargestellt.
Farbbereiche • [<n>]	Mit einem Klick auf die Schaltfläche  wird ein neuer Farbbereich erzeugt. Für jeden Farbbereich wird ein Bereich angelegt, der die entsprechenden Einstellungen erfasst. [<n>]: Die Nummer indiziert den Bereich. Mit einem Klick auf "Löschen" wird der zugehörige Farbbereich mit seinen Einstellungen gelöscht.

Bereich [<n>]

Bereichsanfang	Anfang des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Bereichsende	Ende des Farbbereichs. Er muss innerhalb der definierten Skala [▶ 569] liegen.
Farbe	Farbe des Balkenbereichs

Transparenz wird unter Windows CE nicht unterstützt.

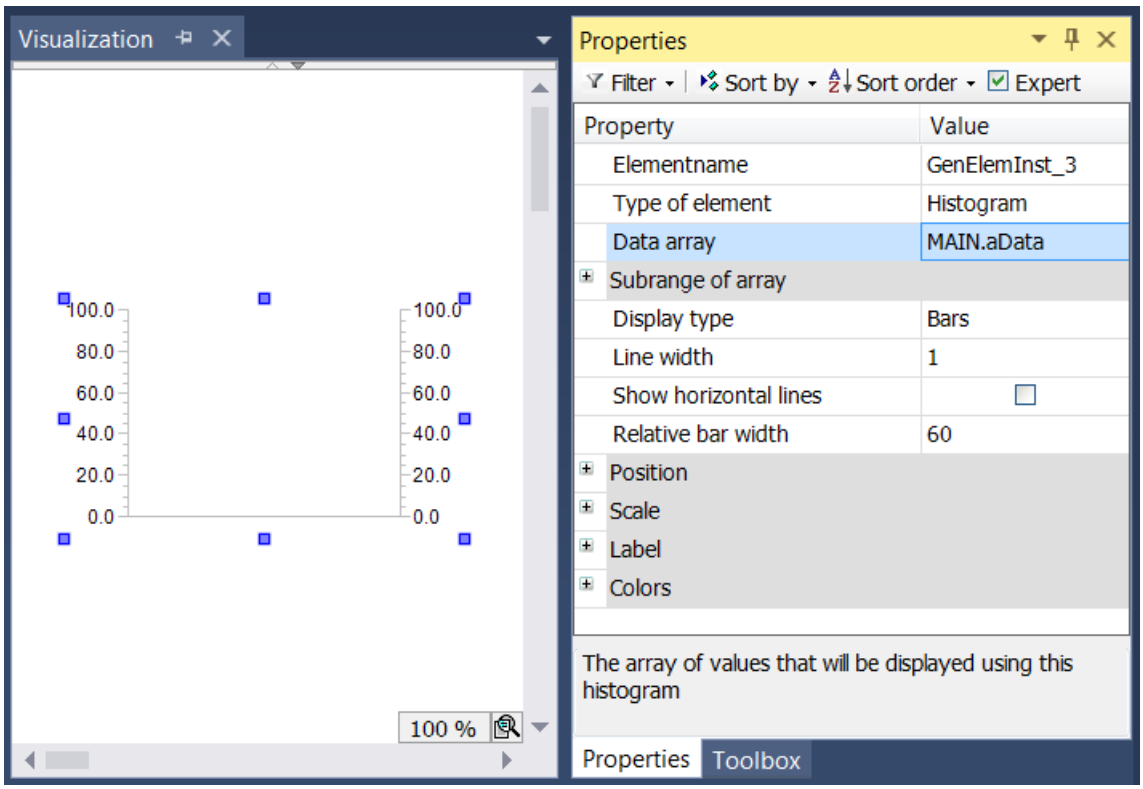
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

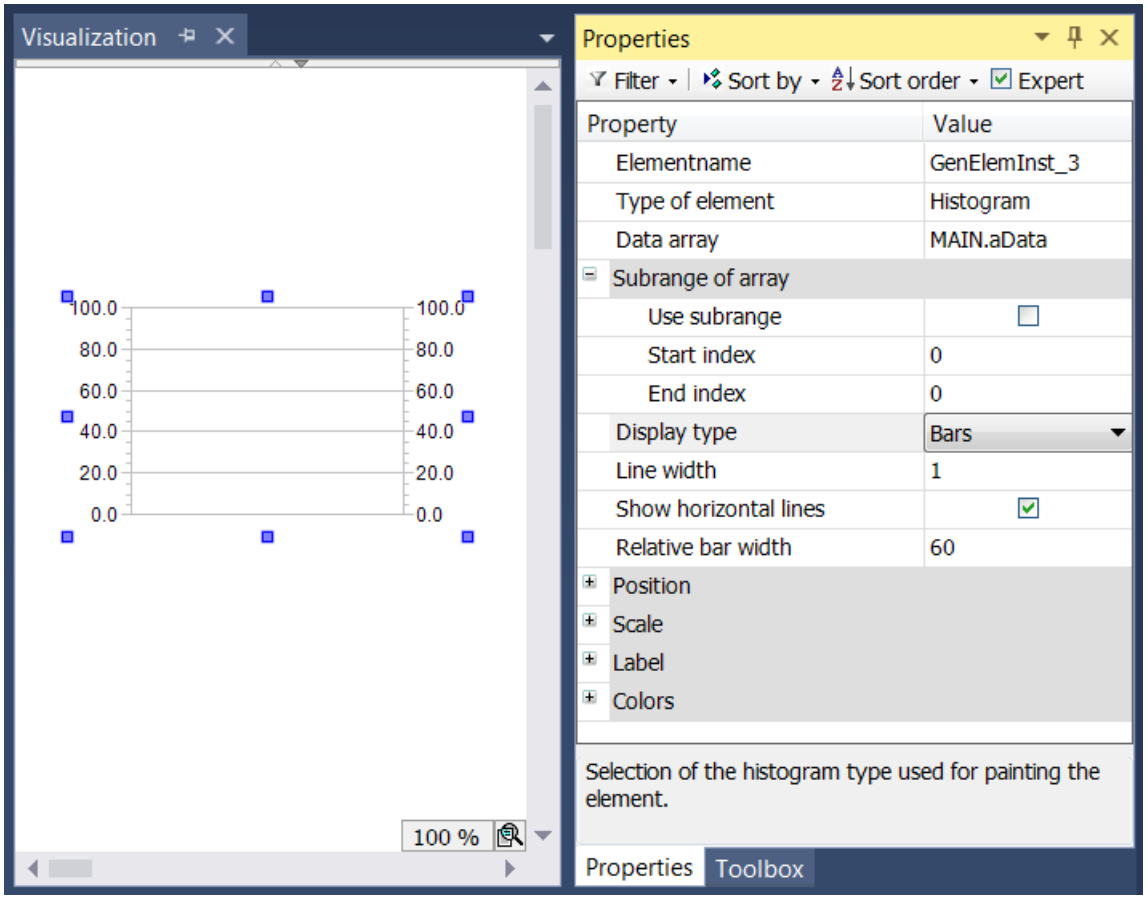
Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.5.6.1 Konfiguration eines Histogramms

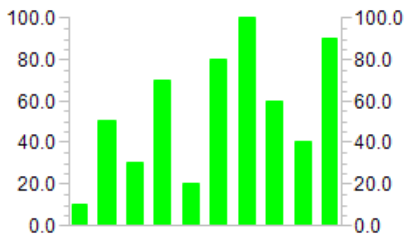
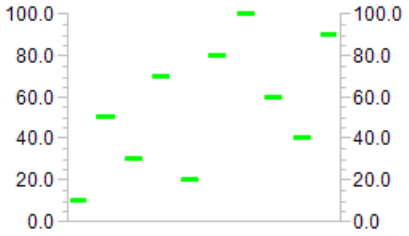
Im Folgenden wird ein Beispiel für die Konfiguration eines Histogramms erläutert.



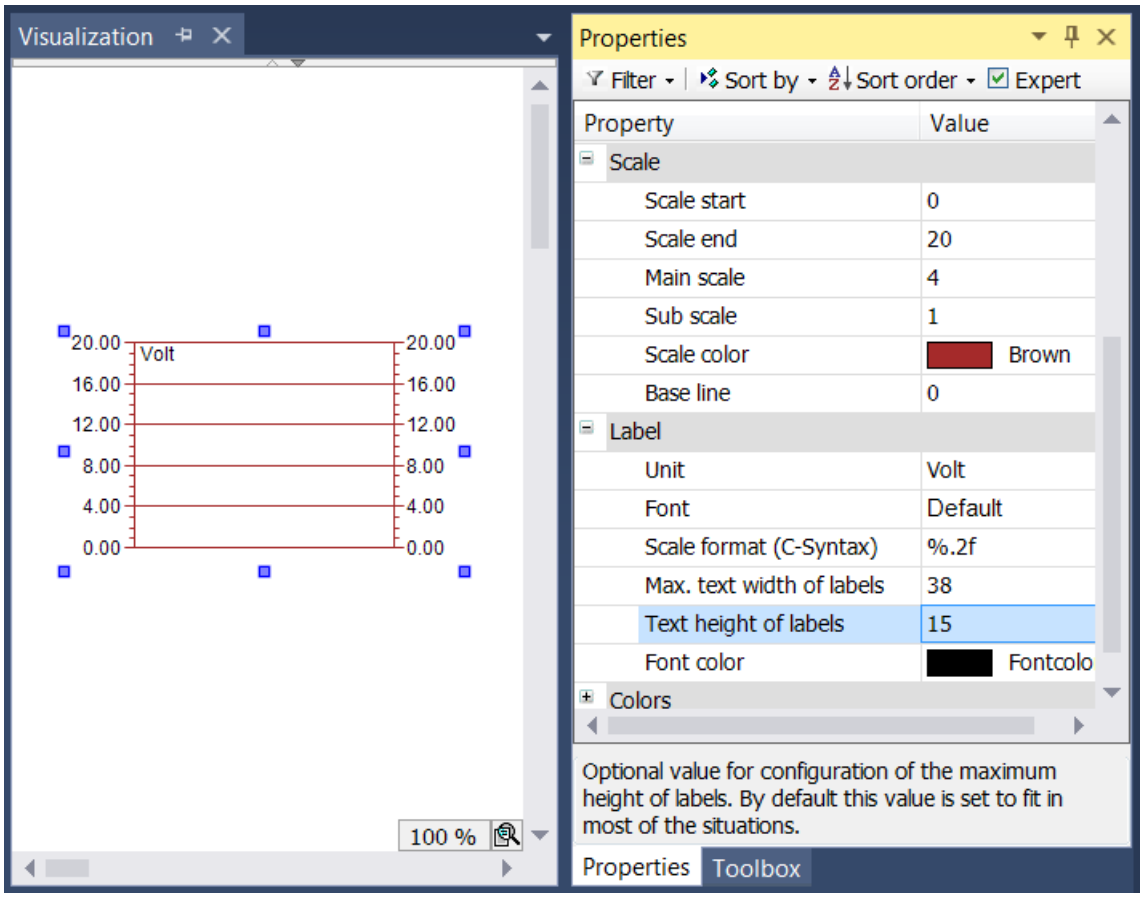
Das zugeordnete Datenarray – im Beispiel ein Array mit dem Namen "aData" – muss in den Elementeigenschaften des Histogramm-Elements angegeben werden. Nach einem Klick in das Eingabefeld der Eigenschaft "Datenarray" steht Ihnen die Schaltfläche [...] zur Verfügung. Sie kann benutzt werden, um im Projekt nach der Variablen zu suchen. Achten Sie darauf, die Eingangsvariable durch die Angabe ihres vollständigen Pfads im Gerätebaum zu spezifizieren.



Die Option "Unterbereiche verwenden" wird benutzt, wenn nicht alle Elemente des Arrays angezeigt werden sollen, sondern nur die Elemente von Index "Startindex" bis "Endindex". Zusätzlich kann der Darstellungstyp der Daten des Arrays in drei unterschiedlichen Formen dargestellt werden:

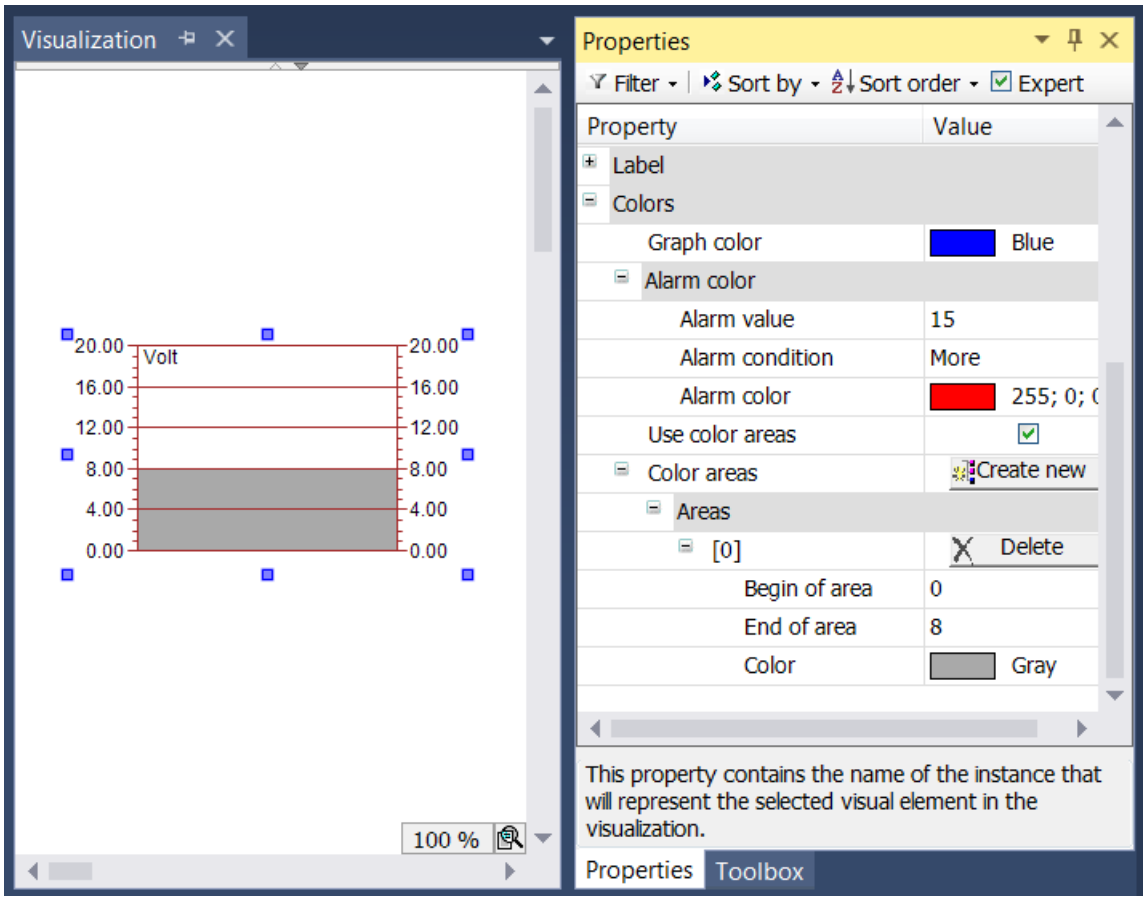
Balken	
Linien	
Kurve	

In diesem Beispiel Histogramm werden horizontale Linien an den Hauptskalen gezeichnet.



In den Eigenschaften der Skala kann der numerische Bereich der Skalierung und die Unterteilung durch Haupt- und Unterskala definiert werden. Der Wertebereich der Skala wird von unten durch den Wert in "Skalenanfang" und von oben durch den Wert in "Skalenende" begrenzt. Der Wert von "Skalenanfang" muss also geringer sein als der von "Skalenende". Beide Distanzangaben in Hauptskala wie Unterskala können auf 0 gesetzt werden, um eine Anzeige der Skalierungsstriche auszuschalten: Wird der Wert der Grobskalierung (Hauptskala) auf 0 gesetzt, so werden unabhängig vom Wert der Feinskalierung keinerlei Skalierungsstriche gezeichnet. Wird die Feinskalierung (Unterskala) auf 0 gesetzt, so werden nur die Skalierungsstriche der groben Unterteilung gezeichnet. Um die Skalenfarbe zu ändern, muss auf das Feld der Skalenfarbe geklickt werden – es öffnet sich ein Dialog, in welchem die Farbe ausgewählt werden kann.

Im Abschnitt Beschriftung kann die Darstellung der Skalierung festgelegt werden. Der Eintrag im Eingabefeld Einheit ist zur Angabe der Einheit der Skalierung gedacht und wird links oben im Histogramm angezeigt. Nach Wahl einer geeigneten Schrift(farbe) kann das Format der Beschriftung angepasst werden. Die Angabe der Formatierung für Zahlenwerte muss entsprechend der Syntax der Programmiersprache C vorgenommen werden. Benutzen Sie "%d" für ganzzahlige Wert und "%.Xf" für Gleitpunktzahlen, wobei "X" durch die gewünschte Anzahl an Nachkommastellen ersetzt werden muss.



Schließlich kann die Einfärbung des Elements innerhalb des Abschnitts Alarmfarbe gesetzt werden. Zuerst wird die Farbe des Histogramms ausgewählt. Innerhalb des Unterbereichs "Alarmfarbe" kann eine Alarmfarbe definiert werden, in welcher der einzelne Balken angezeigt wird, falls der aktuelle Wert des Array-Elements die Alarmbedingung erfüllt. Diese Bedingung ergibt sich aus der Definition des Alarmwertes, welcher entweder nicht überschritten (beim Setzen der Alarmbedingung auf "Mehr") oder nicht unterschritten (beim Setzen der Alarmbedingung auf "Weniger") werden darf.

Teilbereiche der Skala kann dann eine bestimmte Farbe zugewiesen werden, indem für sie mithilfe der Schaltfläche 

ein Farbbereich erzeugt wird. Die Farbbereiche werden aufsteigend nummeriert. Jeder Farbbereich wird innerhalb der Elementeigenschaften mit eigenen Eingabefeldern ausgestattet.

In "Bereichsanfang" und "Bereichsende" können die Grenzen des Unterbereichs festgelegt werden. Seine Einfärbung kann aus dem Aufklappmenü ausgewählt werden. Ein einmal angelegter Farbbereich kann durch einen Klick auf die zugehörige Schaltfläche 

gelöscht werden.

15.8.6 Spezielle Steuerelemente

15.8.6.1 Wartesymbol Blume

Dieses animierte Element kann verwendet werden, um zu signalisieren, dass das System beschäftigt ist oder auf Daten wartet.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Farben

Eine Farbe wird mittels Hexadezimalzahl definiert, die sich aus Rot/ Grün/ Blau (RGB) Anteilen zusammensetzt. Für jede dieser drei Farben sind 256 (0-255) Werte verfügbar, die hier statisch eingetragen werden können. Die Farbe kann aus einer Auswahlliste oder dem Farbauswahldialog, der über die

Schaltfläche  geöffnet werden kann, gewählt werden. Zusätzlich kann für jede Farbe der Grad der Transparenz (0: voll transparent, 255: voll deckend) eingestellt werden.

Rahmenfarbe	Auszuwählen ist eine Rahmenfarbe für das Element.
Füllfarbe	Auszuwählen ist eine Füllfarbe für das Element.

Transparenz wird unter Windows CE nicht unterstützt.

Aussehen

Die Einstellungen unter Aussehen sind statische Definitionen bezüglich der Rahmenlinie und der Füllung des Elements.

Linienstärke	Definiert die Stärke der Rahmenlinie in Pixel. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel. Wird kein Rahmen gewünscht, müssen Sie die Linienart auf unsichtbar setzen.
Füllart	Definiert die Art der Füllung für die Füllfarbe: • Ausgefüllt: Die Füllfarbe ist sichtbar • Unsichtbar: Die Füllfarbe ist nicht sichtbar
Linienart	Definiert eine der folgenden Linienarten für die Umrisslinie: • Durchgezogen • Striche • Punkte • Strich Punkt • Strich Punkt Punkt • Unsichtbar: Umrisslinie ist nicht sichtbar.

Symbolfarbe	Hier kann eine Farbe ausgewählt werden, mit der das Blumensymbol angezeigt werden soll.
Symbollinienstärke	Hier kann eine Zahl angegeben werden, die die Stärke der Symbollinien in Pixel angibt. 0 kodiert das Gleiche wie 1 und setzt die Linienstärke auf 1 Pixel.

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Boolesche Variable. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.

Zugriffsrechte

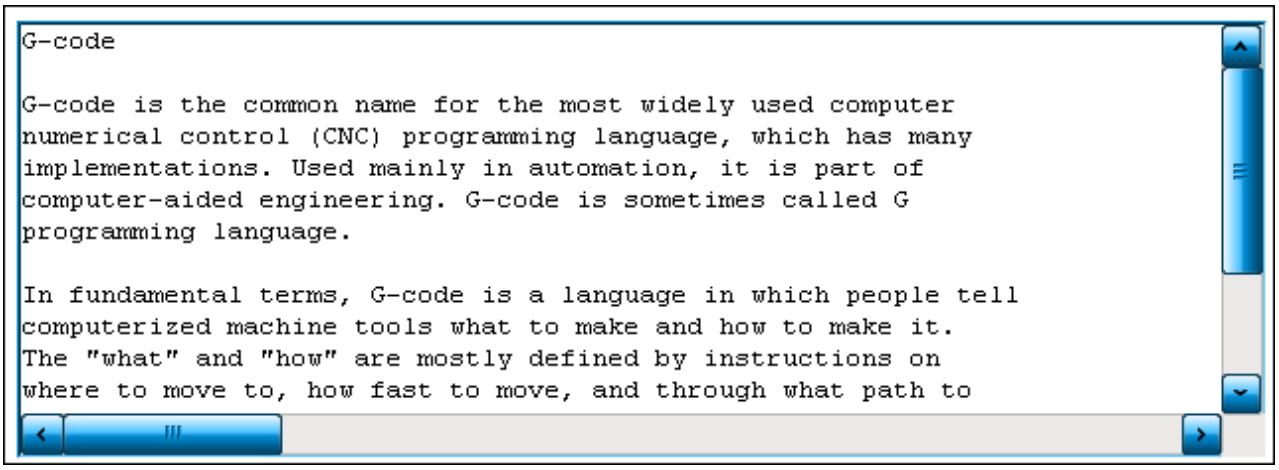
Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.6.2 Texteditor

Das Visualisierungselement Texteditor wird verwendet zum Anzeigen und Editieren des Inhalts von Textdateien in ASCII oder Unicode, die sich auf der Steuerung befinden.

- Benutzereingaben beim Navigieren [▶ 617]
- Beispiel für die Konfiguration [▶ 618]



Das Text-Editor-Element kann nur in Kombination mit der PLC HMI [▶ 635] und/oder der PLC HMI Web [▶ 640] verwendet werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche [...] die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Schriftart

Name	Name einer auf dem System installierten Schriftart, in der dann der Text im Texteditor ausgegeben wird. Dabei muss eine nicht proportionale Schriftart wie Courier New verwendet werden.
Größe	Größe des Schriftgrads Beispiel: 12

Steuervariablen

Datei

Dateiname	STRING-Variable, die den Dateinamen und, falls erforderlich, den Pfad enthält
Öffnen	Boolesche Variablen, um das Öffnen der in "Dateiname" bestimmten Datei zu steuern. Wird die Variable TRUE gesetzt, öffnet sich die Datei.
Schließen	Boolesche Variable, um das Schließen der Datei zu steuern. Wird die Variable TRUE gesetzt, schließt sich die Datei.
Speichern	Boolesche Variable, um das Speichern der Datei zu steuern. Wird die Variable TRUE gesetzt, wird die Datei gespeichert.
Neu	Boolesche Variable, um das Erzeugen einer neuen Datei zu steuern. Wird die Variable TRUE gesetzt, wird eine neue Datei mit dem in "Dateiname" bestimmten Namen erzeugt.

Bearbeiten

Suchen nach	STRING-Variable, die den Suchbegriff enthält
Finden	Boolesche Variable, die das Suchen nach dem Suchbegriff steuert. Wird die Variable TRUE gesetzt, startet das Suchen.
Nächstes Vorkommen finden	Boolesche Variable, die das Suchen nach dem nächsten Vorkommen des Suchbegriffs steuert. Wird die Variabel TRUE gesetzt, startet das Suchen an der aktuellen Position.

Caretposition

Zeile	Integer-Variable, die, solange "Setzen auslösen" FALSE ist, die aktuelle Zeilennummer enthält
Spalte	Integer-Variable, die, solange "Setzen auslösen" FALSE ist, die aktuelle Spaltennummer enthält
Position	Integer-Variable, die, solange "Setzen auslösen" FALSE ist, die aktuelle Position des Carets in fortlaufender Nummerierung enthält
Setzen auslösen	Boolesche Variable, die die Caretposition steuert. Ist die Variable FALSE, enthalten die Variablen in "Zeile", "Spalte" und "Position" die aktuelle Position. Ist die Variable TRUE, wird das Caret an die Position gesetzt, die in "Zeile" und "Spalte" angegeben ist.

Selektion

Startposition	Integer-Variable, die die Startposition der Textselektion in fortlaufender Nummerierung enthält
Endposition	Integer-Variable, die die Endposition der Textselektion in fortlaufender Nummerierung enthält
Start Zeilennummer	Integer-Variable, die die Zeilennummer am Anfang der Textselektion enthält
Start Spaltenindex	Integer-Variable, die den Spaltenindex am Anfang der Textselektion enthält
Ende Zeilennummer	Integer-Variable, die die Zeilennummer am Ende der Textselektion enthält
Ende Spaltenindex	Integer-Variable, die den Spaltenindex am Ende der Textselektion enthält
Zu selektierende Zeile	Integer-Variable, die eine zu selektierende Zeile bestimmt
Selektion auslösen	Boolesche Variable, um die Textselektion zu steuern. Wird die Variable TRUE gesetzt, wird die Textselektion, wie in "Zu selektierende Zeile" definiert, gesetzt.

Fehlerbehandlung

Variable für Fehlernummer	Integer-Variable, die die Fehlernummer im Fehlerfall enthält. Die Nummern sind in GVL_ErrorCodes, Teil der Bibliothek "VisuElemTextEditor", deklariert. Um den Fehlertext zur Fehlernummer zu erhalten, muss die Funktion VisuFctTextEditorGetErrorText() der Bibliothek aufgerufen werden.

Variable für Inhalt geändert	Boolesche Variable. Ist der Variablenwert TRUE, hat sich der Inhalt des Texteditors geändert.
Variable für Zugriffsmodus	Boolesche Variable. Ist der Variablenwert TRUE, hat die Datei nur Leserecht. Ist der Variablenwert FALSE, hat die Datei Schreib- und Leserecht.

Maximale Zeilenlänge	Integer-Variable, die die maximale Länge einer Zeile angibt
Editiermodus	Hier kann der Editiermodus ausgewählt werden: • Nur lesend Die Datei kann nur gelesen werden. In diesem Modus wird der Editor hellgrau unterlegt dargestellt, wenn die Steuerung läuft. • Lesen/schreiben Die Datei kann gelesen und beschrieben werden.

Neue Dateien

Zeichencodierung	Wird eine neue Datei erzeugt, ist hier die Zeichencodierung festzulegen: • ASCII • Unicode (Little endian) • Unicode (Big endian)
Zeilenendzeichen	Wird eine neue Datei erzeugt, ist hier das Zeilenendzeichen zu definieren: • CR/LF: üblich in Windows-Systemen • LF: üblich in Unix-Systemen Wird ein bestehendes File geöffnet, wird das Zeilenendzeichen der öffnenden Datei identifiziert und automatisch verwendet.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.6.2.1 Benutzereingaben beim Navigieren im Online-Modus

Der Texteditor erlaubt allgemein bekannte Benutzereingaben, um im Text zu navigieren. Diese sind nur im Onlinebetrieb verfügbar.

[Pfeil links]	Das Caret wird um ein Zeichen nach links verschoben. Ist die aktuelle Caretposition ein Zeilenanfang, wird es an das Ende der vorigen Zeile gesetzt, falls es diese gibt.
[Pfeil rechts]	Das Caret wird um ein Zeichen nach rechts verschoben. Ist die aktuelle Caretposition ein Zeilenende, wird es an den Anfang der nächsten Zeile gesetzt, falls es diese gibt.
[Pfeil aufwärts]	Das Caret wird in die vorige Zeile verschoben.
[Pfeil abwärts]	Das Caret wird in die nächste Zeile verschoben.
[Pos1]	Das Caret wird an den Anfang der aktuellen Zeile verschoben.
[Ende]	Das Caret wird an das Ende der aktuellen Zeile verschoben.
[Bild]	Die vorige Seite wird angezeigt.
[Bild ¯]	Die nächste Seite wird angezeigt.
[Umschalttaste] + [Pfeil links] [Umschalttaste] + [Pfeil rechts]	Textselektion
[Entf] bei selektiertem Text	Der selektierte Text wird entfernt. Ist nichts selektiert, wird das Zeichen rechts der Caretposition gelöscht.
[Eingabe]	Eine neue Zeile wird erzeugt und das Caret wird an deren Anfang gesetzt.
[Strg] + [Tab]	Ein Tabulatorzeichen wird eingefügt. Aktuell wird es als einzelnes Blank dargestellt.
[Tab]	Der Fokus wird zur nächsten Visualisierung verschoben.

15.8.6.2.2 Konfiguration eines Texteditors

Um Zugriff auf die vom Texteditor [▶ 614] unterstützten Steuervariablen zu erhalten, ist es notwendig, ein zusätzliches Visualisierungselement der Visualisierung hinzuzufügen und es mit dem Texteditor über Steuervariablen zu verbinden.

Im Folgenden soll am Beispiel der Ladefunktionalität des Texteditors das Vorgehen beschrieben werden:

1. Deklarieren der Steuervariablen des Elements "Laden" in IEC-Code in zum Beispiel MAIN:

```
PROGRAMM MAIN
VAR
    bOpen : BOOL;
END_VAR
```

2. Deklarieren der Variablen für Dateinamen des Texteditors in IEC-Code in zum Beispiel MAIN:

```
PROGRAMM MAIN
VAR
    bOpen : BOOL;
    sFileName : STRING;
END_VAR
```

3. Hinzufügen eines Rechteckelements zur Visualisierung und dessen Konfiguration:

- Texte → Text : Load
- Inputkonfiguration → Tasten → Variable : MAIN.bOpen

4. Hinzufügen eines Texteditorelements zur Visualisierung und dessen Konfiguration:

- Steuervariablen → Datei → Dateinamen : MAIN.sFileName
- Steuervariablen → Datei → Öffnen : MAIN.bOpen

Alle Steuervariablen, die in den Eigenschaften des Texteditors vorhanden sind, können so angebunden werden.

Fehlermeldungen

Um den Fehlertext der Fehlernummer, die in Steuervariablen → Fehlerbehandlung → Variable für Fehlernummer zur Verfügung steht, auszugeben, ist der Funktionsaufruf VisuFctTextEditorGetErrorText() in IEC-Code zu programmieren:

1. Deklarieren der erforderlichen Variablen:

```
PROGRAMM MAIN
VAR
…
nErrorCode : USINT;
sErrorMessage : STRING(255);
END_VAR
```

2. Implementieren des Funktionsaufrufs:

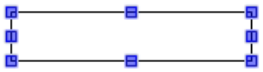
```
sErrorMessage := VisuFctTextEditorGetErrorText(nErrorCode);
```

3. Ausgabe des Fehlertextes in der Visualisierung zum Beispiel in einem Rechteckelement mit folgenden Eigenschaften:

- Texte → Text : %s
- Textvariablen → Textvariable : MAIN.sErrorMessage

15.8.6.3 ActiveX-Element

Das ActiveX-Element erlaubt es, eine existierende ActiveX-Komponente zu integrieren.



Das ActiveX-Control kann nur in Kombination mit der integrierten Visualisierung [▶ 634] und nicht in Kombination mit der PLC HMI [▶ 635] und der PLC HMI Web [▶ 640] verwendet werden.

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Element	Hier kann eine installierte ActiveX-Komponente zugewiesen werden. Die Schaltfläche ... verwenden, wenn eine Komponente mithilfe des Eingabeassistenten ausgewählt werden soll. Eine Variable vom Typ String eingeben, die den Namen oder die ID der Komponente beschreibt.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Boolesche Variable. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.

Initiale Aufrufe

Methodenaufrufe, die während der Initialisierung ausgeführt werden, können unter dieser Eigenschaft definiert werden. Sie werden ausschließlich im ersten Zyklus ausgeführt.

Mit der Schaltfläche Create new können neue Methodenaufrufe und dazugehörige Parameter erstellt werden. Über die Schaltfläche Delete kann der entsprechende Methodenaufruf mit allen Einstellungen, wie zum Beispiel Parameter und Ergebnisparameter, gelöscht werden.

[<Nummer>]	Nummer, die einen Methodenaufruf indiziert
Methode	Name der aufzurufenden Methode
Parameter	Variablen der Parameter
Ergebnisparameter	Variable, die das Ergebnis enthalten soll. Jeder Methode steht grundsätzlich ein Ergebnisparameter zur Verfügung.

Zyklische Aufrufe

Methodenaufrufe, die in jedem Zyklus ausgeführt werden, können unter dieser Eigenschaft definiert werden. Sie werden im Aktualisierungsintervall der Visualisierung aufgerufen.

Mit der Schaltfläche Create new können neue Methodenaufrufe und dazugehörige Parameter erstellt werden. Über die Schaltfläche Delete kann der entsprechende Methodenaufruf mit allen Einstellungen, wie zum Beispiel Parameter und Ergebnisparameter, gelöscht werden.

[<Nummer>]	Nummer, die einen Methodenaufruf indiziert
Methode	Name der aufzurufenden Methode
Parameter	Variablen der Parameter
Ergebnisparameter	Variable, die das Ergebnis enthalten soll. Jeder Methode steht grundsätzlich ein Ergebnisparameter zur Verfügung.

Bedingte Aufrufe

Methodenaufrufe, die nur bedingt ausgeführt werden sollen, können unter dieser Eigenschaft definiert werden. Sie werden im Aktualisierungsintervall der Visualisierung aufgerufen. Anders als beim zyklischen Aufrufen wird eine Aufrufbedingung unter der Eigenschaft "Aufrufbedingung" definiert. Sie werden nur bei steigender Flanke der Aufrufbedingung ausgeführt.

Mit der Schaltfläche Create new können neue Methodenaufrufe und dazugehörige Parameter erstellt werden. Über die Schaltfläche Delete kann der entsprechende Methodenaufruf mit allen Einstellungen, wie zum Beispiel Parameter und Ergebnisparameter, gelöscht werden.

[<Nummer>]	Nummer, die einen Methodenaufruf indiziert
Methode	Name der aufzurufenden Methode
Aufrufbedingung	Boolesche Variable. Ausschließlich bei steigender Flanke dieser Variablen wird die zugewiesene Methode einmal aufgerufen.
Parameter	Variablen der Parameter
Ergebnisparameter	Variable, die das Ergebnis enthalten soll. Jeder Methode steht grundsätzlich ein Ergebnisparameter zur Verfügung.

Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.6.4 Webbrowser

Das Webbrowser-Element zeigt einen Inhalt an, der über eine URL Adresse festgelegt ist. Es kann neben HTML-Seiten auch Videos und PDF-Dokumente laden. Ein Beispiel für die Konfiguration finden Sie im Abschnitt "Konfiguration [▶ 623]".

Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Absolute Bewegung

Das Element kann bewegt werden, indem die X- und Y-Position (Pixels) der linken oberen Ecke des Elements durch eine Integer-Variable verändert werden. Hier werden absolute Koordinatenwerte verwendet.

Bewegung • X • Y	X: Die hier eingetragene Integer-Variable definiert die aktuelle X-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in X-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von links nach rechts) Y: Die hier eingetragene Integer-Variable definiert die aktuelle Y-Position der linken oberen Ecke des Elements (in Pixel). Sie kann verwendet werden, um das Element in Y-Richtung zu verschieben. (Ein positiver Wert verschiebt das Element von oben nach unten)

Zustandsvariablen

Dies sind dynamische Definitionen zur Verfügbarkeit des Elements im Onlinebetrieb.

Unsichtbarkeit	Boolesche Variable. Wenn diese TRUE liefert, ist das Element im Onlinebetrieb unsichtbar.

Steuervariablen

URL	Variable vom Typ String, in der die URL Adresse der zu öffnenden Internetseite gespeichert ist Beispiel: sUrlAddress : STRING := ,'http://www.beckhoff.com/';
Anzeigen	Boolesche Variable. Wenn die Variable eine steigende Flanke enthält, ruft die Visualisierung die in URL angegebene Internetseite auf und stellt dessen Inhalt im Rahmen des Elements dar.
Zurück	Boolesche Variable. Wenn die Variable eine steigende Flanke enthält, stellt die Visualisierung den Inhalt der vorher dargestellten Seite dar.
Vorwärts	Boolesche Variable. Wenn die Variable eine steigende Flanke enthält, stellt die Visualisierung den Inhalt vor dem Zurücknavigieren dar.

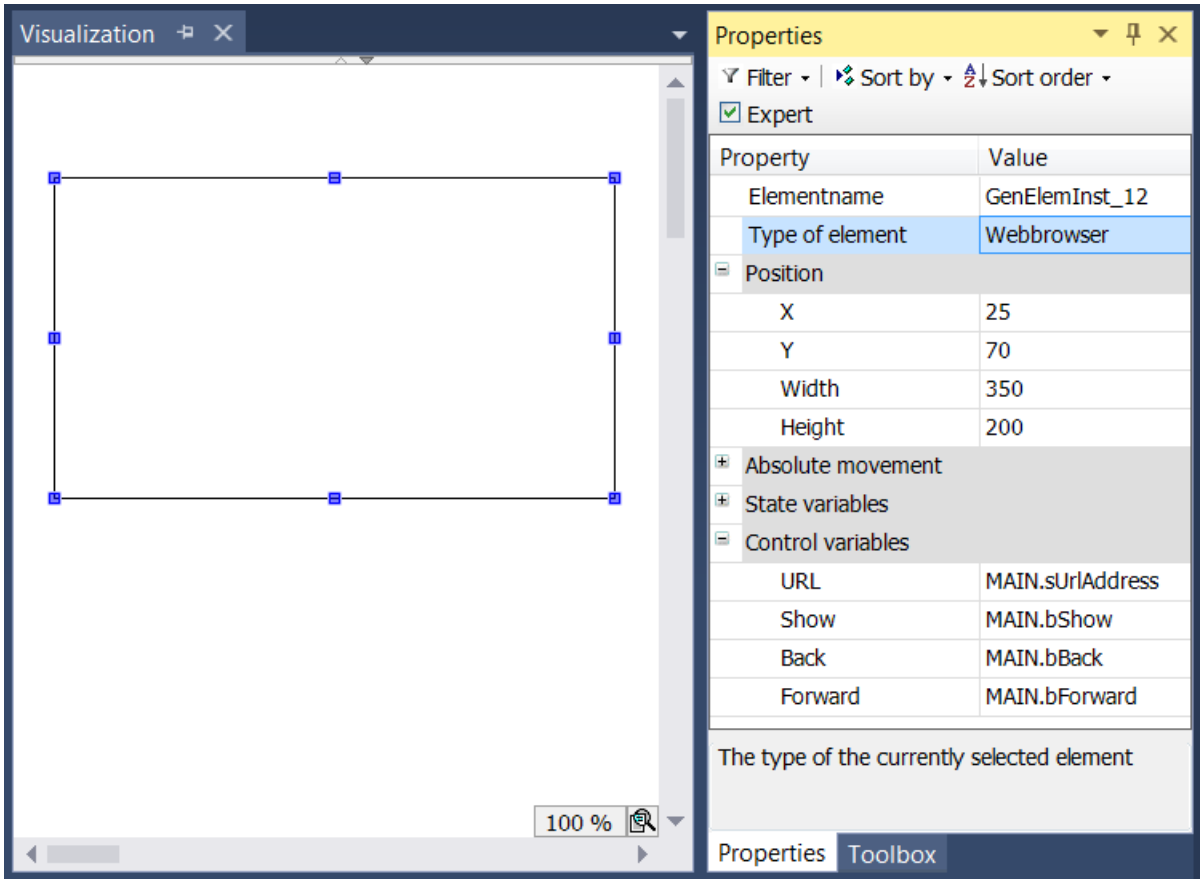
Zugriffsrechte

Bei dieser Einstellung handelt es sich um die Zugriffsrechte, die für das einzelne Element gelten. Mit einem Mausklick kann der Zugriffsrechtedialog [▶ 440] geöffnet werden. Die Einstellung ist nur dann möglich, wenn eine Benutzerverwaltung [▶ 412] zum SPS-Projekt hinzugefügt worden ist. Es existieren folgende Statusmeldungen:

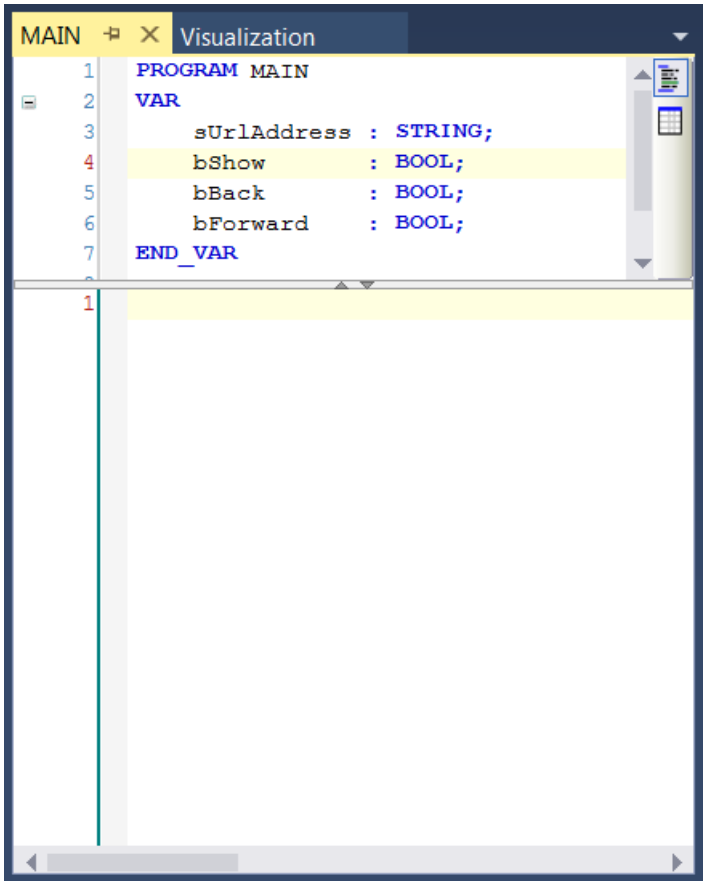
Nicht gesetzt. Alle Rechte.	Die Standardmeldung ist gesetzt, wenn das Element für alle Gruppen bedienbar angezeigt wird.
Rechte sind vergeben: Eingeschränkte Rechte.	Die Meldung ist gesetzt, wenn das Element für mindestens eine Gruppe mit eingeschränktem Verhalten angezeigt wird.

15.8.6.4.1 Konfiguration

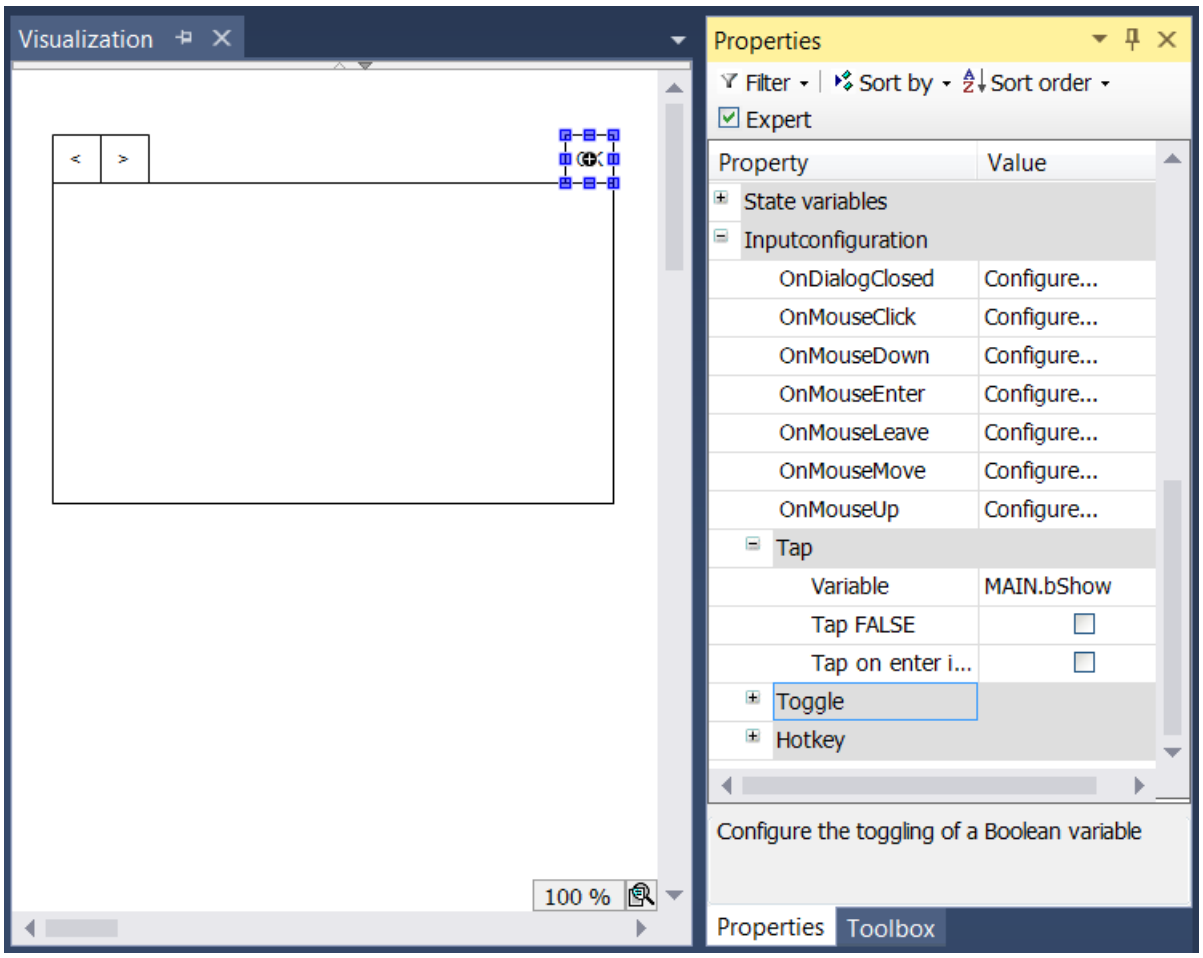
Im Folgenden wird ein Beispiel für die Konfiguration eines Webbrowser-Elements erläutert.



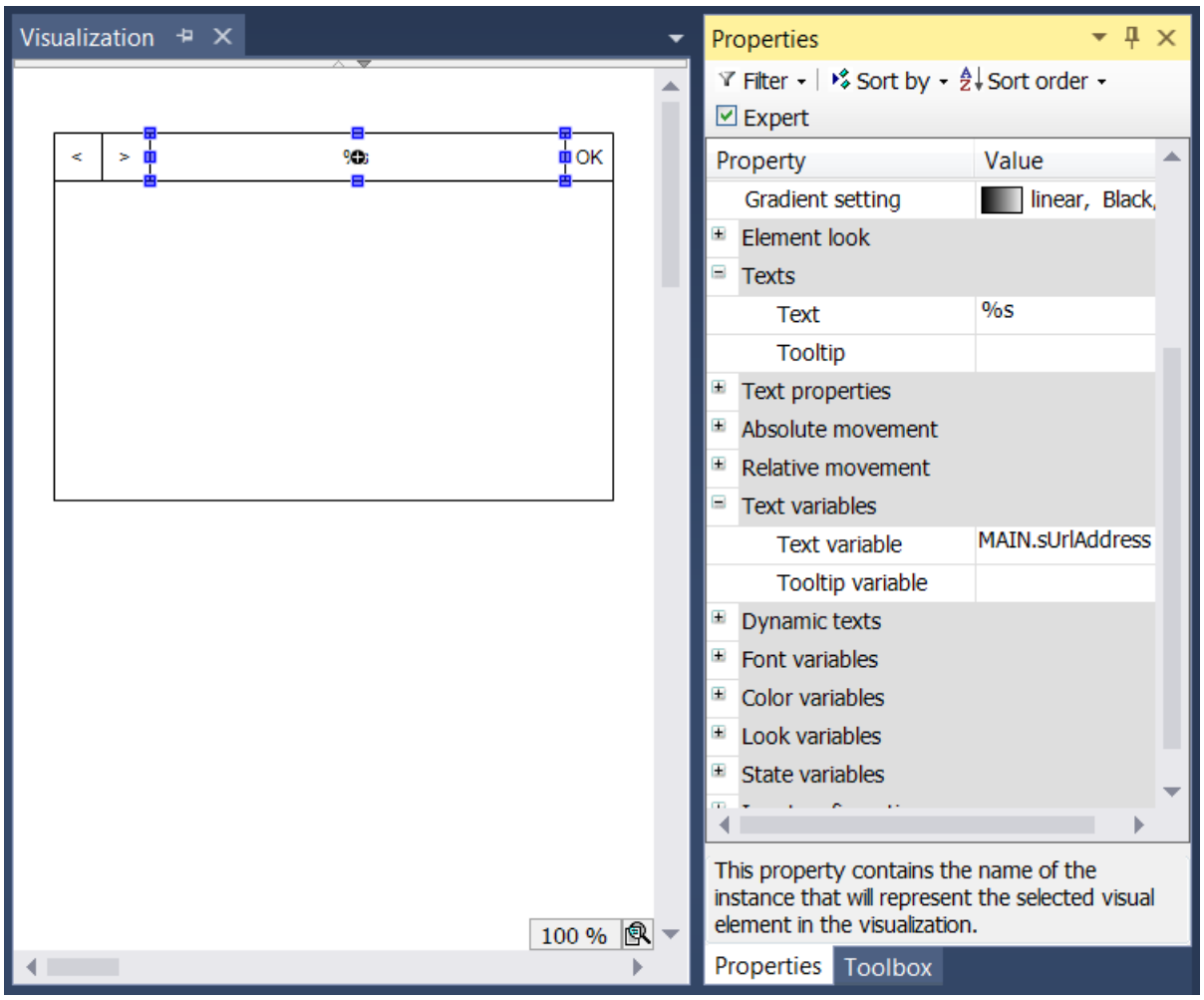
Nachdem das Webbrowser-Element aus dem Werkzeugkasten auf die Visualisierungsseite [▶ 422] gezogen worden ist, können die Position und Größe des Elements über die Eigenschaft "Position" verändert werden. In den "Steuervariablen [▶ 623]" können zudem Variablen für die Bedienung des Browser angegeben werden. In diesem Beispiel wurden die Variablen "sUrlAddress", "bShow", "bBack" und "bForward" eingetragen, die in dem Programm "MAIN" deklariert worden sind:



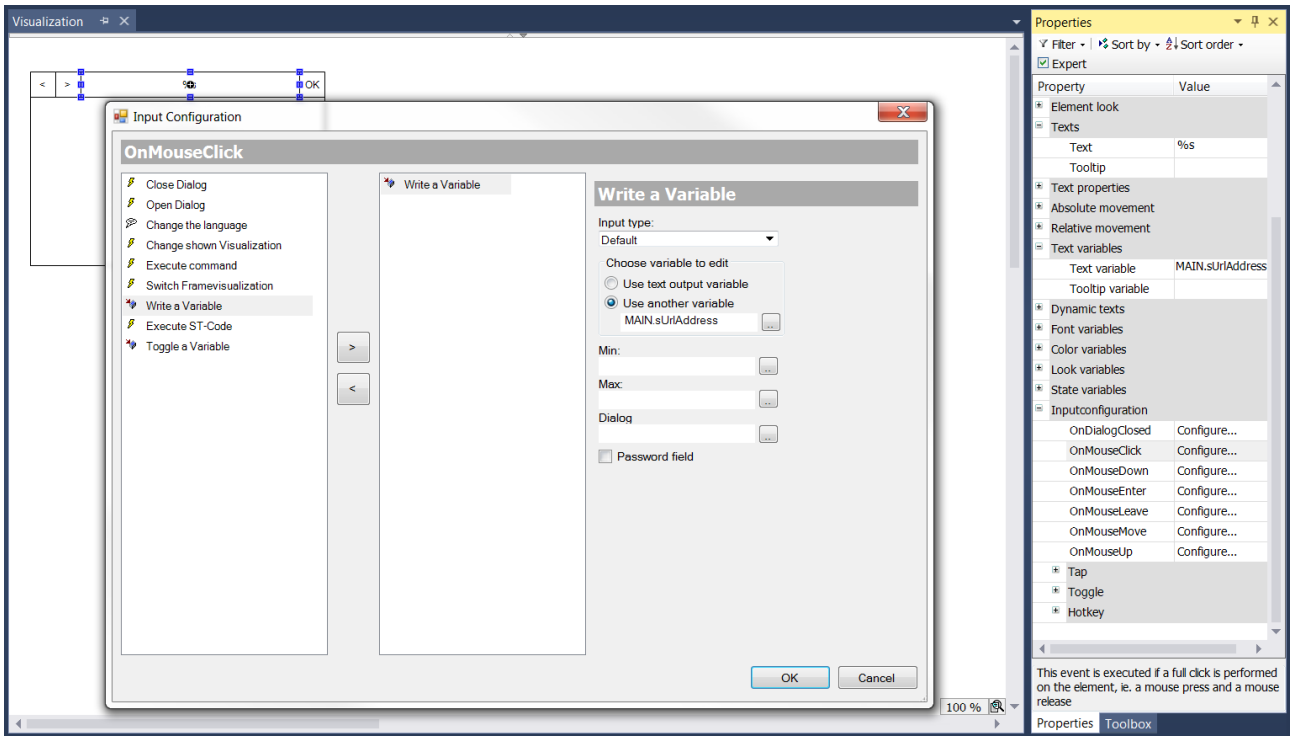
Um die im Programm "MAIN" deklarierten Variablen aus der Visualisierung verändern zu können, werden zunächst drei Rechteck-Elemente auf die Visualisierung gezogen. Das erste Element mit dem Text "<" wird in den Eingabekonfiguration für die Umschaltfunktion mit "bBack" verbunden, das Zweite mit dem Text ">" mit "bForward" und das Dritte mit dem Text "OK" mit "bShow".



Dann wird ein viertes Rechteck-Element hinzugefügt, um die URL Adresse des Webbrowsers ändern zu können. Damit das Element die Adresse anzeigen kann, wird der Platzhalter "%s" in der Eigenschaft "Text" und "sUrlAddress" in "Textvariable" eingetragen.

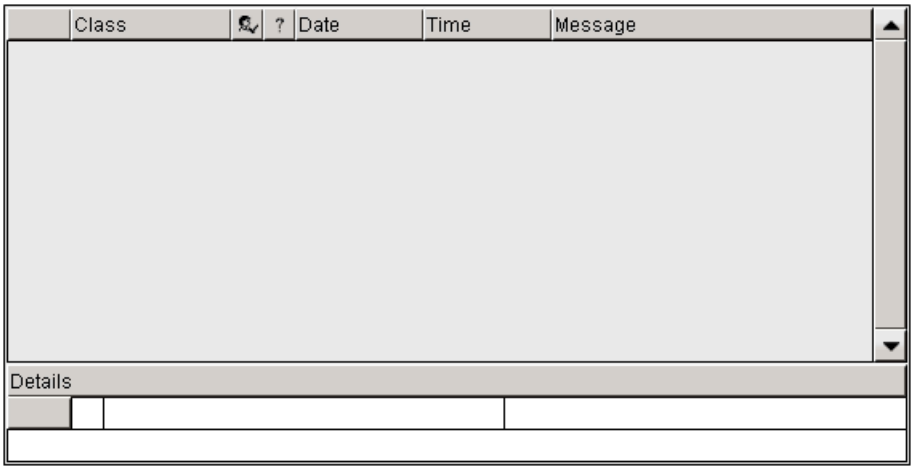


Um auch die URL Adresse in der Visualisierung ändern zu können, wird in den Eingabekonfigurationen das "OnMouseClick" Event genutzt. Hier wird eine "Variable schreiben" Aktion hinzugefügt und mit der Variable "sUrlAddress" verbunden.



15.8.6.5 Event-Tabelle

Mit der Event-Tabelle Element können Meldungen des TcEventLoggers in Form einer Tabelle auf einer Visualisierungsseite angezeigt werden. Die Meldungen werden mithilfe des Funktionsbausteins "FB_AdsReadEvents [▶ 629]" ausgelesen und über ein Array an das Visualisierungselement weitergegeben. Ein Beispiel für die Konfiguration des Elements ist im Abschnitt "Konfiguration der Event-Tabelle [▶ 630]" zu finden. Dieses Element ist ab dem Build 4020.0 verfügbar.



Eigenschafteneditor

Die Eigenschaften eines Visualisierungselementes - mit Ausnahme von Ausrichtung und Reihenfolge [▶ 394] - können alle im Eigenschafteneditor [▶ 403] konfiguriert werden. Dieser öffnet standardmäßig neben dem Visualisierungseditor oder wird explizit über den Befehl "Eigenschaften" (standardmäßig im Ansicht-Menü) geöffnet.

Eine Eigenschaft kann durch Bearbeiten des Feldes "Wert" verändert werden. Dazu wird in diesem Feld abhängig vom Elementtyp ein Eingabefeld, eine Auswahlliste, ein Dialog oder eine zu aktivierende Checkbox bereitgestellt. Das Wertefeld wird geöffnet

- nach einem Doppelklick,
- nach einem Einzelklick in ein selektiertes Feld,
- über die Leertaste, wenn das Feld bereits selektiert worden ist.

Wenn eine Variable zugewiesen wird,

- geben Sie einfach deren Namen ein.
- öffnen Sie mit der Schaltfläche  die Eingabehilfe, um eine Variable auszuwählen. Die Kategorie Variablen listet alle Variablen auf, die im Projekt bisher definiert wurden.

Das Arbeiten in der Liste der Eigenschaften kann mithilfe von Standard-, Sortier- und Filterfunktionen erleichtert werden.

Elementeigenschaften

Im Folgenden werden alle Elementeigenschaften und deren Beschreibungen aufgelistet.

Elementname	Der Elementname kann geändert werden. Standardname ist "GenElemInst_x". "x" steht für eine fortlaufende Nummerierung.
Elementtyp	Hier ist der Typ des Elements eingetragen. Bei drei Elementgruppen ist es möglich zwischen den zugehörigen Elementen zu wechseln, indem der Elementtyp geändert wird: • Rechteck, abgerundetes Rechteck und Ellipse [▶ 499] • Polygon, Linienzug oder Bézierkurve [▶ 516] • Hebelschalter, Energieschalter, Drückschalter, Drückschalter LED, Kippschalter [▶ 561]

Meldungsdatenarray	Hier die Ausgangsvariable "aEvents" des Funktionsbausteins "FB_AdsEventReader [▶ 629]" zuweisen, in welcher die aktiven Meldungen des TcEventLoggers gespeichert werden.

Position

Hier ist die Position (X/Y-Koordinaten) und Größe (Breite und Höhe) des Elements jeweils in Pixel zu definieren. Der Ursprung liegt in der oberen linken Fensterecke. Die positive X-Achse verläuft nach rechts, die positive Y-Achse verläuft nach unten. Werden die Werte editiert, wird gleichzeitig das angezeigte Element im Visualisierungseditor [▶ 394] geändert.

X	Horizontale Position in Pixel – X=0 ist der linke Fensterrand.
Y	Vertikale Position in Pixel – Y=0 ist der obere Fensterrand.
Breite	Breite des Elements in Pixel
Höhe	Höhe des Elements in Pixel

Spalten

Das Element TcEventTable verfügt über eine Tabelle mit den folgenden sieben Spalten:

- Index: Der Index stellt die Nummerierung der Meldungen in der Reihenfolge ihres Auftretens dar.
- Klasse: Die Klasse beschreibt die Art der Meldungen. Sie wird beim Erstellen einer Meldung definiert.
- Bestätigungszustand: Eine Meldung kann mit einer Bestätigungspflicht erstellt werden. Die Bestätigung kann im Programmcode aber auch im Element vorgenommen werden. Der Zustand, ob das Element (noch) bestätigt werden muss, wird in dieser Spalte angezeigt.
- Zurücksetzzustand: Eine Meldung kann, nachdem sie aktiviert worden ist, im Programmcode zurückgesetzt werden. In dieser Zeile wird angezeigt, ob die Meldung bereits zurückgesetzt worden ist.
- Datum, Zeit: In den beiden Spalten ‚Datum' und ‚Zeit' werden das Datum und die Zeit, an denen die Meldung aufgetreten ist, angezeigt.
- Meldung: Der eigentliche Text der Meldung wird in der letzten Spalte angezeigt.

Spalten	Einstellungsmöglichkeiten der sieben Spalten
Zeilenhöhe	Höhe der Zeilen in Pixel
Scrollbarbreite	Breite der Scrollbar in Pixel
Sortierreihenfolge	Hier kann die Reihenfolge, in der die Meldungen in dem Element angezeigt werden, eingestellt werden: • Neustes zuerst • Ältestes zuerst

Spalte

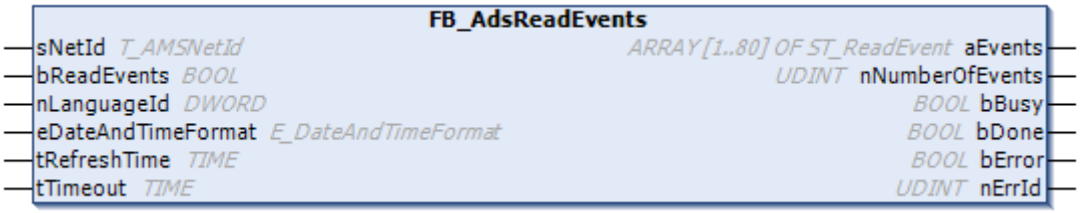
Spaltenbreite	Breite der Spalte in Pixel
Texteigenschaften	Hier können die Texteigenschaften der Spalte geändert werden: • Horizontale Ausrichtung • Vertikale Ausrichtung • Schriftart • Farbe Schriftart

Detaileigenschaften

Das Element TcEventTable hat zusätzlich zur Tabelle, in der die Meldungen angezeigt werden, ein Detailfeld. Wenn zur Laufzeit eine der Meldungen in der Tabelle selektiert wird, werden im Detailfeld weitere Informationen zu der Meldung anzeigt. Über die Quittierschaltfläche  ist es möglich, diese Meldung zu bestätigen.

Allgemeine Texteigenschaften	Hier können die Texteigenschaften für alle Zellen der Details außer der Meldungszelle eingestellt werden: • Horizontale Ausrichtung • Vertikale Ausrichtung • Schriftart • Farbe Schriftart
Meldungstexteigenschaften	Hier können die Texteigenschaften für die Meldungszelle eingestellt werden: • Horizontale Ausrichtung • Vertikale Ausrichtung • Schriftart • Farbe Schriftart

15.8.6.5.1 FB_AdsReadEvents



Der Funktionsbaustein erfragt die aktiven Meldungen des EventLoggers via ADS und stellt sie in Form eines Arrays `aEvents` zur Verfügung. Wenn die Meldungen im Visualisierungselement Event-Tabelle [▶ 627] angezeigt werden sollen, muss das Array `aEvents` in dessen Eigenschaft Meldungsdatenarray [▶ 628] eingetragen werden.

Meldungen mit einer Textlänge kleiner gleich 255 Zeichen können am Ausgang vollständig ausgegeben werden. Meldungen mit einer Textlänge größer 255 Zeichen und kleiner gleich 1023 Zeichen werden mit abgeschnittenem Text ausgegeben. Meldungen mit einer Textlänge größer 1023 Zeichen können nicht ausgegeben werden und der Funktionsbaustein liefert einen Fehler.

VAR_INPUT

```
VAR_INPUT
    sNetId             : T_AMSNetId;
    bReadEvents        : BOOL;
    nLanguageId        : DWORD;
    eDateAndTimeFormat : E_DateAndTimeFormat;
    tRefreshTime       : TIME
    tTimeout           : TIME
END_VAR
```

sNetId: AmsNetId des Geräts, von dem die Meldungen des EventLoggers abgefragt werden sollen. Falls die Meldungen lokal ausgelesen werden sollen, kann ein Leerstring angegeben werden.

bReadEvents: Mit dem Eingang kann die Freigabe zum Auslesen der Meldungen gegeben werden. Mit Rücknahme der Freigabe werden auch die Fehlerausgänge (bError und nErrId) zurückgesetzt.

nLanguageId: (Sprach-ID) Definiert, welche Übersetzung des Meldungstextes abgefragt werden soll.

eDateAndTimeFormat: Definiert, welches Format die Zeitstempel haben sollen. Zur Auswahl stehen:

- de_De – deutsche Schreibweise: dd.MM.yyyy hh:mm:ss (24 h)
- en_GB – britische Schreibweise: dd/MM/yyyy hh:mm:ss (12 h)
- en_US – amerikanische Schreibweise: MM/dd/yyyy hh:mm:ss (12 h)

tRefreshTime: Definiert die Zeitspanne, nach der die Abfrage der Meldungen wiederholt wird.

tTimeout: Definiert die Zeitspanne, nach der ein Zeitüberschreitungsfehler ausgelöst wird.

VAR_OUTPUT

```
VAR_OUTPUT
    aEvents         : ARRAY[1..80] OF ST_ReadEvent;
    nNumberOfEvents : UDINT;
    bBusy           : BOOL;
    bDone           : BOOL;
    bError          : BOOL
    nErrorId        : UDINT;
END_VAR
```

aEvents: Über dieses Array stellt der Funktionsbaustein die ausgelesenen Meldungen zur Verfügung. Maximal können in dem Array 80 Meldungen gespeichert werden. (siehe ST_ReadEvent)

nNumberOfEvents: Gibt an, wie viele Meldungen aktuell im Array `aEvents` gespeichert sind.

bBusy: Gibt an, ob der Baustein gerade arbeitet.

bDone: TRUE, wenn der Baustein gerade nicht arbeitet, aber mindestens einmal gearbeitet hat.

bError: Gibt an, ob ein Fehler aufgetreten ist.

nErrorId: Gibt die Fehlernummer an.

Voraussetzungen

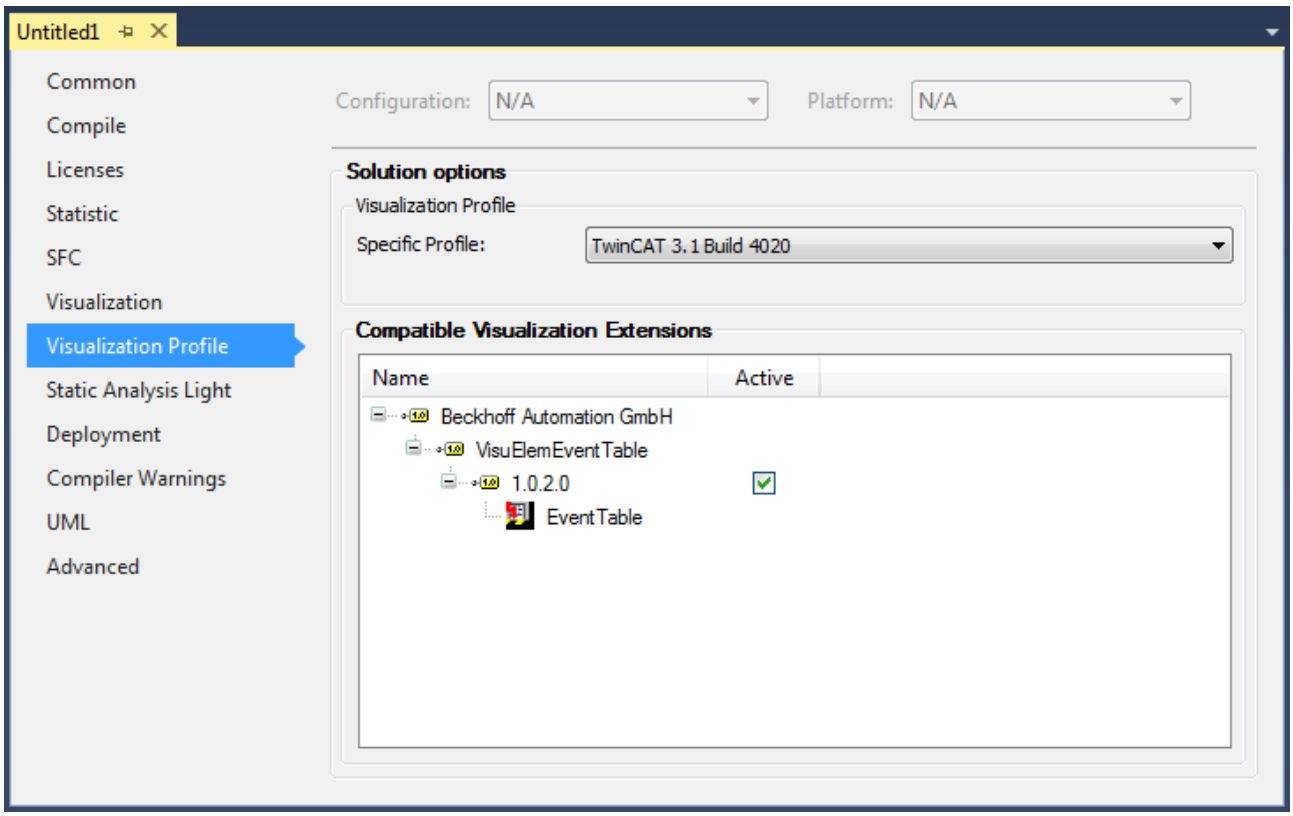
Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken (Kategoriegruppe)
TwinCAT v3.1.0	PC oder CX (x86, x64, ARM)	Tc2_Utilities (System)

15.8.6.5.2 Konfiguration der Event-Tabelle

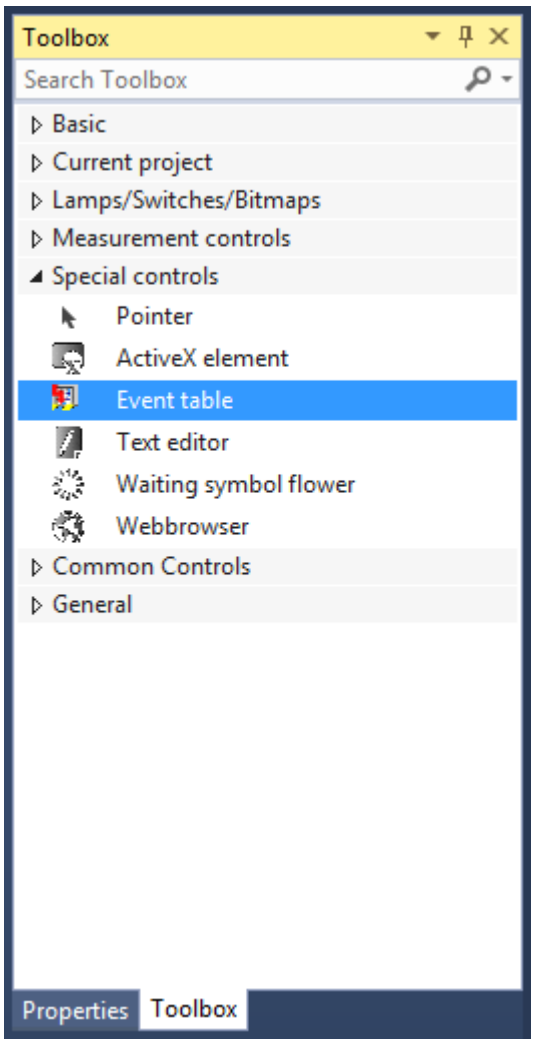
Das Visualisierungselement "Event-Tabelle" arbeitet in Kombination mit dem Funktionsbaustein "FB_AdsReadEvents [▶ 629]". Deshalb wird zunächst der Funktionsblock zum SPS Programm hinzugefügt. Er ist enthalten in der Bibliothek "Tc2_Utilities". In diesem Beispiel wird der Funktionsblock im Programm "MAIN" deklariert und aufgerufen. Da die Meldungen des lokalen TcEventLoggers ausgelesen werden sollen, kann ein Leerstring beim Eingang "sNetId" eingetragen werden.

```
PROGRAM MAIN
VAR
    fbAdsReadEvents : FB_AdsReadEvents;
    bReadEvents : BOOL;
END_VAR
fbAdsReadEvents (    sNetId := ,',
                bReadEvents := bReadEvents,
                nLanguageId := 1031,
                eDateAndTimeFormat := E_DateAndTimeFormat.de_DE,
                tRefreshTime := T#1S,
                tTimeout := T#5s);
```

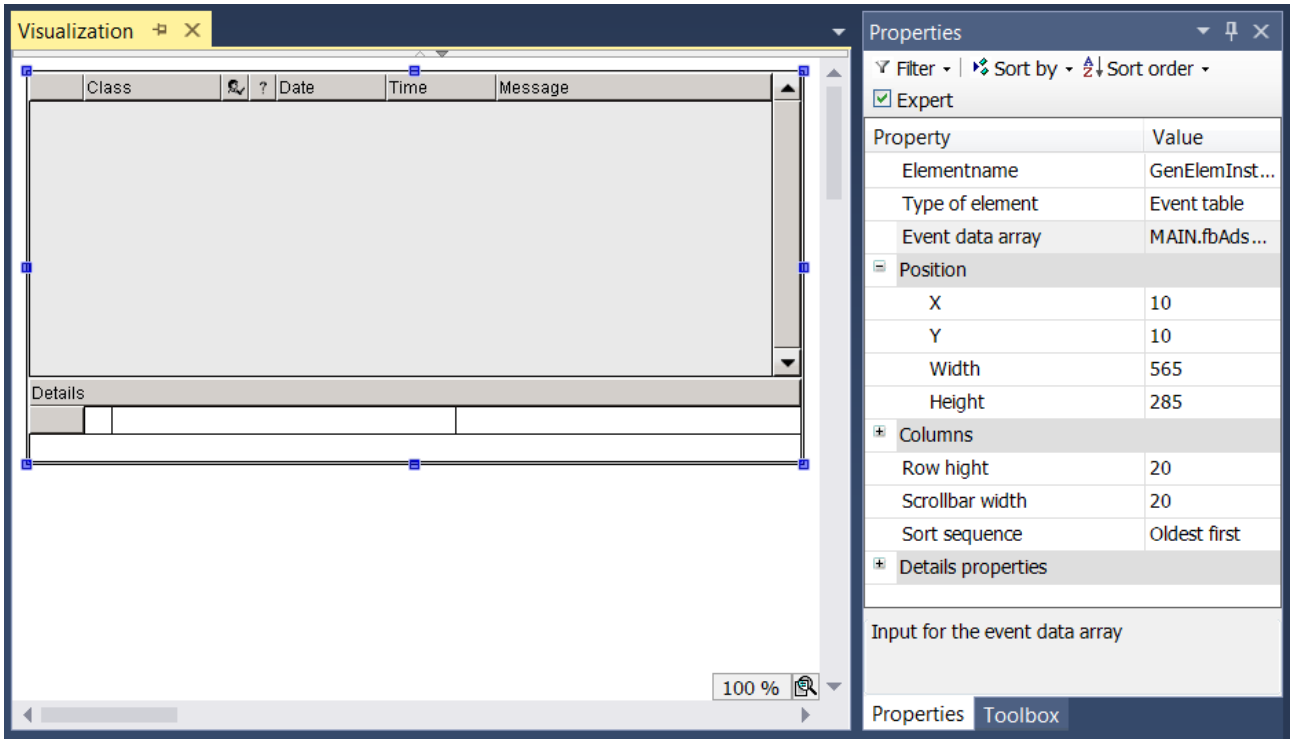
Um das Visualisierungselement "Event-Tabelle" auf einer Visualisierungsseite hinzufügen zu können, muss in den SPS-Projekteinstellungen [▶ 421] unter der Kategorie "Visualization Profile" die entsprechende Erweiterung für das Element ausgewählt werden.



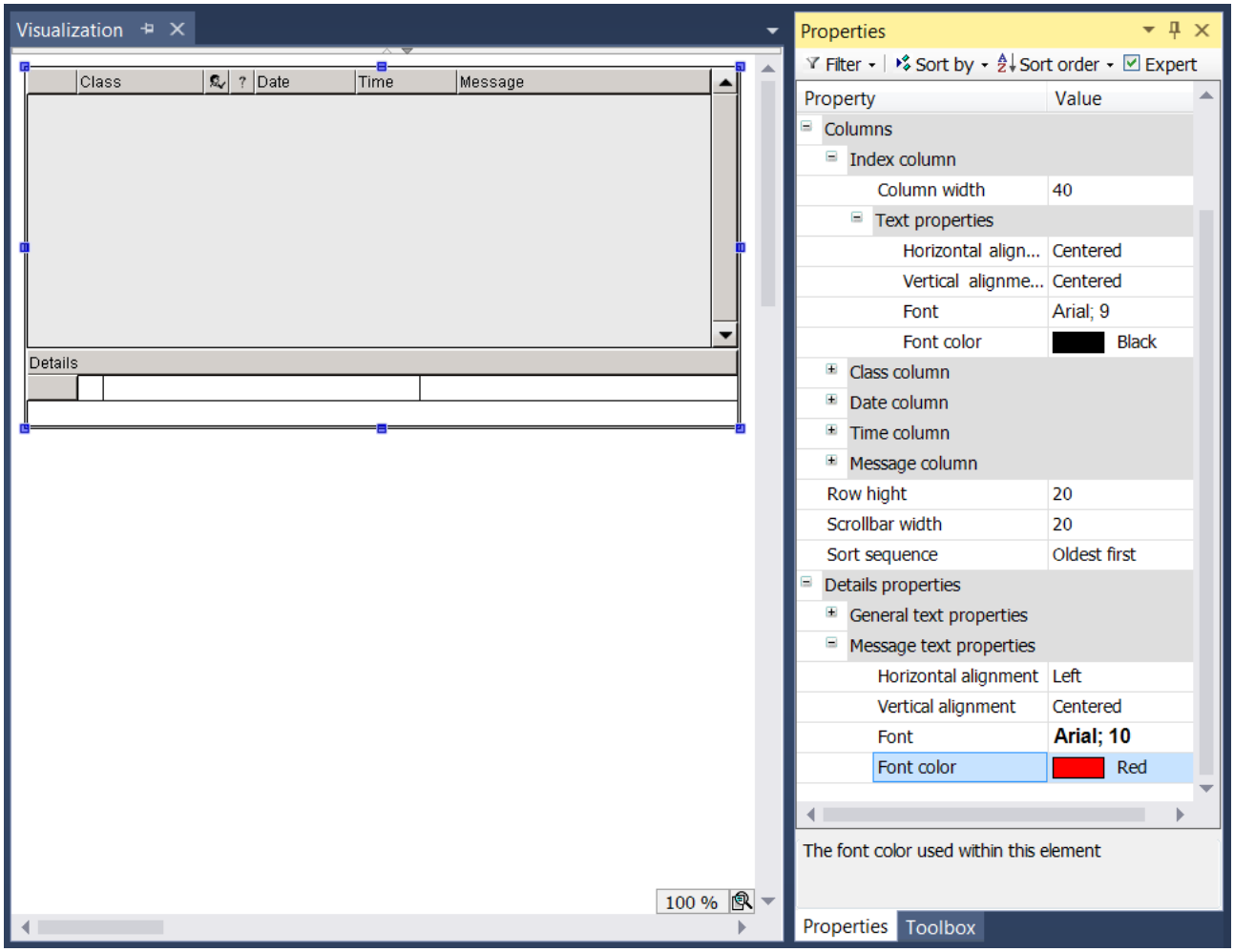
Nach der Aktivierung dieser Einstellung muss das TwinCAT-Projekt einmal neu gestartet werden. Dann ist das Element im Werkzeugkasten [▶ 402] in der Kategorie "Spezielle Steuerelemente [▶ 612]" verfügbar und kann auf der Visualisierungsseite verwendet werden.



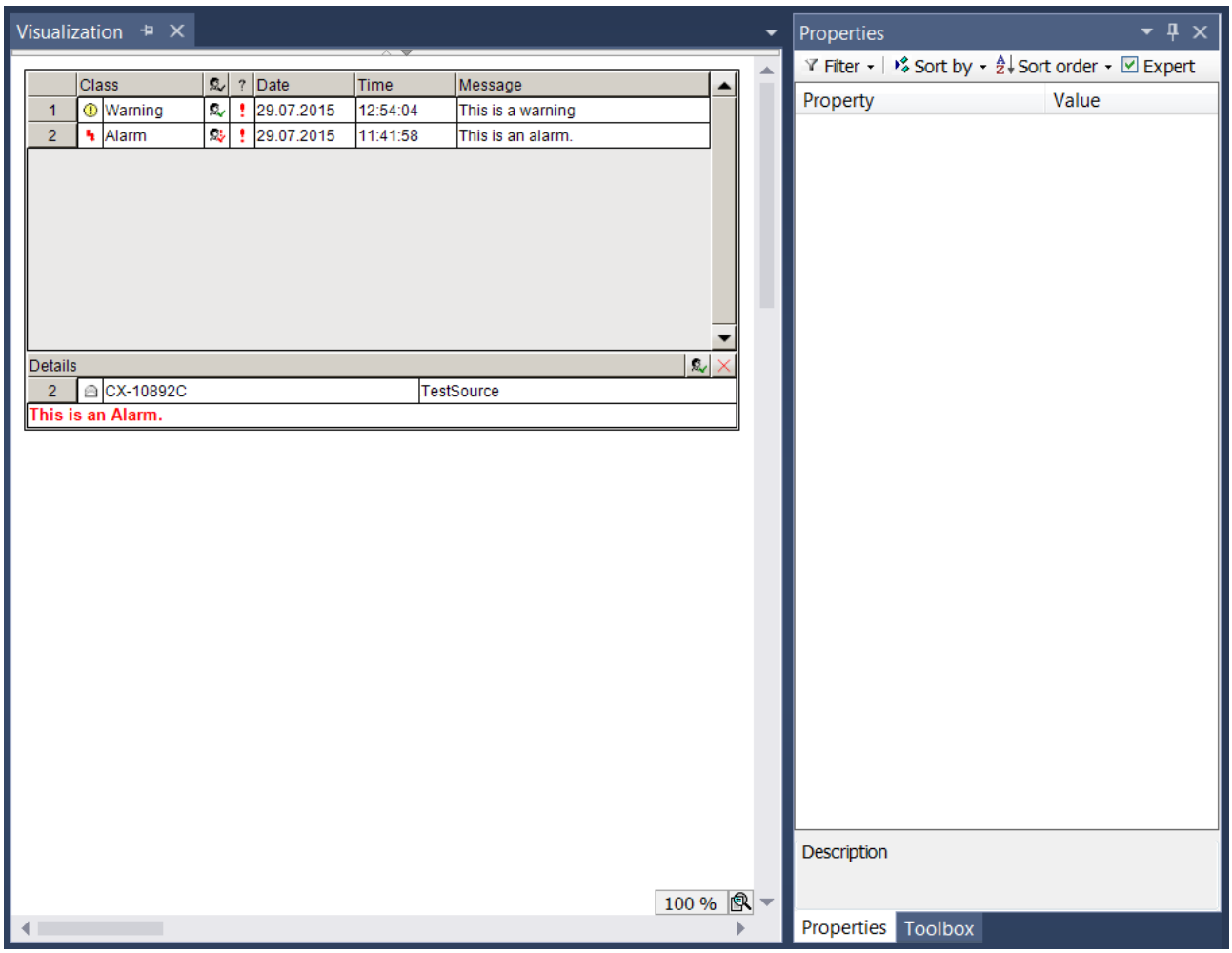
Nachdem das Element zur Visualisierungsseite hinzugefügt worden ist, wird in den Eigenschaften das Array "aEvents" der Instanz "fbAdsReadEvents", in dem die Meldungen gespeichert werden, eingetragen und die Größe auf 565x285 Pixel eingestellt.



Um den Text der Indexspalte zentriert auszurichten, wird in der Kategorie "Spalten" → "Indexspalte" die Texteigenschaften "Horizontale Ausrichtung" in „Zentral“ geändert. Zudem wird die Schriftgröße und -farbe des Meldungstextes in den "Detaileigenschaften" angepasst, um ihn von den übrigen Informationen abzuheben.



Zur Laufzeit sieht das Visualisierungselement wie folgt aus, wenn zwei Beispielmeldungen ausgelöst worden sind.



15.9 Visualisierungsvarianten

Es werden drei verschiedene Varianten der Visualisierung im SPS-Projekt unterschieden:

- Integrierte Visualisierung [▶ 634] – Visualisierung läuft in der Entwicklungsumgebung von TwinCAT auf dem Programmiersystem
- PLC HMI [▶ 635] – Visualisierung läuft ohne Entwicklungsumgebung auf dem Steuerungsrechner oder einem dritten Rechner
- PLC HMI Web [▶ 640] – Visualisierung läuft in einem Browser

Trotz der drei verschiedenen Varianten ist nur ein Engineering notwendig, wodurch das Look-and-feel in allen Visualisierungen identisch ist. PLC HMI und PLC HMI Web können ganz einfach durch Hinzufügen des TargetVisualization [▶ 638]- und WebVisualization [▶ 641]-Objekts freigeschaltet werden. Die Verfügbarkeiten der einzelnen Visualisierungselemente [▶ 424] und Funktionalitäten in den verschiedenen Varianten sind im Abschnitt "Verfügbarkeiten [▶ 643]" zu finden.

15.9.1 Integrierte Visualisierung

Zu Diagnosezwecken kann es gewünscht sein, eine Visualisierung nur innerhalb des Programmiersystems laufen zu lassen, ohne Visualisierungscode auf die Steuerung laden zu müssen. Diese integrierte Visualisierung wird automatisch verwendet, wenn kein "TargetVisualization [▶ 638]" oder "WebVisualization [▶ 641]" Client-Objekt unterhalb des Visualisierungsmanagers [▶ 405] hinzugefügt worden ist. Dann wird kein Visualisierungscode erzeugt und auf die Steuerung geladen. Dies bedeutet allerdings einige Einschränkungen, die im Folgenden aufgelistet sind.

Einschränkungen für Ausdrücke, Monitoring

Die Diagnose-Visualisierung unterstützt nur Ausdrücke, die vom Monitoring-Mechanismus des Programmiersystems gehandhabt werden können. Diese sind:

- normale Variablenzugriffe wie MAIN.fbTest.nCounter
- komplexe Zugriffe wie im Folgenden aufgelistet:
  - Zugriff auf ein Array von skalaren Datentypen, wobei eine Variable als Index verwendet wird (a[i])
  - Zugriff auf ein Array von komplexen Datentypen (Strukturen, Funktionsbausteinen, Arrays) wobei eine Variable als Index verwendet wird (a[i].x)
  - Zugriff auf ein mehrdimensionales Array von allen Arten von Datentypen, mit einer oder mehreren Variablen-Indizes (a[i, 1, j].x)
  - Zugriff auf ein Array mit konstantem Index (a[3])
  - Zugriffe wie die oben beschriebenen, in denen einfache Operatoren für die Berechnungen innerhalb der Indexklammer verwendet werden (a[i + 3])
  - verschachtelte Kombinationen der oben gelisteten komplexen Ausdrücke (a[i + 4 * j].aInner[j * 3].x)
- in Index-Berechnungen unterstützte Operatoren: +, -, *, /, MOD
- Pointer-Monitoring wie p^.x
- nicht unterstützt werden Methoden oder Funktionsaufrufe außer den folgenden:
  - alle Standard-Stringfunktionen
  - alle Typkonvertierungsfunktionen wie INT_TO_DWORD
  - alle Operatoren wie SEL, MIN, ...

Einschränkungen für Eingaben

Innerhalb der Eingabeaktion "ST-Code ausführen" wird nur eine Liste von Zuweisungen unterstützt.

Beispiel:

```
PLC_PRG.n := 20 * PLC_PRG.m;
// nicht erlaubt
IF PLC_PRG.n < MAX_COUNT THEN
PLC_PRG.n := PLC_PRG.n + 1;
END_IF
//statt dessen folgendes verwenden:
PLC_PRG.n := MIN(MAX_COUNT, PLC_PRG.n + 1);
```

Wenn eine Liste von Zuweisungen verwendet wird, wird der Wert auf der linken Seite erst im nächsten Zyklus zugewiesen. Eine unmittelbar folgende Verarbeitung in der nächsten Zeile ist nicht möglich.

Visualisierungsschnittstelle

Innerhalb der Schnittstellendefinition einer Visualisierung darf der Typ "Schnittstelle" ("INTERFACE") nicht verwendet werden.

15.9.2 PLC HMI

Die PLC HMI ist eine Erweiterung des Laufzeitsystems und ermöglicht es, die Visualisierung ohne eine Entwicklungsumgebung auf dem Steuerungsrechner oder einem dritten Rechner auszuführen. Der Visualisierungscode wird entsprechend der vorhandenen Visualisierungsobjekte erstellt und auf den Steuerungsrechner heruntergeladen. Der Verzicht auf die Entwicklungsumgebung ermöglicht eine erhebliche Einsparung an Speicherverbrauch. Das kann für kleine Rechner nützlich sein.

Nachfolgend werden folgende Themen beschrieben:

- Inbetriebnahme der PLC HMI [▶ 636]
- Remote-Betrieb eines PLC HMI Clients [▶ 638]
- Editor des Objekts TargetVisualization [▶ 638]

Inbetriebnahme der PLC HMI

Schritt 1: PLC HMI freischalten

Das Objekt „TargetVisualization“ ( ) schaltet die PLC HMI frei. Sie fügen es dem Objekt „Visualization Manager“ im SPS-Projektbaum über den Kontextmenübefehl **Add > TargetVisualization** hinzu (siehe auch Dokumentation PLC: Visualisierung erstellen > Visualisierungsobjekt [▶ 422]).

Mit dem TargetVisualization-Objekt wird automatisch eine Visualisierungstask „VISU_TASK“ in der Projektmappe und eine Referenz auf diese Task in dem SPS-Projekt erstellt. Mithilfe der Referenz wird der Visualisierungscode aufgerufen. Nach dem Hinzufügen des Objekts müssen Sie daher die Konfiguration neu aktivieren.

Löschen eines TargetVisualization-Objekts

Wenn Sie ein TargetVisualization-Objekt löschen und kein zusätzliches WebVisualization-Objekt hinzugefügt haben, müssen Sie im TwinCAT-Projektbaum unter **System > Tasks** die Task „VISU_TASK“ löschen. Diese Task wird in der integrierten Visualisierung nicht benötigt. (Siehe auch Editor des Objekts WebVisualization [▶ 641] und Integrierte Visualisierung [▶ 634])

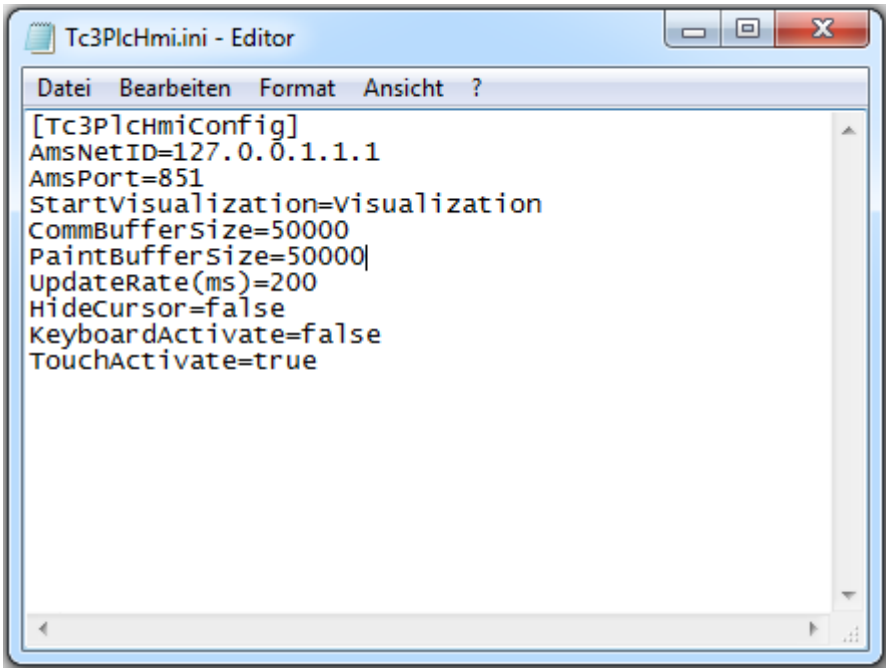
Schritt 2: PLC HMI Client konfigurieren

Führen Sie Schritt 2 nur aus, wenn Sie ein Build <4022.0 nutzen oder einen PLC HMI Client mit Remote-Verbindung zum Laufzeitgerät starten wollen. Ab dem Build 4022.0 wird die .ini-Datei automatisch im Ordner *C:\TwinCAT\3.1\Boot\Plc* generiert und aktualisiert. Ab dem 4026.0 wird die .ini-Datei automatisch im Ordner *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot\Plc* generiert und aktualisiert.

Um die Verbindung zwischen dem Client und dem Gerät herzustellen, auf dem der entsprechende Visualisierungscode ausgeführt wird, müssen Sie die Tc3PlcHmi.ini-Datei anpassen.

Die .ini-Datei ist für Builds <4022.0 im Ordner *C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi* verfügbar, für Builds >=4022.0 im Ordner *C:\TwinCAT\3.1\Boot\Plc* und für Builds >=4026.0 im Ordner *C:\Program Files (x86)\Beckhoff\TwinCAT\3.1\Components\Plc\Tc3PlcHmi*.

Beispiel für eine .ini-Datei:



AmsNetID	AmsNetID des Geräts, auf dem der Visualisierungscode ausgeführt wird. Voreinstellung: 127.0.0.1.1.1
AmsPort	AmsPort des PLC Projektes, zu dem die Visualisierung gehört. Voreinstellung: 851
StartVisualization	Name des Visualisierungsobjekts, das als Startseite geöffnet werden soll. Voreinstellung: Visualization
CommBufferSize	Speichergröße in Bytes, den die Visualisierung für diesen PLC HMI Client allokiert und für die Kommunikation verwendet. Voreinstellung: 50000
PaintBufferSize	Speichergröße in Bytes, den die Visualisierung für diesen PLC HMI Client allokiert und bei Zeichenaktionen verwendet. Voreinstellung: 50000
UpdateRate(ms)	Aktualisierungsrate in Millisekunden, mit der die Daten des Clients erneut abgefragt werden. Voreinstellung: 200
HideCursor	Einstellung, über die der Cursor ausgeblendet werden kann. Voreinstellung: false
KeyboardActivate	Einstellung, über die die Eingabe über eine Hardware-Tastatur aktiviert wird. Wenn diese Einstellung nicht aktiv ist, wird automatisch eine Software-Tastatur genutzt. Voreinstellung: false
TouchActivate	Einstellung, über die die Eingabe via Touch aktiviert wird. Voreinstellung: true

Schritt 3: PLC HMI als Startup-Anwendung einstellen

Führen Sie Schritt 3 nur aus, wenn Sie ein Build <4024.0 nutzen oder einen PLC HMI Client mit Remote-Verbindung zum Laufzeitgerät starten wollen. Ab dem Build 4024 wird der PLC HMI Client automatisch lokal auf dem Laufzeitgerät gestartet.

Wenn die PLC HMI automatisch beim Hochfahren des Rechners mit dem Boot-Projekt starten soll, muss sich im Ordner *StartUp* eine Verknüpfung zur Tc3PlcHmi.exe-Anwendung befinden.

Führen Sie dazu folgende Schritte aus:

1. Öffnen Sie das Verzeichnis *C:\TwinCAT\3.1\Target\StartUp*.
2. Fügen Sie über den Kontextmenübefehl **Neu** eine neue Verknüpfung hinzu.
3. Geben Sie als Speicherort *C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi\Tc3PlcHmi.exe* an.
4. Bestätigen Sie diesen und den folgenden Dialog.

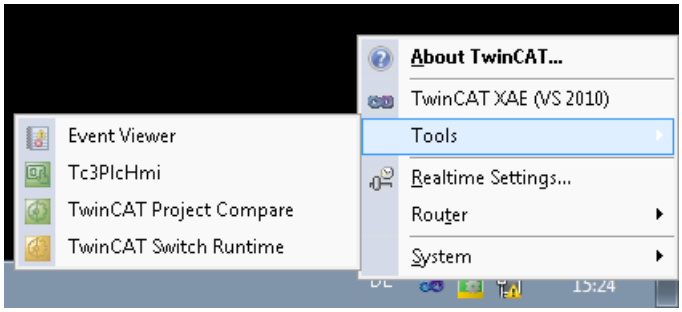
Für Beckhoff-CE-Geräte führen Sie die folgende Schritte aus:

1. Starten Sie den Beckhoff Startup Manager unter **Start > StartMan**.
2. Fügen Sie ein neues Item über die Schaltfläche **New** hinzu.
3. Geben Sie dem Item den Namen „Tc3PlcHmi“ und wählen Sie den Typ „ShellCommand“ aus.
4. Bestätigen Sie den Dialog.
5. Wählen Sie unter den **Startup Options** „Autostart“ und tragen Sie bei **Delay** eine Zeit ein, um den Client erst dann zu öffnen, wenn das SPS Projekt schon gestartet worden ist.
6. Wechseln Sie zur Registerkarte **Shell Command**.
7. Geben Sie in das Feld **Enter Shell command** „\Hard Disk\TwinCAT\3.1\Components\Plc\Tc3PlcHmi\X.exe“ ein. Das „X“ muss durch den Namen der Client Exe ersetzt werden, die unter dem genannten Pfad abgespeichert ist. Dieser kann sich zum Beispiel von ARM zu ATOM Geräte unterscheiden.
8. Bestätigen Sie den Dialog.

Schritt 4: PLC HMI Client starten

Führen Sie Schritt 4 nur aus, wenn Sie ein Build <4024.0 nutzen oder einen PLC HMI Client mit Remote-Verbindung zum Laufzeitgerät starten wollen. Ab dem Build 4024 wird der PLC HMI Client automatisch lokal auf dem Laufzeitgerät gestartet.

Sie starten einen PLC HMI Client mithilfe der Tc3PlcHmi.exe-Anwendung. Diese befindet sich im Verzeichnis *C:\TwinCAT\3.1\Components\Plc\Tc3PlcHmi*, kann aber an einem beliebigen Ort verknüpft werden. Wenn Sie im Verzeichnis *C:\TwinCAT\3.1\Target\StartMenuAdmin\Tools* eine Verknüpfung anlegen, können Sie die Anwendung über das TwinCAT Icon im Kontextmenü unter **Tools** zu starten.



Wenn der Entwicklungsrechner verbunden ist, kann die Visualisierung zusätzlich in der Entwicklungsumgebung angezeigt werden. Sie ist aber nicht äquivalent zur integrierten Visualisierung, sondern basiert auch auf einem PLC HMI Client.

Für Beckhoff-CE-Geräte müssen Sie vor dem Starten des Clients eine Einstellung im Visualisierungsmanager aktivieren, die eine automatische Umwandlung aller Bilddateien im svg-Format in das bmp-Format freischaltet. Der Schritt ist notwendig, da unter CE nur Bilddateien im bmp-Format im PLC HMI Client unterstützt werden. Es werden trotzdem beide Formate der Bilddatei auf das Zielsystem geladen, da ein PLC HMI Web Client weiterhin das svg-Format nutzt. Der PLC HMI Client für CE ist im Verzeichnis *\Hard Disk\TwinCAT\3.1\Components\Plc\Tc3PlcHmi* zu finden.

Siehe auch:

- Dokumentation PLC: Visualisierung erstellen > Visualisierungsmanager > Einstellungen [▶ 406]
- Dokumentation PLC: Visualisierung erstellen > Visualisierungsvarianten > Integrierte Visualisierung [▶ 634]
- Dokumentation TC3 PLC HMI Web [▶ 640]

Remote-Betrieb eines PLC HMI Clients

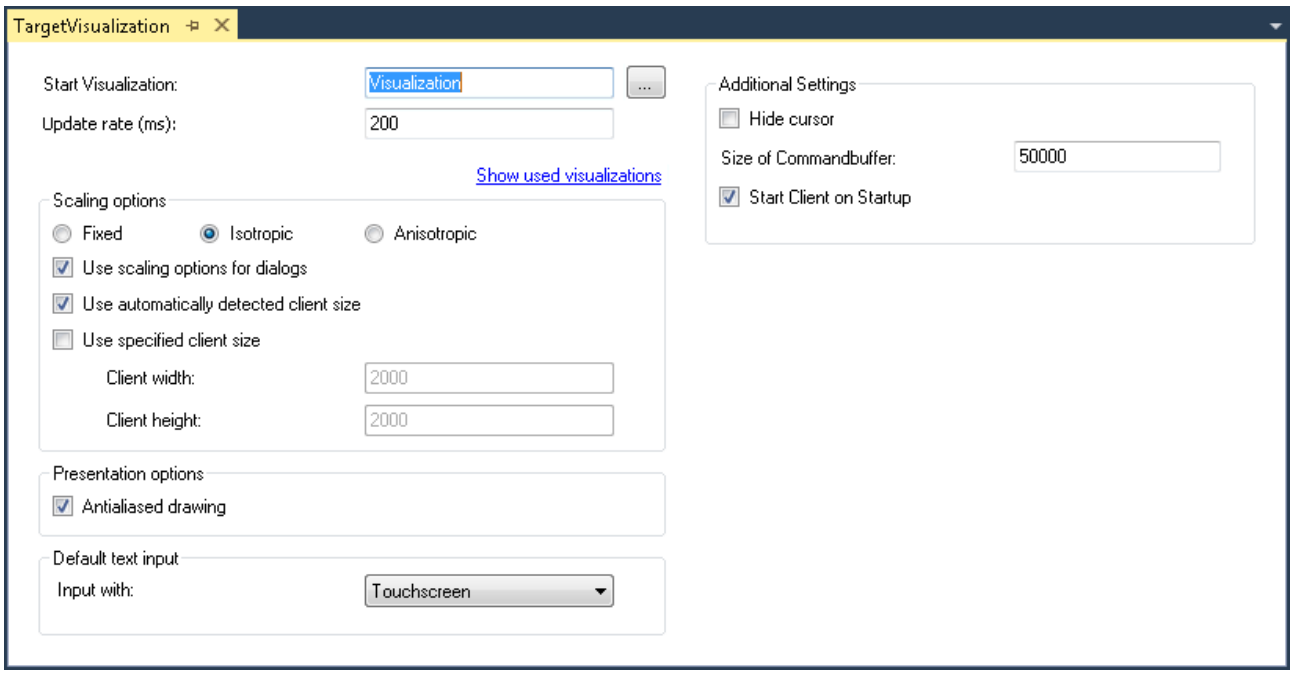
Ein PLC HMI Client kann auch Remote auf einem dritten Rechner, also weder dem Entwicklungs- noch dem Steuerungsrechner, ausgeführt werden. Dafür müssen folgende Anforderungen erfüllt sein:

- Ein TwinCAT 3 Build 4018.0 ADS oder neuer ist auf dem System installiert.
- Eine ADS-Kommunikation zum Steuerungsrechner, auf dem auch der Visualisierungscode ausgeführt wird, ist aufgebaut (**TwinCAT Icon > Router > Routes editieren > Add…**).
- Der Tc3PlcHmi-Ordner ist vom Entwicklungs- oder Steuerungsrechner auf das dritte System kopiert worden. Der Pfad für den Ordner muss händisch hinzugefügt werden.
- Die Tc3PlcHmi.ini-Datei ist auf dem System, auf dem der Client ausgeführt werden soll, entsprechend angepasst worden.

Editor des Objekts TargetVisualization

Das Objekt „TargetVisualization“ ( ), das Sie im SPS-Projektbaum unterhalb des Objekts „Visualization Manager“ hinzufügen können, schaltet die PLC HMI frei und enthält ihre Einstellungen. Um die Einstellungen in einem Editorfenster zu bearbeiten, klicken Sie doppelt auf das Objekt.

Die Einstellungen im Objekt „TargetVisualization“ werden ab dem Build 4022.0 automatisch in die .ini-Datei übernommen. Wenn Sie ein älteres Build nutzen oder einen PLC HMI Client mit Remote-Verbindung zum Laufzeitgerät starten wollen, müssen Sie Änderung in den Einstellungen per Hand in die .ini-Datei übernehmen.



Startvisualisierung	Name des Visualisierungsobjekts, das beim Starten des PLC HMI als erste Seite geöffnet werden soll. Standardmäßig ist hier bereits ein Visualisierungsobjekt eingetragen. Zur Auswahl eines anderen Visualisierungsobjekts kann die Eingabehilfe verwendet werden. Ist nur ein Visualisierungsobjekt im SPS-Projekt vorhanden, wird diese automatisch als Startvisualisierung verwendet.
Aktualisierungsrate (ms)	Aktualisierungsrate in Millisekunden, mit der die Daten innerhalb der PLC HMI aktualisiert werden.
Verwendete Visualisierungen anzeigen	Schaltfläche zum Öffnen des Standarddialogs des Visualisierungsmanagers: Hier können Sie die Visualisierungen, die für die PLC HMI verwendet werden sollen, auswählen. (Siehe auch Dokumentation PLC: Visualisierung erstellen >Visualisierungsmanager > Visualisierungen [▶ 412])

Skalierungsoptionen

Fest	Die Größe der Visualisierung wird unabhängig vom Bildschirm beibehalten.
Isotropisch	Die Größe der Visualisierung richtet sich nach der Größe des Bildschirms. Die Visualisierung behält ihre Proportionen.
Anisotropisch	Die Größe der Visualisierung richtet sich nach der Größe des Bildschirms. Die Visualisierung behält allerdings nicht ihre Proportionen.
Skalierungsoptionen für Dialoge verwenden	Die Dialoge, auch Keypad und Numpad, werden wie die Visualisierung mit dem gleichen Skalierungsfaktor skaliert. Dies ist vorteilhaft, wenn ein Dialog passend zur Visualisierung erstellt wurde.
Automatisch ermittelte Clientgröße verwenden	Die PLC HMI füllt den Client-Bildschirm aus.
Angegebene Clientgröße verwenden	Die PLC HMI füllt den durch folgende Maße bestimmten Bildschirmbereich. • Client Höhe: Höhe in Pixel • Client Breite: Breite in Pixel

Darstellungsoptionen

Zeichen mit Antialiasing	Aktivieren Sie diese Option, wenn beim Zeichnen der Visualisierungen im Visualisierungseditor-Fenster des Programmiersystems Kantenglättung verwendet werden soll. (Offline oder Online)

Standardtexteingabe

Diese Einstellung ist nur dann wirksam, wenn Sie in der Eingabekonfiguration des Visualisierungselements den Eingabetyp „Standard“ auswählen. Dann werden die im Visualisierungsmanager definierten Standard-Texteingaben verwendet.

Touchscreen	Wählen Sie diese Option, wenn das Zielgerät standardmäßig mit einem Touchscreen bedient wird.
Tastatur	Wählen Sie diese Option, wenn das Zielgerät standardmäßig mit einer Tastatur bedient wird.

Weitere Einstellungen

Mauszeiger verstecken	Einstellung, über die der Cursor ausgeblendet werden kann.
Größe des Commandbuffers	Speichergröße in Bytes, den die Visualisierung für diesen PLC HMI Client allokiert und für die Kommunikation verwendet.
Start Client on Startup	Der PLC HMI Client wird automatisch lokal auf dem Runtime-System gestartet.

15.9.3 PLC HMI Web

Die PLC HMI Web erlaubt die Darstellung der Visualisierung in einem beliebigen Webbrowser. Sie ist als Java-Skript realisiert, welches die Darstellungsinformation vom Webserver abfragt. Hierbei werden nur Änderungen der Darstellung zyklisch übertragen. Bei einem Download eines Visualisierungsprojekts werden alle für die PLC HMI Web benötigten Dateien bis <TC3.1.4026.0 in das Verzeichnis *C:\TwinCAT\3.1\Boot\Plc\Port_851\Visu* und ab >=TC3.1.4026.0 in das Verzeichnis *C:\ProgramData\Beckhoff\TwinCAT\3.1\Boot Plc\Port_851\Visu* übertragen. Dazu gehören das Java-Skript, die Basis-HTML-Seite (HTM-Datei) der Visualisierung, sowie alle in der Visualisierung benötigten Abbildungen.

Die PLC HMI Web kann momentan nur für SPS-Projekte, die über den Port 851 erreichbar sind, konfiguriert werden.

Nachfolgend werden folgende Themen beschrieben:

- Voraussetzungen [▶ 640]
- Inbetriebnahme PLC HMI Web [▶ 640]
- Editor des Objekts WebVisualization [▶ 641]

Voraussetzungen

- Server-seitig müssen die entsprechenden Konfigurationen des Web Servers vorgenommen sein.
- Client-seitig muss mindestens ein Microsoft Internet Explorer 10 oder die neuste Version von Mozilla Firefox, Google Chrome oder Safari vorhanden sein.

Datensicherheitsverletzungen

Um das Risiko von Datensicherheitsverletzungen zu minimieren, werden die folgenden organisatorischen und technischen Maßnahmen für das System, auf dem Ihre Applikation läuft, empfohlen:

- Vermeiden Sie soweit wie möglich, die SPS und Steuerungsnetzwerke offenen Netzwerken und dem Internet auszusetzen.
- Verwenden Sie zum Schutz zusätzliche Sicherungsschichten wie einen VPN für Remote-Zugriffe und installieren Sie Firewall-Mechanismen.
- Beschränken Sie den Zugriff auf autorisierte Personen, ändern Sie eventuell vorhandene Standard-Passwörter bei der ersten Inbetriebnahme und auch weiterhin regelmäßig.

Inbetriebnahme der PLC HMI Web

Schritt 1: Microsoft Internet Information Services (IIS) konfigurieren

Die PLC HMI Web nutzt den IIS von Microsoft als Web Server. Dafür muss der IIS entsprechend konfiguriert werden. Die Konfiguration übernimmt die Installation TF1810 | TC3 PLC HMI Web, die auf der Beckhoff Homepage zum Download zur Verfügung steht.

Schritt 2: PLC HMI Web freischalten

Das Objekt „WebVisualization“ () schaltet die PLC HMI Web frei. Sie fügen es dem Objekt „Visualization Manager“ im SPS-Projektbaum über den Kontextmenübefehl **Add > WebVisualization** hinzu (siehe auch Dokumentation PLC: Visualisierung erstellen > Visualisierungsobjekt).

Mit dem WebVisualization-Objekt wird automatisch eine Visualisierungstask „VISU_TASK“ in der Projektmappe und eine Referenz auf diese Task in dem Projekt erstellt. Mithilfe der Referenz wird der Visualisierungscode aufgerufen. Nach dem Hinzufügen des Objekts müssen Sie daher die Konfiguration neu aktivieren.

Löschen eines WebVisualization-Objekts

Wenn Sie ein WebVisualization-Objekt löschen und kein zusätzliches TargetVisualization-Objekt hinzugefügt haben, müssen Sie im TwinCAT-Projektbaum unter **System > Tasks** die Task „VISU_TASK“ löschen. Diese Task wird in der integrierten Visualisierung nicht benötigt. (Siehe auch: TF1800 :Editor des Objekts TargetVisualization und PLC: Integrierte Visualisierung)

Schritt 3: PLC HMI Web aufrufen

Um die Startseite der Visualisierung aufzurufen, tragen Sie die folgende Adresse in den Webbrowser ein: https://Gerätename/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm

Beispiel: https://localhost/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm

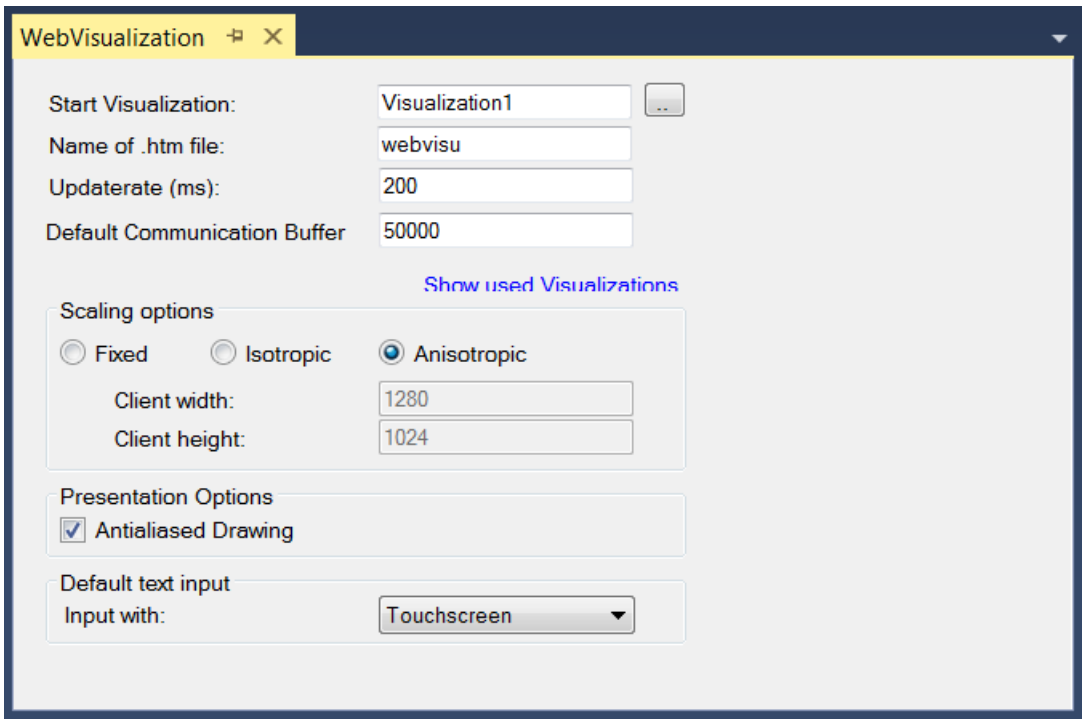
„webvisu“ ist die in den PLC-HMI-Web-Einstellungen definierte HTML-Startseite der Visualisierung. Über sie wird nach dem Aufruf im Browser zunächst die Startvisualisierung dargestellt, die ebenfalls im Manager definiert ist. Danach kann die Visualisierung im Browser bedient werden.

Optional können Sie die PLC HMI Web beim Aufruf mit einem Namen versehen, um sie später in der Applikation gezielt ansprechen zu können. Fügen Sie dazu hinter der URL den Parameter ClientName=<Name> hinzu.

Beispiel: https://localhost/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm?Clientname=V_ClientXY

Editor des Objekts WebVisualization

Das Objekt „WebVisualization“ (), das Sie im SPS-Projektbaum unterhalb des Objekts „Visualization Manager“ hinzufügen können, schaltet die PLC HMI Web frei und enthält die Einstellungen für die Web-Visualisierung. Um die Einstellungen in einem Editorfenster zu bearbeiten, klicken Sie doppelt auf das Objekt.



Startvisualisierung	Name der Visualisierung, die automatisch angezeigt werden soll, wenn die PLC HMI Web gestartet wird. Standardmäßig ist hier „Visualisierung“ eingetragen. Zur Auswahl einer anderen Visualisierung kann die Eingabehilfe verwendet werden.
Name der .htm-Datei	Name der Basis-HTML-Seite der Visualisierung, die dann auch als Adresse im Webbrowser eingegeben werden muss. Beispiel: https://localhost/Tc3PlcHmiWeb/Port_851/Visu/webvisu.htm
Aktualisierungsrate (ms)	Aktualisierungsrate in Millisekunden, mit der die Daten innerhalb des Webbrowsers aktualisiert werden.
Standardgröße	Kommunikationspuffergröße in Bytes. Gibt den maximal verfügbaren Speicher für die Übertragung der Daten zwischen Web-Client und Webbrowser an.
Verwendete Visualisierung anzeigen	Schaltfläche zum Öffnen des Standarddialogs des Visualisierungsmanagers: Hier können Sie die Visualisierungen, die für die PLC HMI Web verwendet werden sollen, auswählen. (Siehe auch Dokumentation PLC: Visualisierung erstellen > Visualisierungsmanager > Visualisierungen [▶ 412])

Skalierungsoptionen

Fest	Die Größe der Visualisierung wird unabhängig von der Größe des Browser-Fensters beibehalten.
Isotropisch	Die Größe der Visualisierung richtet sich nach der Größe des Browser-Fensters. Die Visualisierung behält allerdings ihre Proportionen.
Anisotropisch	Die Größe der Visualisierung richtet sich nach der Größe des Browser-Fensters. Die Visualisierung behält nicht ihre Proportionen.
Client-Größe	Die Anzeigegröße der PLC HMI Web wird durch die folgenden Einstellungen definiert: • Client Höhe: Höhe in Pixel • Client Breite: Breite in Pixel

Darstellungsoptionen

Zeichen mit Antialiasing	Aktivieren Sie diese Option, wenn beim Zeichnen der Visualisierungen im Visualisierungseditor-Fenster des Programmiersystems Kantenglättung verwendet werden soll. (Offline oder Online)

Standardtexteingabe

Diese Einstellung ist nur dann wirksam, wenn Sie in der Eingabekonfiguration des Visualisierungselements den Eingabetyp „Standard“ auswählen. Dann werden die im Visualisierungsmanager definierten Standard-Texteingaben verwendet.

Touchscreen	Wählen Sie diese Option, wenn die Web-Clients standardmäßig mit einem Touchscreen bedient werden.
Keyboard	Wählen Sie diese Option, wenn die Web-Clients standardmäßig mit einer Tastatur bedient werden.

15.9.4 Verfügbarkeiten

Im Folgenden sind die Verfügbarkeiten der einzelnen Visualisierungselemente und Funktionalitäten für die verschiedenen Visualisierungsarten dargestellt.

Visualisierungselemente

	Integrierte Visualisierung	PLC HMI	PLC HMI CE	PLC HMI Web
Allgemeine Steuerelemente [▶ 441]	✓	✓	✓	✓
Basis [▶ 499]	✓	✓	✓	✓
Lampen/ Schalter/ Bilder [▶ 556]	✓	✓	✓	✓
Messgeräte [▶ 567]	✓	✓	✓	✓
Spezielle Steuerelemente [▶ 612]				
• Text-Editor [▶ 614]	✗	✓	✓	✓
• ActiveX-Element [▶ 619]	✓	✗	✗	✗
• Webbrowser [▶ 621]	✓	✗	✗	✗
• TcEventTable [▶ 627]	✓	✓	✓	✓

Funktionalitäten

	Integrierte Visualisierung	PLC HMI	PLC HMI CE	PLC HMI Web
Benutzerverwaltung [▶ 412]	✗	✓	✓	✓
CurrentVisu [▶ 407], CurrentLanguage [▶ 648]	✗	✓	✓	✓
Sprachumschaltung [▶ 648]	✓	✓	✓	✓
Innere Rotation [▶ 455]	✓	✓	✗	✓
Farbverlauf [▶ 500]	✓	✓	✗	✓
Transparenz [▶ 500]	✓	✓	✗	✓
Textformat [▶ 442]	✓	✓	✗	✓

Unterstützte Bildformate

	Integrierte Visualisierung	PLC HMI	PLC HMI CE	PLC HMI Web
SVG	✓	✓	✗	✓
BMP	✓	✓	✓	✓
JPG	✓	✓	✓	✓
PNG	✓	✓	✓	✓

Die Visualisierungsstile [▶ 407] mit einer Version, die größer als 3.0.0.0 ist, verwenden intern Bilddateien im SVG-Format. Um diese Stile mit der PLC HMI CE verwenden zu können, kann im Visualisierungsmanager die Funktion Convert Images to [▶ 408] aktiviert werden. Die Funktion konvertiert die Bilddateien wahlweise in BMP oder PNG. Mithilfe der Einstellung SVG und konvertierte Bilder übertragen [▶ 408] werden sowohl die konvertierten als auch die originalen Bilddateien auf das Zielsystem übertragen. Ein PLC HMI Web Client verwendet dann automatisch die SVG- und ein PLC HMI CE Client die BMP- oder PNG-Variante.

15.10 Anwendungstipps

15.10.1 Handling von Visualisierungsseiten

Das TwinCAT-PLC-Control-Konzept von Visualisierungsreferenzen und Platzhaltern wird in TwinCAT 3 durch ein ähnliches ersetzt.

Referenz zu einer anderen Visualisierung

Es ist möglich, eine Visualisierungsseite [▶ 422] in eine andere einzufügen und damit zu referenzieren. Für diesen Zweck ist ein Frame [▶ 544]-Element zu verwenden. Auf diese Weise kann eine Visualisierungsseite aus verschiedenen anderen Seiten zusammengesetzt werden. Ein Frame-Element kann eine oder mehrere Referenzen zu Visualisierungsseiten einschließen. Diese Visualisierungsseiten werden im Dialog Frame-Auswahl [▶ 555] definiert.

Schnittstellen für Platzhalter

Jede Visualisierungsseite [▶ 422] kann eine Schnittstelle mithilfe des Schnittstellen-Editors [▶ 395] bereitstellen, in der Eingabevariablen vergleichbar mit einem Funktionsbaustein definiert werden können. Diese Eingabeparameter funktionieren als Platzhalter. In einer Instanz (Referenz) der Visualisierungsseite müssen sie durch Werte oder Ausdrücke für die spezielle Verwendung im lokalen Objekt ersetzt werden.

Das Ersetzen muss in den Eigenschaften [▶ 451] des Frame-Elements, der die Visualisierungsinstanz einbindet, vorgenommen werden. Beachten Sie dabei, dass den Eingabevariablen einer Visualisierungsinstanz gültige Variablen zugewiesen werden müssen. Falls Variablen im Schnittstellen-Editor geändert werden, wird für die Platzhalter jeder Instanz der Dialog "Aktualisierung der Frameparameter [▶ 555]" geöffnet. Hier können Platzhalter hinzugefügt oder verändert werden.

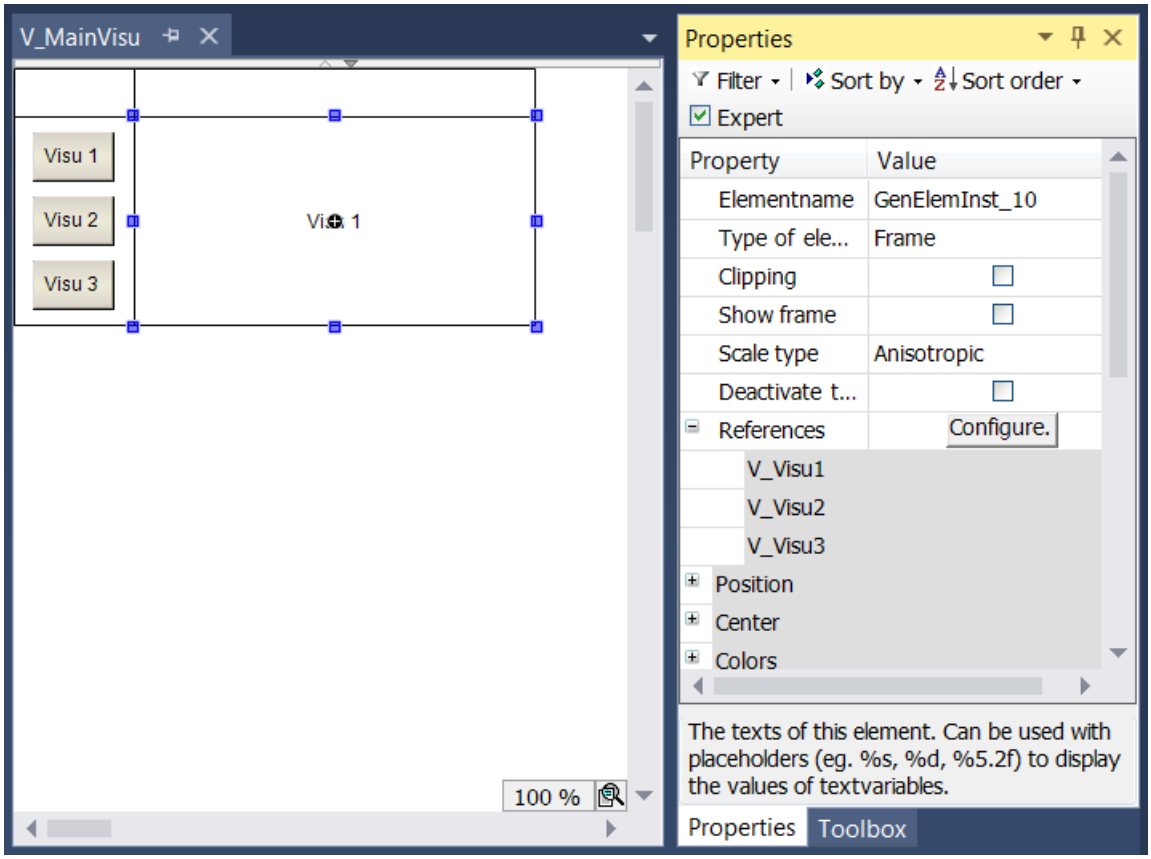
Umschalten zwischen Visualisierungsseiten innerhalb eines Frames

Wenn ein Frame-Element mehrere Visualisierungsreferenzen enthält, können Benutzereingaben auf ein anderes Visualisierungselement so konfiguriert werden, dass sie einen Wechsel der Anzeige dieser Referenzen im Frame bewirken. Die Eingabekonfiguration [▶ 458] bietet zu diesem Zweck die Aktion "Framevisualisierung umschalten [▶ 433]". Auf diese Weise kann auf einer Basisvisualisierungsseite zwischen mehreren anderen Visualisierungsseiten umgeschaltet werden.

Beispiel

Eine Visualisierungsseite "V_MainView" verfügt über ein Auswahlmenü, das aus drei Buttons und einem Frame-Element besteht. Jedem Button ist eine Visualisierungsseite zugeordnet, die beim Betätigen im Onlinebetrieb im Frame-Element angezeigt werden soll.

1. Anlegen eines Auswahlmenüs mit drei Buttons
2. Einfügen eines Frame-Elements
3. Diesem Frame werden über den Frame-Auswahl-Dialog drei Visualisierungen zugewiesen, zwischen denen umgeschaltet werden soll.
4. Für jeden Button wird über die Eingabekonfiguration [▶ 458] eine "OnMouseClick"-Aktion vom Typ "Framevisualisierung umschalten [▶ 433]" für die entsprechende Visualisierungsseite hinzugefügt.



Umschalten zwischen Visualisierungsseiten

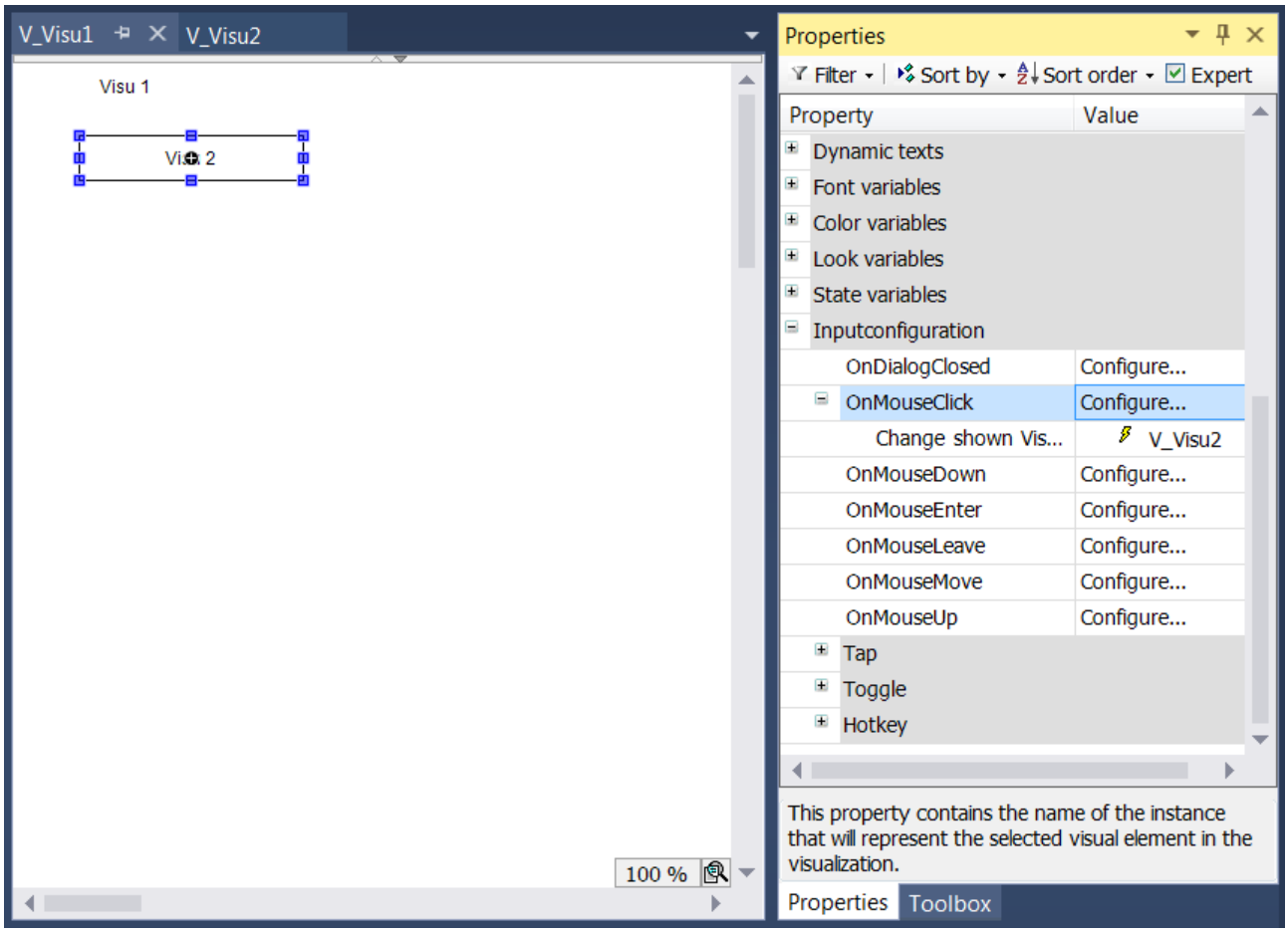
Zusätzlich zu der Umschaltmöglichkeit innerhalb eines Frame-Elements kann die gesamte aktuell sichtbare Visualisierungsseite geändert werden. Dazu bietet die Eingabekonfiguration [▶ 458] eine weitere Aktion mit dem Namen "Visualisierungswechsel [▶ 430]".

Beispiel

Es werden zwei Visualisierungsseiten mit Namen "V_Visu1" und "V_Visu2" erstellt. Auf jeder Seite ist eine Schaltfläche vorhanden mit der auf die jeweils andere Visualisierungsseite gewechselt werden kann.

1. Legen Sie zwei Visualisierungsobjekte mit den Namen "V_Visu1" und "V_Visu2" in einem SPS-Projekt an.
2. Fügen Sie auf beiden Visualisierungsseiten eine Beschriftung hinzu, um sie voneinander unterscheiden zu können.
   - "V_Visu1": Tragen Sie in den Eigenschaften der Beschriftung den Text "Visu 1" ein.
   - "V_Visu2": Tragen Sie in den Eigenschaften der Beschriftung den Text "Visu 2" ein.
3. Fügen Sie auf beiden Seiten ein Rechteckelement hinzu.
   - "V_Visu1": Tragen Sie in den Eigenschaften des Rechtecks den Text "Visu 2" ein.
   - "V_Visu2": Tragen Sie in den Eigenschaften des Rechtecks den Text "Visu 1" ein.

4. Konfigurieren Sie für beide Rechtecke ein "OnMouseClick [▶ 458]-Event mit der Aktion "Visualisierungswechsel [▶ 430]".
   - "V_Visu1": Weisen Sie der Aktion die Visualisierung "V_Visu2" zu.
   - "V_Visu2": Weisen Sie der Aktion die Visualisierung "V_Visu1" zu.



CurrentVisu Variable

Eine Seitenumschaltung ist zusätzlich über die CurrentVisu [▶ 407] Variable möglich. Nach der Zuweisung einer Visualisierungsseite werden automatisch alle aktiven Clients auf diese Seite aktualisiert.

Zuweisung einer Variablen:

```
VisuElems.CurrentVisu := sVisuName;
```

Zuweisung eines Texts:

```
VisuElems.CurrentVisu := ‚Visualization‘;
```

15.10.2 Text und Sprache

Texte und Sprachen werden in der PLC HMI grundsätzlich mithilfe von Textlisten [▶ 145] verwaltet. Dabei stehen ANSI und Unicode [▶ 407] als Zeichenkodierungsvarianten zur Verfügung. In der Visualisierung können Elemente [▶ 424] auf zwei verschiedene Arten beschriftet werden:

- Statischer Text
- Dynamischer Text

Statischer Text

Ein Visualisierungselement kann mit einem statischen Text versehen werden, indem dieser in den Elementeigenschaften in der Kategorie ‚Texte‘ eingetragen wird. Er kann auch direkt im Editorfenster eingetragen werden, wenn das Element selektiert ist und die Leertaste gedrückt wird. Dieser Text kann zur Laufzeit nicht geändert werden. Nur Änderungen der Sprache werden für den Text übernommen.

Jeder statische Text wird automatisch in der Textliste "GlobalTextList [▶ 148]" gespeichert. In dieser Textliste werden auch die Übersetzungen in andere Sprachen verwaltet.

GlobalTextList

ID	Default	de	en
0	%s		
1	Activate	Aktivieren	Activate
2	Deactivate	Deaktivieren	Deactivate

Dynamischer Text

Dynamischer Text wird in den Elementeigenschaften in der Kategorie ‚Dynamische Texte' eingetragen. Dazu muss zunächst der Name der Textliste aus den im SPS-Projekt vorhandenen ausgewählt werden. Hier kann die "GlobalTextList [▶ 148]" nicht verwendet werden, sondern ausschließlich selbst erstellte Textlisten [▶ 145]. Zusätzlich muss eine Variable für die ID des Texteintrages angegeben werden, über die zur Laufzeit der Text geändert werden kann.

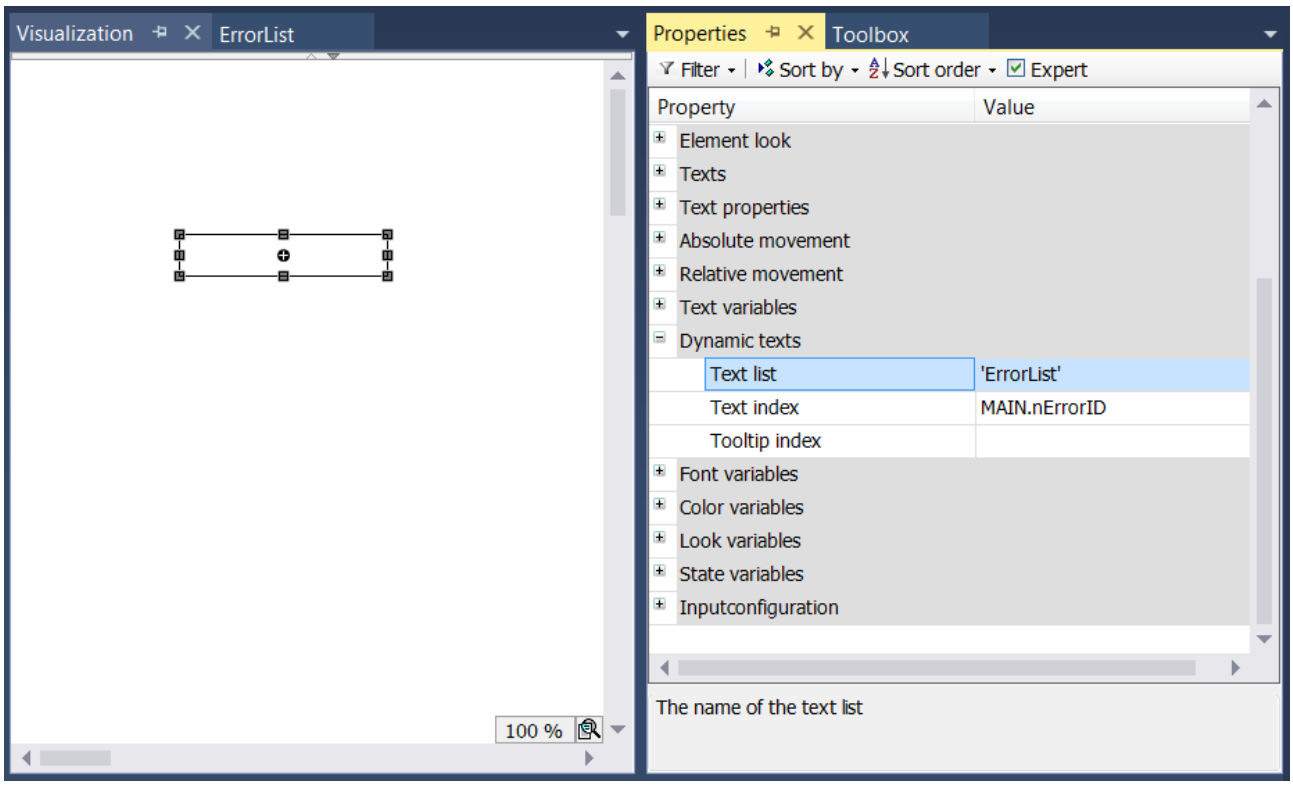
Beispiel

Beispielhaft wird eine Textliste mit dem Namen "ErrorList" angelegt. In dieser Liste wird eine Reihe von Fehlermeldungen mit ihren Übersetzungen eingetragen.

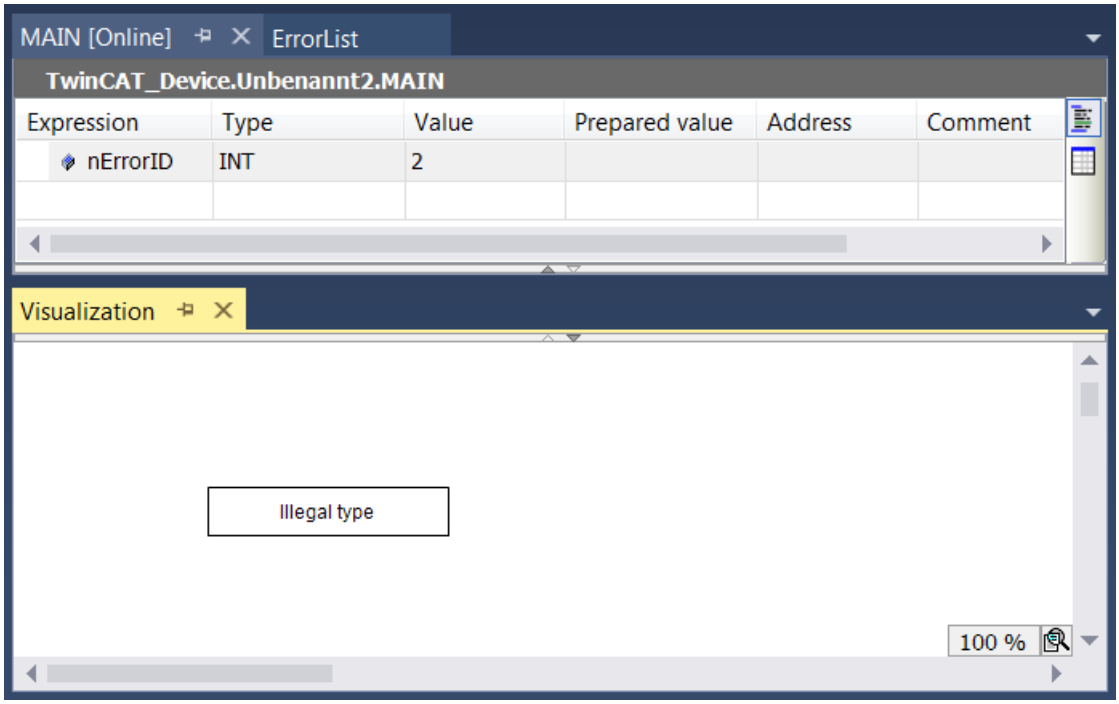
ErrorList

ID	Default	de	en
0	Wrong argument	Falsches Argument	Wrong argument
1	Bad format	Ungültiges Format	Bad format
2	Illegal type	Ungültiger Typ	Illegal type
3	Bad result	Ungültiges Ergebnis	Bad result
4	Wrong data type	Ungültiger Datentyp	Wrong data type

In einer Visualisierung mit dem Namen "Visualization" wird ein Rechteckelement [▶ 499] hinzugefügt. In den Eigenschaften dieses Elements werden unter "Dynamische Texte [▶ 456]" die Textliste "ErrorList" und die Variable "nErrorID", die in der "MAIN" deklariert ist, ausgewählt.



Zur Laufzeit kann nun eine der in der Textliste "ErrorList" gespeicherten Fehlermeldungen in dem Rechteck angezeigt werden, indem die Variable "nErrorID" auf einen Wert zwischen größer gleich 0 und kleiner gleich 4 gesetzt wird. Im Beispiel hat "nErrorID" den Wert 2.



Sprachumschaltung

Wenn die aktuell verwendete Textliste [▶ 145] Übersetzungen der Texte in mehrere Sprachen definiert, kann die beim Start der Visualisierung zu verwendende Sprache [▶ 407] festgelegt werden. Zusätzlich kann zur Laufzeit die Sprache mithilfe von Schaltflächen auf der Visualisierung geändert werden. Dafür wird die Aktion "Sprachumschaltung [▶ 430]" in den Elementeinstellungen unter der Kategorie ‚Eingabekonfiguration' genutzt.

Für die Abfrage der aktuell verwendeten Sprache kann die Variable "CURRENTLANGUAGE" der Bibliothek "VisuElems [▶ 420]" verwendet werden. Über diese Variable kann auch im Programmcode die verwendete Sprache geändert werden:

```
fbTrigger(CLK := bPressed);
IF fbTrigger.Q THEN
    IF VisuElems.CURRENTLANGUAGE = 'de' THEN
        VisuElems.CURRENTLANGUAGE := 'en';
    ELSE
        VisuElems.CURRENTLANGUAGE := 'de';
    END_IF
END_IF
```

Falls kein Eintrag in der Textliste vorhanden ist, der mit der gerade eingestellten Sprache übereinstimmt, wird der Eintrag unter Standard verwendet.

Die Definition einer Standardsprache beim Start der Visualisierung und die Verwendung der Variablen "VisuElems.CURRENTLANGUAGE" sind nur in Verbindung mit der PLC HMI [▶ 635] und/ oder der PLC HMI Web [▶ 640] möglich. Bei der integrierten Visualisierung [▶ 634] wird beim Start automatisch die Textversion ‚Standard' verwendet. Eine Sprachumschaltung zur Laufzeit über Schalflächen auf der Visualisierung ist auch in der integrierten Visualisierung möglich.

Formatierung des Textes

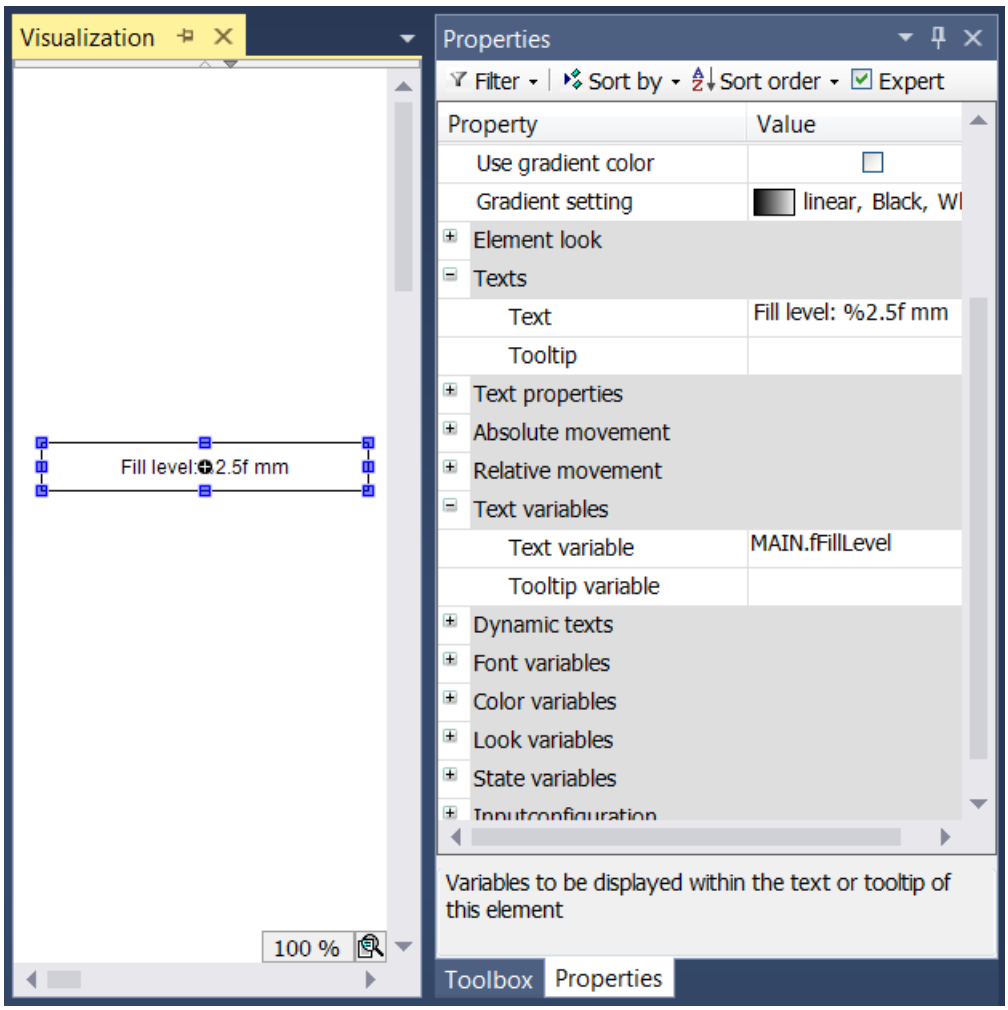
Neben der Eingabe eines reinen Textes in den Elementeigenschaften [▶ 403] in der Kategorie ‚Texte' können auch Formatierungsangaben eingefügt werden, um die Textanzeige im Onlinebetrieb zu gestalten. Eine Formatierungsangabe beginnt immer mit einem „%", gefolgt von einem Zeichen, das die Art der Formatierung bestimmt. Sie kann allein oder in Kombination mit dem eigentlichen Text verwendet werden.

Die Variable, die als Text im Element ausgegeben werden soll, ist in den Elementeigenschaften in der Kategorie ‚Textvariablen' anzugeben. Falls Sie den Instanznamen einer Variablen, die einem Visualisierungsbaustein als Eingangsparameter übergeben wurde, anzeigen lassen wollen, so ist vom dem Pragma Attribut 'parameterstringof [▶ 856]' Gebrauch zu machen.

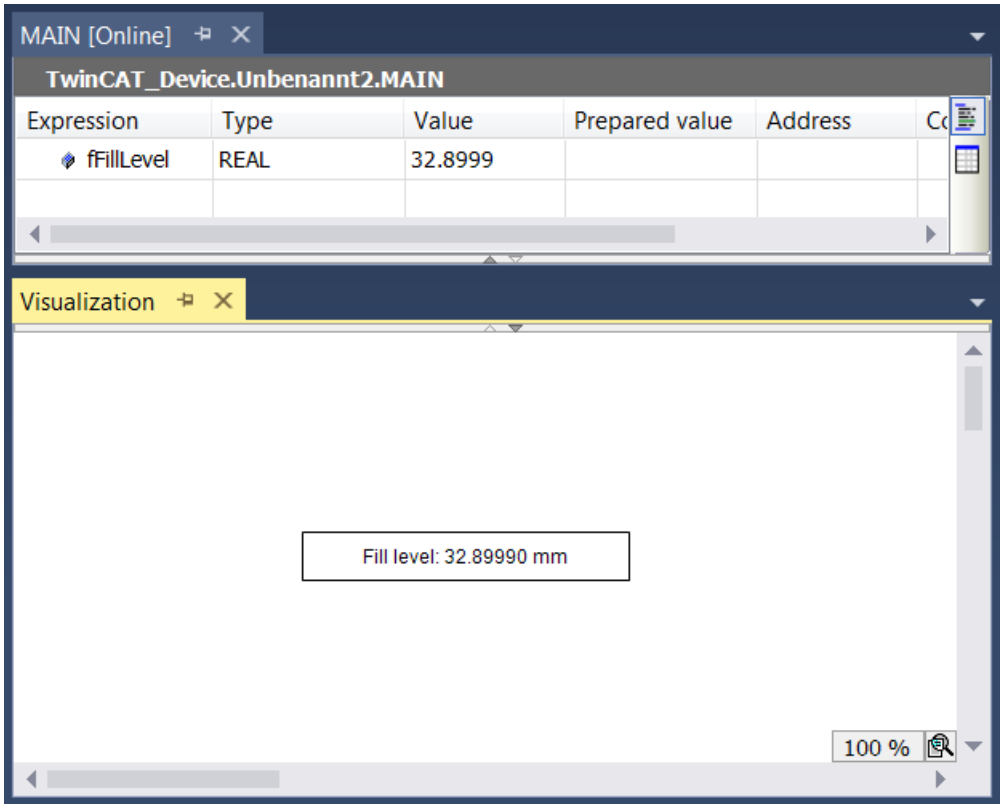
%b	Binärzahl
%c	Einzelnes Zeichen
%d, %i	Dezimalzahl
%f	REAL-Wert
%o	Vorzeichenlose Oktalzahl (ohne vorangestellte Null)
%s	Zeichenfolge
%u	Vorzeichenlose Dezimalzahl
%x	Vorzeichenlose Hexadezimalzahl (ohne vorangestelltes ‚0x')

Beispiel

Im Eigenschaftenfeld "Text" in der Kategorie "Texte" der Eigenschaften [▶ 403] eines Rechteckelements wird "Fill level: %2.5 f mm" eingetragen. Die zu verwendende Variable wird im Eingabefeld "Textvariable" in der Kategorie ‚Textvariablen' eingegeben. In diesem Beispiel ist es "fFillLevel". Sie ist in dem Programm MAIN als REAL deklariert.



Zur Laufzeit sieht das Rechteckelement wie folgt aus:



Wenn ein Prozentzeichen % in Kombination mit einer der oben beschriebenen Formatierungsangaben angezeigt werden soll, muss „%%“ eingegeben werden.

Beispiel

Geben Sie „Rate in %%: %s“ ein, um im Onlinebetrieb folgendes zu erhalten: „Rate in %: 12“ (wenn die Textvariable gerade „12“ liefert).

Ausgabe der Systemzeit

Eine Kombination aus "%t" und nachfolgenden speziellen Platzhaltern innerhalb von eckigen Klammern wird im Onlinebetrieb durch die aktuelle Systemzeit ersetzt. Die Platzhalter definieren das Anzeigeformat.

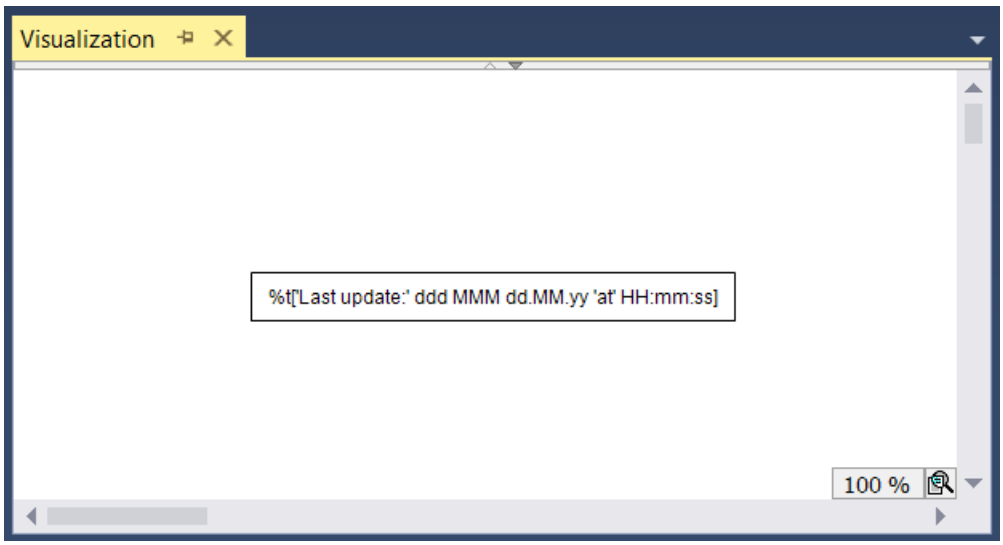
Import von TwinCAT 2 PLC-Control-Projekten: Die frühere verwendete Zeitformatierung %t wird automatisch in das neue %t[] Format konvertiert, wenn ein altes Projekt importiert wird, jedoch werden die folgenden Platzhalter nicht mehr unterstützt: %U, %W, %z, %Z.

Gültige Platzhalter:

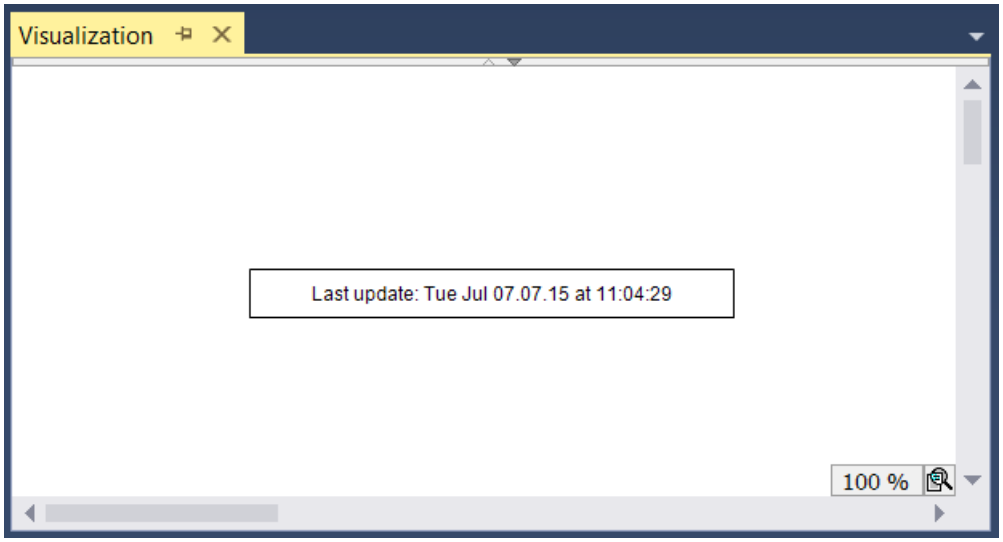
ddd	Name des Wochentags, abgekürzt, z.B. „Wed“
dddd	Name des Wochentags, z.B. „Wednesday“
ddddd	Wochentag als Zahl (0 – 6; Sonntag ist 0)
MMM	Name des Monats, abgekürzt, z.B. „Feb“
MMMM	Name des Monats, z.B. „February“
d	Tag im Monat als Zahl (1 – 31), z.B. „8“
dd	Tag im Monat als Zahl (01 – 31), z.B. „08“
M	Monat als Zahl (1 – 12), z.B. „4“
MM	Monat als Zahl (01 – 12), z.B. „04“
jjj	Tag im Jahr als Zahl (001-366), z.B. „067“
y	Jahr ohne Angabe des Jahrhunderts (0-99), z.B. „9“
yy	Jahr ohne Angabe des Jahrhunderts (00-99), z.B. „09“
yyyy	Jahr mit Angabe des Jahrhunderts, z.B. „2009“
HH	Stunde, 24-Stunden-Format (01-24), z.B. „16“
hh	Stunde, 12-Stunden-Format (01-12), z.B. „4“ für 16 Uhr
m	Minuten (0-59), ohne vorangestellte Null, z.B. „6“
mm	Minuten (00-59), mit vorangestellter Null, z.B. „06“
s	Sekunden (0-59), ohne vorangestellte Null, z.B. „6“
ss	Sekunden (00-59), mit vorangestellter Null, z.B. „06“
ms	Millisekunden (0-999), ohne vorangestellter Null, z.B. „322“
t	Kennzeichner für die Anzeige im 12-Stunden-Format: A (Stunden <12) bzw. P (Stunden >12), z.B. „A“ wenn es 9 Uhr morgens ist
tt	Kennzeichner für die Anzeige im 12-Stunden-Format: AM (Stunden <12) bzw. PM (Stunden >12), z.B. „AM“ wenn es 9 Uhr morgens ist
' '	Zeichenfolgen, die einen der oben aufgeführten Platzhalter enthalten, müssen in einzelne Hochkommas gefasst werden; alle anderen Texte innerhalb des Format-Strings können ohne Hochkommas stehen; z.B. 'update', weil ein "d" und ein "t" enthalten sind

Beispiel

Im Eigenschaftenfeld "Text" in der Kategorie "Texte" der Eigenschaften [▶ 403] eines Rechteckelements wird "%t['Last update:' ddd MMM dd.MM.yy 'at' HH:mm:ss]" eingetragen.



Zur Laufzeit sieht das Rechteckelement wie folgt aus:



Schriftart und Ausrichtung

Die Schriftart und die horizontale und vertikale Ausrichtung des Elementtextes können in den Elementeigenschaften [▶ 403] definiert werden. Siehe hierzu die Kategorien "Schriftartvariablen" und "Texteigenschaften" des entsprechenden Elements.

15.10.3 Bilder

Bilder aus externen Bilddateien werden in SPS-Projekten in Bildersammlungen [▶ 153] verwaltet. Sie können grundsätzlich auf drei verschiedene Arten auf einer Visualisierungsseite [▶ 422] eingefügt werden:

- Visualisierungselement "Bild [▶ 534]"
- Visualisierungselement "Bildwechsler [▶ 556]"
- Hintergrundbild [▶ 395] einer Visualisierungsseite

Visualisierungselement "Bild"

Das Element "Bild [▶ 534]" bietet die Möglichkeit eine externe Bilddatei statisch oder verschiedene externe Bilddateien dynamisch auf einer Visualisierungsseite anzuzeigen. Dazu kann entweder eine statische Bild-ID [▶ 535] oder eine String-Variable [▶ 458], in der zur Laufzeit die ID des aktuell darzustellenden Bildes gespeichert ist, in den Elementeinstellungen eingetragen werden.

Visualisierungselement "Bildwechsler"

Mithilfe des "Bildwechsler [▶ 556]" Elements kann eine benutzerdefinierte Schaltfläche erstellt werden, die vergleichbar zu den Standardschaltern [▶ 556] eine Tasten- oder Umschaltfunktion besitzen kann. In den Elementeigenschaften [▶ 557] können verschiedene Bilder für die Zustände "an", "aus" und "gedrückt" definiert werden.

Hintergrundbild einer Visualisierungsseite

Für jede Visualisierungsseite kann ein Hintergrundbild [▶ 395] eingestellt werden. Es ist möglich, dass die Größe der Visualisierungsseite [▶ 422] im Client sich anhand dieses Hintergrundbilds automatisch anpasst.

Verzeichnisse, die Bilddateien für die Verwendung in der Visualisierung bereitstellen, sind in den Projekteigenschaften [▶ 421] in der Kategorie "Visualization" definiert.

15.10.4 Tastaturbedienung im Onlinebetrieb

Für die Tastaturbedienung einer Visualisierung Im Onlinebetrieb werden einige Standard-Tastaturkürzel von jedem Gerät unterstützt.

Standard-Tastaturkürzel:

Taste(n)	Aktion
[Tab]	Nächstes Element gemäß der chronologischen Reihenfolge der Einfügung wird ausgewählt; innerhalb einer Tabelle wird dabei jede einzelne Zelle angewählt; wenn ein Frame-Element ausgewählt ist, wird die Selektion danach an die einzelnen enthaltenen Elementen weitergegeben
[Shift + Tab]	Vorheriges Element wird ausgewählt; umgekehrte Reihenfolge wie mit [Tab]
[Enter]	Eingabe-Aktion auf das ausgewählte Element wird ausgeführt
[Arrow keys]	Nächstes Element in der durch die Pfeiltaste gegebene Richtung wird ausgewählt

Für die integrierte Visualisierung kann die Tastaturbedienung über den Befehl "Tastaturbedienung aktivieren" explizit aktiviert und deaktiviert werden. Dies kann erwünscht sein, weil solange die Tastaturbedienung für die Visualisierung aktiviert ist, andere Befehle, die über Tastaturkürzel gegeben werden, nicht ausgeführt werden.

Es gibt die Möglichkeit, Tastaturereignisse im Applikationscode zu behandeln.

16 Referenz Programmierung

16.1 Programmiersprachen und ihre Editoren

Sie programmieren eine POU jeweils im Editor für die Implementierungssprache, die Sie beim Anlegen der POU ausgewählt haben. TwinCAT 3 PLC bietet einen Texteditor für ST und grafische Editoren für AS, FUP/KOP/AWL(textuell) und CFC.

Den Editor öffnen Sie mit einem Doppelklick auf die POU im SPS-Projektbaum oder mit dem Befehl **Öffnen** im Kontextmenü.

Jeder der Programmiersprachen-Editoren besteht aus zwei Teilfenstern:

- Im oberen Teil nehmen Sie im Deklarationseditor die Deklarationen vor, je nach Einstellung in textueller oder tabellarischer Form.
- Im unteren Teil fügen Sie den Implementierungscode in der jeweiligen Sprache ein.

Die Darstellung und das Verhalten jedes Editors können Sie projektweit in der zugehörigen Registerkarte der TwinCAT-Optionen konfigurieren.

16.1.1 Deklarationseditor

Im Deklarationseditor deklarieren Sie Variablen in Variablenlisten und POUs.

Wenn der Deklarationseditor in Verbindung mit einem Programmierspracheneditor verwendet wird, öffnet er sich in einem Fenster oberhalb des Programmierspracheneditors.

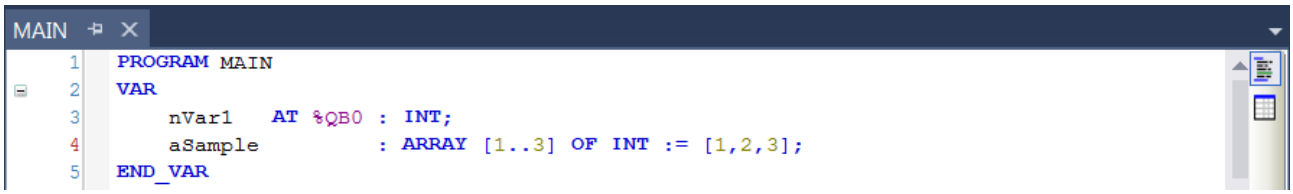
Der Deklarationseditor bietet zwei mögliche Ansichten: textuell( ) und tabellarisch ( ). Im Dialog **Extras > Optionen > TwinCAT > SPS Programmierumgebung > Deklarationseditor** definieren Sie, ob entweder nur die textuelle oder nur die tabellarische Ansicht verfügbar ist, oder ob Sie über die Schaltflächen am rechten Rand des Editorfensters zwischen den beiden Ansichten wechseln können.

In der textuellen Ansicht des Deklarationseditors ist eine Rechteckselektion möglich. Die Tastenkombinationen für die Rechteckselektion finden Sie im Kapitel ST-Editor [▶ 656].

Textueller Deklarationseditor

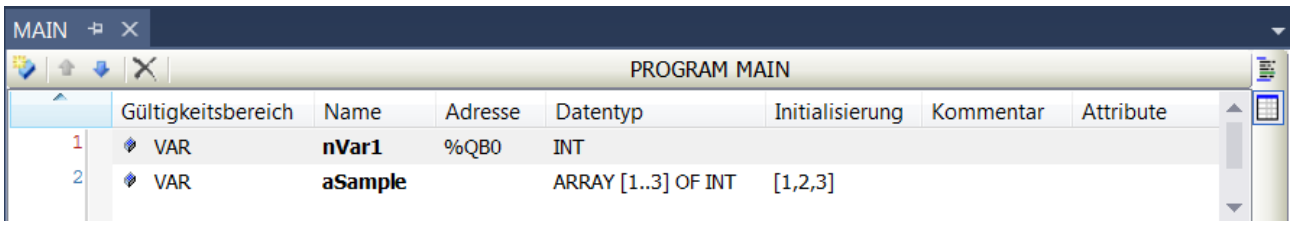
Das Verhalten und das Aussehen des textuellen Editors konfigurieren Sie mit den Einstellungen im Dialog **Extras > Optionen > TwinCAT > SPS Programmierumgebung > Texteditor**. Die Einstellungen betreffen Farben, Zeilennummern, Tab-Breiten, Einrückungen etc. Im textuellen Editor stehen die üblichen Windows-Funktionen zur Verfügung, gegebenenfalls auch die IntelliMouse-Funktionen.

Im textuellen Deklarationseditor können Sie auch Kommentare verwenden (siehe ST-Kommentare [▶ 667]).



Tabellarischer Deklarationseditor

Im tabellarischen Deklarationseditor fügen Sie Variablendeklarationen in einer Tabelle mit den folgenden Spalten ein: Gültigkeitsbereich, Name, Adresse, Datentyp, Initialisierung, Kommentar und Attribute (Pragmas).



Deklarationseditor im Onlinebetrieb

Im Onlinebetrieb sehen Sie die tabellarische Ansicht des Editors. Die Kopfzeile enthält immer den aktuellen Objektpfad: <Gerätename>.<Projektname>.<Objektname>. Die Tabelle enthält im Unterschied zum Offlinebetrieb auch die Spalten **Wert** und **Vorbereiteter Wert**.

Wert	Zeigt den Istwert auf der Steuerung, bietet also Monitoring-Funktionalität. Wenn der Ausdruck ein Array mit mehr als 1000 Elementen ist, können Sie den Bereich der zu monitorenden Arrayindizes festlegen. Dazu öffnen Sie mit einem Doppelklick in die Spalte **Datentyp** den Dialog Monitoringbereich [▶ 236]. In diesem Dialog ist der deklarierte Arraybereich als Gültiger Bereich für das Monitoring eingetragen. Pro Array können maximal 20000 Elemente angezeigt werden. Den Bereich der anzuzeigenden Arrayindizes definieren Sie durch Angabe von Start- und End-Index. Um diesen Bereich einfacher bei gleichbleibendem Umfang zu verschieben, können Sie die dafür vorhandenen Scrollbalken gekoppelt verwenden. Zum Umschalten zwischen gekoppelten Scrollbalken und nicht gekoppelten Scrollbalken klicken Sie auf das Symbol rechts der Balken. Im nicht gekoppelten Zustand können Sie den Umfang des anzuzeigenden Bereichs beliebig vergrößern oder verkleinern.
Vorbereiteter Wert	Enthält den Wert, den Sie gegebenenfalls zum Forcen oder Schreiben [▶ 225] vorbereitet haben. Wenn Sie auf ein Feld in der Spalte **Vorbereiteter Wert** doppelklicken, können Sie direkt einen Wert für das Schreiben oder Forcen eingeben. Bei Enumerationen öffnet sich eine Combobox, aus der Sie einen Wert auswählen können. Bei einer booleschen Variablen können Sie den vorbereiteten Wert mithilfe der Tasten **[Eingabe]** oder **[Leerzeichen]** toggeln (umschalten). Wenn ein Ausdruck (Variable) von einem strukturierten Datentyp ist, beispielsweise die Instanz eines Funktionsbausteins oder eine Arrayvariable, dann ist ein Plus- oder Minuszeichen vorangestellt. Das Format der Darstellung von Gleitpunkten sowie die Darstellung der Vererbungshierarchie können Sie im Kontextmenü anpassen.

Siehe auch:

- Variablen deklarieren [▶ 68]
- Forcen im Deklarationsteil [▶ 228]
- Dokumentation TC3 User Interface: Dialog Optionen - Deklarationseditor [▶ 1013]
- Dokumentation TC3 User Interface: Dialog Wert vorbereiten [▶ 1006]

16.1.2 Generelle Funktionalitäten in allen grafischen Editoren

Der Implementierungsteil der grafischen Editoren für FUP, KOP, CFC und AS enthält in der rechten unteren Ecke eine Symbolleiste:

	Zurück in den normalen Editiermodus: Der Mauszeiger bekommt die standardmäßige Pfeilform. Sie können wieder Elemente im Editorfenster auswählen und bearbeiten.
	Schwenkwerkzeug: Der Mauszeiger bekommt die Form zweier gekreuzter Pfeile. Sie können nun an beliebiger Stelle ins Editorfenster klicken, die Maustaste gedrückt halten und dann mit der Maus den sichtbaren Bereich des FUP/KOP/AWL-Editors oder des CFC-Diagramms innerhalb des Fensters schwenken/verschieben.
	Lupenwerkzeug: Ein zusätzliches Lupenfenster öffnet sich in der unteren rechten Ecke des Editorfensters und der Mauszeiger bekommt die Form zweier gekreuzter Pfeile. Solange Sie nun den Mauszeiger über Ihr Diagramm bewegen, zeigt die Lupe jeweils den Bereich des Diagramms um den Zeiger 100% Größe. Wenn Sie nun ins Fenster klicken, wird die Lupe geschlossen und der Teil des Diagramms, der in ihr zu sehen gewesen war, in 100% Größe angezeigt. Wenn Sie den vorher eingestellten Zoomfaktor beibehalten wollen, sollten Sie deshalb  verwenden, um zurück in den normalen Editiermodus zu gelangen.
	Vergrößerungs-/Verkleinerungswerkzeug: Die Schaltfläche öffnet ein Untermenü zur Auswahl eines Zoomfaktors. Die Auswahl der drei Punkte ... öffnet den Dialog Zoom, in dem Sie einen anderen Wert angeben können. Der aktuelle Zoomfaktor wird immer links von der Schaltfläche angezeigt.

Vergrößern/Verkleinern mit dem Mausrad: Wenn Sie die Taste **[Strg]** drücken, während Sie das Mausrad bewegen, wird der Zoomfaktor stufenweise um 10% verändert.

Jeder grafische Editor bringt eine Toolbox (Ansicht **Werkzeugkasten**) mit, standardmäßig rechts vom Editorfenster. Die Toolbox enthält die verfügbaren Elemente, die Sie mit der Maus auf die jeweiligen Einfügepositionen ins Editorfenster ziehen können. Die jeweiligen Einfügepositionen hebt TwinCAT mit grauen rauten-, dreieck- oder pfeilförmigen Positionsmarken hervor. Sobald der Mauszeiger über einer dieser Marken steht, wird diese Marke grün. Wenn Sie nun die Maustaste loslassen, fügt TwinCAT das Element an der aktuellen Position ein.

Das Verschieben von Elementen im Editor ist ebenfalls mit der Maus möglich.

Sie können in den grafischen Editoren FUP, KOP und CFC Deklarationen von Funktionsbausteinen in das Editorfenster ziehen. Selektieren Sie dazu die vollständige Deklaration (Variablenname und Datentyp) und ziehen Sie diese auf eine geeignete Stelle im Editorfenster. Beim Kontaktplan können Sie zusätzlich boolesche Deklarationen in den Editor ziehen und als Kontakt einfügen.

Siehe auch:

- AS-Editor [▶ 667]
- FUP/KOP/AWL-Editor [▶ 685]
- CFC-Editor [▶ 701]

16.1.3 Strukturierter Text und Erweiterter Strukturierter Text (ExST)

16.1.3.1 ST-Editor

Der ST-Editor ist ein Texteditor und dient der Implementierung von Code in strukturiertem Text (ST) und Erweitertem Strukturiertem Text (ExST).

Am linken Rand des Editors befindet sich die Zeilennummerierung. Bei der Eingabe der Programmierelemente helfen Ihnen die Funktion **Komponenten auflisten** und die Eingabehilfe (**F2**). Wenn der Cursor auf einer Variablen steht, zeigt TwinCAT im Tooltip Informationen zur Deklaration der Variablen an.

Sie können eine Rechteckselektion mit folgenden Tastenkombinationen ausführen:

- **[Umschalt]** + **[Alt]** + **[Pfeil nach rechts]**: Der selektierte Bereich wird um eine Stelle nach rechts erweitert.
- **[Umschalt]** + **[Alt]** + **[Pfeil nach links]**: Der selektierte Bereich wird um eine Stelle nach links erweitert.

- **[Umschalt]** + **[Alt]** + **[Pfeil nach oben]**: Der selektierte Bereich wird um eine Zeile nach oben erweitert.
- **[Umschalt]** + **[Alt]** + **[Pfeil nach unten]**: Der selektierte Bereich wird um eine Stelle nach unten erweitert.

Zoomen im Editorfenster ist durch Drehen des Mausrads bei gedrückter Taste **Strg** möglich.

Das Verhalten (wie zum Beispiel Klammerung, Mausaktionen, Tabulatoren) und Aussehen des Editors konfigurieren Sie in den TwinCAT-Optionen in der Kategorie **SPS Programmierumgebung > Texteditor**.

Für eine inkrementelle Suche nach Zeichenfolgen innerhalb des Editors öffnen Sie mit der Tastenkombination **[Strg]** + **[Umschalt]** + **[i]** ein Eingabefeld am unteren Rand des Editors. Sobald Sie beginnen, eine Zeichenfolge einzutippen, werden im Editor die entsprechenden Fundstellen farbig markiert. Rechts des Eingabefelds steht die Anzahl der gefundenen Übereinstimmungen. Mit Hilfe der Pfeilschaltflächen oder den Tastenkombinationen **[Alt]** + **[Bild auf]** oder **[Alt]** + **[Bild ab]** können Sie den Cursor auf eine Fundstelle setzen.

Wenn Sie den Cursor auf einen Symbolnamen setzen, werden alle Verwendungsstellen des Symbols innerhalb des Editors farblich hervorgehoben. Die Fundstellen entsprechen den Treffern der Querverweisliste. Bei sehr großen Projekten kann dies zu Verzögerungen bei der Eingabe führen. In diesem Fall können Sie die Funktion in den Optionen des Texteditors abschalten.

TwinCAT erkennt Syntaxfehler bereits während des Editiervorgangs und unterkringelt die Fehlerstellen rot. Voraussetzung ist, dass die entsprechende Option in den TwinCAT-Optionen, Kategorie **SPS Programmierumgebung > Intelligentes Kodieren** aktiviert ist. Während der Übersetzung ermittelte syntaktische Fehler werden in den Meldungen in der Ansicht **Fehlerliste** (Menü **Ansicht**) angezeigt.

Siehe auch:

- Programmieren in Strukturiertem Text (ST) [▶ 129]
- ST-Ausdrücke [▶ 657]
- Zuweisungen [▶ 659]
- Anweisungen [▶ 662]
- Dokumentation TC3 User Interface: Dialog Optionen - Texteditor [▶ 1030]

16.1.3.2 ST-Editor im Onlinebetrieb

Im Onlinebetrieb zeigt TwinCAT im ST-Editor die verwendeten Variablen und Ausdrücke an. Das Schreiben und Forcen der Variablen sowie Ausdrücke und Debugging-Funktionen (Haltepunkte, Einzelschrittabarbeitung) sind ebenfalls möglich.

Wenn Sie in der ST-Programmierung Zuweisungen als Ausdrücke verwenden, entstehen keine weiteren Haltepunkt-Positionen innerhalb einer Zeile.

Siehe auch:

- Monitoring in Programmierobjekten aufrufen [▶ 234]
- Haltepunkte verwenden [▶ 222]
- Überwachungslisten verwenden [▶ 239]
- Forcen und Schreiben von Variablen [▶ 225]
- SPS-Projekt testen und Fehler beheben [▶ 222]
- Schrittweises Abarbeiten eines Programms (Stepping) [▶ 224]
- Ablaufkontrolle [▶ 231]
- Aktuelle Abarbeitungsposition mit der Aufrufliste bestimmen [▶ 232]

16.1.3.3 ST-Ausdrücke

Ein Ausdruck ist ein Konstrukt, das nach seiner Auswertung einen Wert zurück liefert.

Ausdrücke sind zusammengesetzt aus Operatoren und Operanden. Ein Operand kann eine Konstante, eine Variable, ein Funktionsaufruf oder ein weiterer Ausdruck sein.

Bespiele:

2014	(* Konstante *)
nVar	(* Variable *)
F_Fct(a,b)	(* Funktionsaufruf *)
(x*y)/z	(* Ausdruck *)

Siehe auch:

- Operatoren [▶ 732]
- Operanden [▶ 780]

Auswertung von Ausdrücken

Die Auswertung eines Ausdrucks erfolgt durch Abarbeitung der Operatoren nach bestimmten Bindungsregeln. TwinCAT arbeitet den Operator mit der stärksten Bindung zuerst ab. Operatoren mit gleicher Bindungsstärke werden von links nach rechts abgearbeitet.

Operation	Symbol	Bindungsstärke
Einklammern	(Ausdruck)	Stärkste Bindung
Funktionsaufruf	Funktionsname (Parameterliste) alle Operatoren mit Syntax: <operator> ()	
Potenzieren	EXPT	
Negieren Komplementbildung	- NOT	
Multiplizieren Dividieren Modulo	* / MOD	
Addieren Subtrahieren	+ -	
Vergleichen	<,>,<=,>=	
Gleichheit Ungleichheit	= <>	
Bool AND	AND AND_THEN	
Bool XOR Bool OR	XOR OR OR_ELSE	Schwächste Bindung

Beispiel:

Im folgenden Beispiel werden die Operatoren AND_THEN [▶ 749] und OR [▶ 750] verwendet. Zu beachten ist, dass OR die schwächste Bindung hat und dass TwinCAT die Ausdrücke an weiteren Operanden des AND_THEN-Operators nur ausführt, wenn der erste Operand des AND_THEN-Operators TRUE ist.

Daher ergibt sich bei den vier dargestellten Ausdrücken folgendes:

- Bei keinem der vier Ausdrücke wird der Zeiger „pSample“ dereferenziert. Hintergrund ist, dass der AND_THEN-Operator verwendet wird und die Dereferenzierung an den weiteren Operanden des AND_THEN-Operators stattfinden würde. Da aber bereits der erste Operand des AND_THEN-Operators FALSE liefert (da „pSample“ den Wert 0 hat), werden die weiteren AND_THEN-Operanden und damit die Zeiger-Dereferenzierungen nicht ausgeführt.
  Aufgrund der Verwendung des AND_THEN-Operators wird somit eine Nullpointer-Exception zur

Laufzeit vermieden. Wenn anstelle des AND_THEN-Operators der AND-Operator verwendet werden würde, würde innerhalb der Operanden der Zeiger „pSample“ als Nullpointer dereferenziert, was eine Nullpointer-Exception zur Folge hätte.

- Bei den Ausdrücken 1 und 2 folgt OR als weiterer Operator hinter den AND_THENs. Daher inkrementieren die Zähler „nCounter1“ und „nCounter2“, wenn „pSample“ 0 und „bTest“ TRUE ist. Die Kurzform der Ausdrücke lautet „IF <FALSE> OR TRUE THEN“, welche TRUE liefert.
- Bei den Ausdrücken 3 und 4 hingegen inkrementieren „nCounter3“ und „nCounter4“ nicht, wenn „pSample“ 0 und „bTest“ TRUE ist, da bereits der erste Operand „pSample <> 0“ FALSE liefert. Aufgrund des AND_THEN und der Klammerung werden keine weiteren Operanden oder Operatoren überprüft. Die Kurzform der Ausdrücke lautet „IF <FALSE> AND_THEN <…>“, welche FALSE liefert.

```
PROGRAM MAIN
VAR
    pSample    : POINTER TO INT;
    bTest      : BOOL := TRUE;
    nCounter1  : INT;
    nCounter2  : INT;
    nCounter3  : INT;
    nCounter4  : INT;
END_VAR
```

```
// Expression 1
IF (pSample <> 0) AND_THEN (pSample^ = 250) AND_THEN (pSample^ <> 300) OR bTest THEN
    nCounter1 := nCounter1 + 1;   // increasing if (pSample = 0) and (bTest = TRUE)
END_IF

// Expression 2
IF ((pSample <> 0) AND_THEN (pSample^ = 250) AND_THEN (pSample^ <> 300)) OR bTest THEN
    nCounter2 := nCounter2 + 1;   // increasing if (pSample = 0) and (bTest = TRUE)
END_IF

// Expression 3
IF (pSample <> 0) AND_THEN ((pSample^ = 250) AND_THEN (pSample^ <> 300) OR bTest) THEN
    nCounter3 := nCounter3 + 1;   // not increasing if (pSample = 0) and (bTest = TRUE)
END_IF

// Expression 4
IF (pSample <> 0) AND_THEN ((pSample^ = 250) OR bTest) THEN
    nCounter4 := nCounter4 + 1;   // not increasing if (pSample = 0) and (bTest = TRUE)
END_IF
```

16.1.3.4 Zuweisungen

16.1.3.4.1 ST-Zuweisungsoperator

Syntax:

<operand> := <expression>

Dieser Zuweisungsoperator führt die gleiche Funktion aus wie der MOVE-Operator.

Siehe auch:

- MOVE [▶ 742]

16.1.3.4.2 ST-Zuweisungsoperator für Ausgänge

Der Zuweisungsoperator => weist den Ausgang einer Funktion, eines Funktionsbausteins oder einer Methode einer Variablen zu. Der Platz rechts vom Operator kann auch leer sein.

Syntax:

<output> => <variable>

Beispiel:

```
bFBCompOutput1 => bVar1;
bFBCompOutput2 => ;
```

bFBCompOutput1 und bFBCompOutput2 sind Ausgänge eines Funktionsbausteins. Der Wert von bFBCompOutput1 wird der Variablen bVar1 zugewiesen.

16.1.3.4.3 ExST-Zuweisung S=

Wenn der Operand der Set-Zuweisung auf TRUE schaltet, bewirkt die Zuweisung, dass der Variablen links des Operators ein TRUE zugewiesen wird. Die Variable wird gesetzt.

Syntax:

<variable name> S= <operand name> ;

Die Variable und der Operand haben den Datentyp BOOL.

Beispiel:

```
PROGRAM MAIN
VAR
    bOperand     : BOOL := FALSE;
    bSetVariable : BOOL := FALSE;
END_VAR
```

```
bSetVariable S= bOperand;
```

Wenn der Operand bOperand von FALSE auf TRUE schaltet, wird der Variablen bSetVariable ein TRUE zugewiesen. Dann aber behält die Variable diesen Zustand, auch wenn der Operand weiterhin seinen Zustand wechselt.

Mehrfachzuweisungen

Bei Mehrfachzuweisungen innerhalb einer Codezeile werden die einzelnen Zuweisungen nicht von rechts nach links abgearbeitet, sondern alle Zuweisungen beziehen sich auf den Operanden am Codezeilenende.

Beispiel:

```
FUNCTION F_Sample: BOOL
VAR_INPUT
    bIn : BOOL;
END_VAR
```

```
IF bIn = TRUE THEN
    F_Sample := TRUE;
    RETURN;
END_IF
```

```
PROGRAM MAIN
VAR
    bSetVariable   : BOOL;
    bResetVariable : BOOL := TRUE;
    bVar           : BOOL;
END_VAR
```

```
bSetVariable S= bResetVariable R= F_Sample(bIn := bVar);
```

bResetVariable erhält die R=-Zuweisung des Rückgabewerts von F_Sample. bSetVariable erhält die S=-Zuweisung des Rückgabewerts von F_Sample, aber nicht von bResetVariable.

16.1.3.4.4 ExST-Zuweisung R=

Wenn der Operand der Reset-Zuweisung auf TRUE schaltet, bewirkt die Zuweisung, dass der Variablen links des Operators ein FALSE zugewiesen wird. Die Variable wird zurückgesetzt.

Syntax:

<variable name> R= <operand name> ;

Die Variable und der Operand haben den Datentyp BOOL.

Beispiel:

```
PROGRAM MAIN
VAR
    bOperand       : BOOL := FALSE;
    bResetVariable : BOOL := TRUE;
END_VAR
```

```
bResetVariable R= bOperand;
```

Wenn der Operand bOperand von FALSE auf TRUE schaltet, wird der Variablen bResetVariable ein FALSE zugewiesen. Dann aber behält die Variable ihren Zustand, auch wenn der Operand weiterhin seinen Zustand wechselt.

Mehrfachzuweisungen

Bei Mehrfachzuweisungen innerhalb einer Codezeile werden die einzelnen Zuweisungen nicht von rechts nach links abgearbeitet, sondern alle Zuweisungen beziehen sich auf den Operanden am Codezeilenende.

Beispiel:

```
FUNCTION F_Sample: BOOL
VAR_INPUT
    bIn : BOOL;
END_VAR
```

```
IF bIn = TRUE THEN
    F_Sample := TRUE;
    RETURN;
END_IF
```

```
PROGRAM MAIN
VAR
    bSetVariable   : BOOL;
    bResetVariable : BOOL := TRUE;
    bVar           : BOOL;
END_VAR
```

```
bSetVariable S= bResetVariable R= F_Sample(bIn := bVar);
```

bResetVariable erhält die R=-Zuweisung des Rückgabewerts von F_Sample. bSetVariable erhält die S=-Zuweisung des Rückgabewerts von F_Sample, aber nicht von bResetVariable.

16.1.3.4.5 ExST-Zuweisung als Ausdruck

Im ExST erlaubt TwinCAT in Erweiterung zur Norm IEC 61131-3 die Verwendung von Zuweisungen als Ausdrücke.

Beispiele:

`nVarInt1 := nVarInt2 := nVarInt3 + 9;`	(*nVarInt1und nVarInt2 erhalten Wert von nVarInt3 + 9*)
`fVarReal1 := fVarReal2 := nVarInt;`	(*fVarReal1 und fVarReal2 erhalten den Wert von nVarInt*)
`nVarInt:= fVarReal1:= nVarInt;`	(*fehlerhafte Zuweisung, die Datentypen stimmen nicht überein!*)

`IF b := (i = 1) THEN`
`  i := i + 1;`
`END_IF`
	(*b erhält den Wert des booleschen Ausdrucks i=1 und wird daraufhin in der If-Abfrage geprüft*)

16.1.3.4.6 Zuweisungsoperator REF=

Der Operator erzeugt eine Referenz [▶ 807] (Pointer) auf einen Wert.

Syntax:

```
<variable name> REF= <variable name>
```

Beispiel:

```
PROGRAM MAIN
VAR
    refA  : REFERENCE TO ST_Sample;
    stA   : ST_Sample;
    refB  : REFERENCE TO ST_Sample;
    stB1  : ST_Sample;
    stB2  : ST_Sample;
END_VAR
```

```
refA REF= stA;   // represents => refA  := ADR(stA);
refB REF= stB1;  // represents => refB  := ADR(stB1);
refA := refB;    // represents => refA^ := refB^; (value assignment of refB as refA and refB are
implicitly dereferenced)
refB := stB2;    // represents => refB^ := stB2; (value assignment of stB2 as refB is implicitly
dereferenced)
END_VAR
```

Siehe auch:

- REFERENCE [▶ 807]

16.1.3.5 Anweisungen

16.1.3.5.1 ST-Anweisung IF

Die IF-Anweisung verwenden Sie, um eine Bedingung zu prüfen und bei erfüllter Bedingung die nachfolgenden Anweisungen auszuführen.

Eine Bedingung wird als Ausdruck [▶ 657] codiert, der einen booleschen Wert zurückliefert. Wenn der Ausdruck TRUE liefert, ist die Bedingung erfüllt und die zugehörigen Anweisungen nach THEN werden ausgeführt. Wenn der Ausdruck FALSE liefert, werden die folgenden Bedingungen, die mit ELSIF gekennzeichnet sind, ausgewertet. Wenn eine ELSIF-Bedingungen TRUE liefert, werden die Anweisungen nach dem zugehörigen THEN ausgeführt. Wenn alle Bedingungen FALSE sind, werden die Anweisungen nach ELSE ausgeführt.

Es wird also höchstens ein Zweig der IF-Anweisung ausgeführt. Die ELSIF-Zweige und der ELSE-Zweig sind optional.

Syntax

```
IF <condition> THEN
    <statements>
(ELSIF <condition> THEN
    <statements>)*
(ELSE
    <statements>)?
END_IF;

// ( ... )* None, once or several times
// ( ... )? Optional
```

Beispiel:

```
PROGRAM MAIN
VAR
    nTemp       : INT;
    bHeatingOn  : BOOL;
    bOpenWindow : BOOL;
END_VAR
```

```
IF nTemp < 17 THEN
    bHeatingOn  := TRUE;
ELSIF nTemp > 25 THEN
    bOpenWindow := TRUE;
ELSE
    bHeatingOn  := FALSE;
    bOpenWindow := FALSE;
END_IF;
```

Zur Laufzeit wird das Programm folgendermaßen durchlaufen:

Wenn die Auswertung des Ausdrucks nTemp < 17 = TRUE ergibt, wird die nachfolgende Anweisung ausgeführt und die Heizung eingeschaltet. Wenn die Auswertung des Ausdrucks nTemp < 17 = FALSE ergibt, wird die nachfolgenden ELSIF-Bedingung nTemp > 25 ausgewertet. Wenn diese wahr ist, wird die Anweisung unter ELSIF ausgeführt und das Fenster geöffnet. Wenn alle Bedingungen FALSE sind, werden die Anweisungen unter ELSE ausgeführt. Die Heizung wird ausgeschaltet und das Fenster wird geschlossen.

Siehe auch:

- ST-Ausdrücke [▶ 657]

16.1.3.5.2 ST-Anweisung FOR

Die FOR-Schleife verwenden Sie, um Anweisungen mit einer bestimmten Anzahl von Wiederholungen auszuführen.

Syntax:

```
FOR <counter> := <start value> TO <end value> {BY <increment> } DO
   <instructions>
END_FOR;
```

Der Abschnitt innerhalb der geschweiften Klammern {} ist optional.

TwinCAT führt die <instructions> solange aus, wie der <counter> nicht größer, oder - bei negativer Schrittgröße increment - kleiner als der <end value> ist. Dies wird vor der Ausführung der <instructions> geprüft.

Immer wenn die Anweisungen <instructions> ausgeführt worden sind, wird der Zähler <counter> automatisch um die Schrittgröße <increment> erhöht. Die Schrittgröße <increment> kann jeden ganzzahligen Wert haben. Wenn Sie keine Schrittgröße angeben, ist die Standard-Schrittgröße 1.

Beispiel:

```
FOR nCounter := 1 TO 5 BY 1 DO
    nVar1 := nVar1*2;
END_FOR;
nErg := nVar1;
```

Wenn Sie nVar1 mit 1 vorbelegt haben, hat nVar1 nach der FOR-Schleife den Wert 32.

Endwert der FOR-Schleife

Der Endwert <end value> darf nicht den gleichen Wert erhalten wie die Obergrenze des Datentyps des Zählers.

In Erweiterung zum Standard IEC 61131-3 können Sie innerhalb der FOR-Schleife die CONTINUE-Anweisung verwenden.

Siehe auch:

- ExST-Anweisung CONTINUE [▶ 666]
- Ganzzahlige Datentypen [▶ 794]

16.1.3.5.3 ST-Anweisung CASE

Die CASE-Anweisung verwenden Sie, um mehrere bedingte Anweisungen mit derselben Bedingungsvariablen in einem Konstrukt zusammenzufassen.

Syntax:

```
CASE <Var1> OF
<value1>:<instruction1>
<value2>:<instruction2>
<value3, value4, value5>:<instruction3>
<value6 ... value10>:<instruction4>
...
```

<value n>:<instruction n>
{ELSE <ELSE-instruction>}
END_CASE;

Der Abschnitt innerhalb der geschweiften Klammer {} ist optional.

Abarbeitungsschema einer CASE-Anweisung:

- Wenn die Variable <Var1> den Wert <<value i> hat, wird die Anweisung <instruction i> ausgeführt.
- Wenn die Variable <Var1> keinen der angegebenen Werte hat, dann wird die <ELSE-instruction> ausgeführt.
- Wenn für mehrere Werte der Variablen dieselbe Anweisung auszuführen ist, können Sie diese Werte durch Kommata getrennt hintereinander schreiben.

Beispiel:

```
CASE nVar OF
     1,5 : bVar1 := TRUE;
         bVar3 := FALSE;

     2 : bVar2 := FALSE;
         bVar3 := TRUE;

10..20 : bVar1 := TRUE;
         bVar3 = TRUE;
ELSE
         bVar1 := NOT bVar1;
         bVar2 := bVar1 OR bVar2;
END_CASE;
```

16.1.3.5.4 ST-Anweisung WHILE

Die WHILE-Schleife verwenden Sie wie die FOR-Schleife, um Anweisungen mehrfach auszuführen, bis die Abbruchbedingung zutrifft. Die Abbruchbedingung einer WHILE-Schleife ist ein boolescher Ausdruck.

Syntax:

WHILE <boolean expression> DO
  <instructions>
END_WHILE;

TwinCAT führt die Anweisungen <instructions> solange wiederholt aus, wie der boolesche Ausdruck <boolean expression> TRUE ergibt. Wenn der boolesche Ausdruck bereits bei der ersten Auswertung FALSE ist, dann führt TwinCAT die Anweisungen niemals aus. Wenn der boolesche Ausdruck niemals den Wert FALSE annimmt, werden die Anweisungen endlos wiederholt, wodurch ein Laufzeitfehler entsteht.

Beispiel:

```
WHILE nCounter <> 0 DO
    nVar1 := nVar1*2
    nCounter := nCounter-1;
END_WHILE;
```

Sie müssen programmatisch sicherstellen, dass keine Endlos-Schleife verursacht wird.

Die WHILE- und die REPEAT-Schleifen sind in gewissem Sinne mächtiger als die FOR-Schleife, da man nicht bereits vor der Ausführung der Schleife die Anzahl der Schleifendurchläufe wissen muss. In manchen Fällen ist es somit nur möglich, mit diesen beiden Schleifenarten zu arbeiten. Wenn jedoch die Anzahl der Schleifendurchläufe klar ist, dann ist eine FOR-Schleife zu bevorzugen, um Endlosschleifen zu vermeiden.

In Erweiterung zum Standard IEC 61131-3 können Sie innerhalb der WHILE-Schleife die CONTINUE-Anweisung verwenden.

Siehe auch:

- ExST-Anweisung CONTINUE [▶ 666]
- ST-Anweisung FOR [▶ 663]

16.1.3.5.5 ST-Anweisung REPEAT

Die REPEAT-Schleife verwenden Sie wie die WHILE-Schleife, jedoch mit dem Unterschied, dass TwinCAT die Abbruchbedingung erst nach dem Ausführen der Schleife überprüft. Dieses Verhalten hat zur Folge, dass die REPEAT-Schleife mindestens einmal durchlaufen wird, egal wie die Abbruchbedingung lautet.

Syntax:

```
REPEAT
   <instructions>
UNTIL <boolean expression>
END_REPEAT;
```

TwinCAT führt die Anweisungen <instructions> solange aus, bis der boolesche Ausdruck <boolean expression> TRUE ergibt.

Wenn der boolesche Ausdruck bereits bei der ersten Auswertung TRUE ergibt, dann führt TwinCAT die Anweisungen genau einmal aus. Wenn der boolesche Ausdruck niemals den Wert TRUE annimmt, werden die Anweisungen endlos wiederholt, wodurch das Programm einen Laufzeitfehler verursacht.

Beispiel:

```
REPEAT
    nVar1 := nVar1*2;
    nCounter := nCounter-1;
UNTIL
    nCounter = 0
END_REPEAT;
```

Die WHILE- und die REPEAT-Schleifen sind in gewissem Sinne mächtiger als die FOR-Schleife, da die Anzahl der Schleifendurchläufe nicht bereits vor Ausführung der Schleife bekannt sein muss. In manchen Fällen können Sie nur mit diesen beiden Schleifenarten arbeiten. Wenn jedoch die Anzahl der Schleifendurchläufe klar ist, dann ist eine FOR-Schleife zu bevorzugen, um Endlosschleifen zu vermeiden.

In Erweiterung zum Standard IEC 61131-3 können Sie innerhalb der WHILE-Schleife die CONTINUE-Anweisung verwenden.

Siehe auch:

- ExST-Anweisung CONTINUE [▶ 666]
- ST-Anweisung FOR [▶ 663]
- ST-Anweisung WHILE [▶ 664]

16.1.3.5.6 ST-Anweisung RETURN

Die RETURN-Anweisung verwenden Sie, um einen Funktionsbaustein zu verlassen. Dies können Sie beispielsweise von einer Bedingung abhängig machen.

Beispiel:

```
IF bVar1 = TRUE THEN
    RETURN;
END_IF;
nVar2 := nVar2 + 1;
```

Wenn der Wert von bVar1 TRUE ist, wird der Funktionsbaustein sofort verlassen und TwinCAT führt die Anweisung `nVar2 := nVar2 + 1;` nicht aus.

Siehe auch:

- ST-Anweisung IF [▶ 662]

16.1.3.5.7 ST-Anweisung JMP

Die JMP-Anweisung wird verwendet, um einen unbedingten Sprung zu einer Programmzeile auszuführen, die durch eine Sprungmarke gekennzeichnet ist.

Syntax:

<label>: <instructions>
JMP <label>;

Die Sprungmarke <label> ist ein beliebiger, eindeutiger Bezeichner, den Sie an den Anfang einer Programmzeile stellen. Bei Erreichen der JMP-Anweisung erfolgt ein Rücksprung zu der Programmzeile mit der <Sprungmarke>.

Beispiel:

```
nVar1 := 0;
_label1 : nVar1 := nVar1+1;
(*instructions*)
IF (nVar1 < 10) THEN
    JMP _label1;
END_IF;
```

Sie müssen programmatisch sicherstellen, dass keine Endlosschleife verursacht wird. Beispielsweise können Sie den Sprung an eine Bedingung knüpfen.

16.1.3.5.8 ST-Anweisung EXIT

Die EXIT-Anweisung verwenden Sie in einer FOR-, WHILE- oder REPEAT-Schleife, um die Schleife zu beenden, ungeachtet anderer Abbruchbedingungen.

Siehe auch:

- ST-Anweisung FOR [▶ 663]
- ST-Anweisung REPEAT [▶ 665]
- ST-Anweisung WHILE [▶ 664]

16.1.3.5.9 ExST-Anweisung CONTINUE

CONTINUE ist eine Anweisung des „Erweiterten Strukturierten Texts (ExST)“.

Die Anweisung verwenden Sie innerhalb von FOR-, WHILE-, und REPEAT-Schleifen, damit die Ausführung zum Anfang des nächsten Schleifendurchlaufs springt.

Beispiel:

```
FOR nCounter :=1 TO 5 BY 1 DO
    nInt1:=nInt1/2;
    IF nInt1=0 THEN
        CONTINUE; (* to avoid a division by zero *)
    END_IF
    nVar1:=nVar1/nInt1; (* executed, if nInt1 is not 0 *)
END_FOR;
nRes:=nVar1;
```

Siehe auch:

- ST-Anweisung FOR [▶ 663]
- ST-Anweisung WHILE [▶ 664]
- ST-Anweisung REPEAT [▶ 665]

16.1.3.5.10 ST-Aufruf Funktionsbaustein

Syntax:

<FB-instance>(<FB input variable>:=<value or adress>|, <other FB input variables>);

Beispiel:

```
fbTMR : TON;
fbTMR (IN := %OX5, PT := T#300ms);
bVarA := fbTMR.Q;
```

Der Timer-Funktionsbaustein TON wird in `fbTMR : TON;` instanziiert und mit Zuweisungen für die Parameter IN und PT aufgerufen.

Der Ausgang Q wird mit fbTMR.Q angesprochen und der Variablen bVarA zugewiesen.

Siehe auch:

- Objekt Funktionsbaustein [▶ 86]

16.1.3.5.11 ST-Kommentare

Kommentar	Beschreibung	Beispiel:
Einzeilig	Beginnt mit // und endet am Ende der Zeile.	`// Dies ist ein Kommentar`
Mehrzeilig	Beginnt mit (* und endet mit *).	`(* Dies ist ein mehrzeiliger Kommentar *)`
Verschachtelt	Beginnt mit (* und endet mit *). Innerhalb dieses Kommentars können weitere (*....*) Kommentare enthalten sein.	`(* a:=fbTest.out; (* 1.Kommentar *) b:=b+1; (* 2.Kommentar *)*)`

Eine oder mehrere markierte Zeilen können Sie zudem per Menübefehl oder per Tastaturkürzel ein-/auskommentieren:

- Befehl Auswahl auskommentieren [▶ 922]
- Befehl Auskommentierung der Auswahl aufheben [▶ 922]

Kommentare für Tooltips

Einen Tooltip für einen POU können Sie erzeugen, indem Sie einen Kommentar oberhalb der Deklaration des POU einfügen.

Beispiel:

```
// Das ist ein Beispiel-Funktionsbaustein
FUNCTION_BLOCK FB_Sample
VAR_INPUT
END_VAR
```

16.1.4 Ablaufsprache (AS)

16.1.4.1 AS-Editor

Der AS-Editor ist ein grafischer Editor. Eine neu hinzugefügte AS-POU enthält einen Init-Schritt und eine nachfolgende Transition.

Sie können im AS-Editor die einzelnen Elemente über Befehle des Menüs **AS**, des Kontextmenüs oder aus der Ansicht **Werkzeugkasten** in ein Diagramm einfügen.

Beim Einfügen über einen Menübefehl stehen die an der aktuell selektierten Position einfügbaren Elemente zur Auswahl.

Vor dem Einfügen von Verzweigungen parallel zu mehreren Aktionen und Transitionen, müssen Sie diese Aktionen und Transitionen durch Mehrfachauswahl selektieren.

Sie können AS-Elemente auch per Drag-and-drop aus der Ansicht **Werkzeugkasten** in das Diagramm ziehen. Sobald Sie ein Element in den Editor ziehen, markiert TwinCAT alle möglichen Einfügepositionen mit grauen Rechtecken. Wenn Sie die Maus über eine mögliche Einfügeposition bewegen, dann ändert sich die Farbe des Rechtecks auf grün. Wenn Sie die Maustaste nun loslassen, wird das Element an dieser Stelle eingefügt.

Wenn Sie eine Verzweigung per Drag-and-drop einfügen, müssen Sie den Beginn und das Ende der Verzweigung mit der Maus kennzeichnen. Den Beginn der Verzweigung markieren Sie durch Loslassen der Maus auf einer Einfügeposition. Dabei ändert sich die Farbe des Rechtecks auf rot. Durch Anklicken einer zweiten Einfügeposition legen Sie das Ende der Verzweigung fest. Danach fügt TwinCAT eine Verzweigung um die Objekte zwischen Beginn-Markierung und Ende-Markierung herum ein.

Für das Kopieren von Schritt- und Transitionselementen, die Aktionsobjekte oder Transitionsobjekte aufrufen, können zwei unterschiedliche Duplizierungsmodi eingestellt werden. Entweder werden die Referenzen mit kopiert, oder die referenzierten Objekte werden „eingebettet“ und beim Kopieren dupliziert.

Das Verhalten und Aussehen des Editors definieren Sie in den TwinCAT-Optionen in der Kategorie **TwinCAT > SPS Programmierumgebung > AS-Editor**.

Siehe auch:

- Generelle Funktionalitäten in allen grafischen Editoren [▶ 655]
- Programmieren in Ablaufsprache (AS) [▶ 129]
- Dokumentation TC3 User Interface: AS [▶ 1047]
- Dokumentation TC3 User Interface: Dialog Optionen - AS-Editor [▶ 1022]

16.1.4.2 AS-Editor im Onlinebetrieb

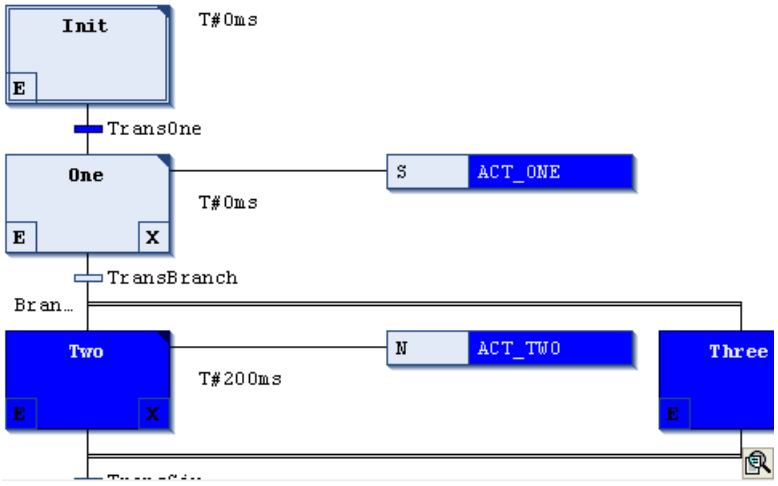
Im AS-Editor können Sie zur Laufzeit die verwendeten Variablen und Ausdrücke auf der Steuerung anzeigen. Sie können die Variablen und Ausdrücke auch schreiben und forcen. Eine Debugging-Funktionalität (Haltepunkte, schrittweise Ausführung etc.) ist noch nicht verfügbar.

Einstellungen bezüglich der Online-Darstellung der AS-Elemente und Attribute können Sie in den Optionen des AS-Editors vornehmen.

Falls Sie AS-Flags explizit deklariert haben, werden diese im Onlinebetrieb im Deklarationsteil angezeigt. Im Offlinebetrieb werden sie nicht angezeigt.

Beachten Sie die Abarbeitungsreihenfolge der Elemente in einem AS-Diagramm.

TwinCAT stellt aktive Schritte im Onlinebetrieb blau dar.



Siehe auch:

- Implizite Variablen [▶ 670]
- Abarbeitungsreihenfolge in AS [▶ 668]

16.1.4.3 Abarbeitungsreihenfolge in AS

Grundsätzliches Verhalten der Elemente

- Aktiver Schritt: Ein aktiver Schritt ist ein Schritt, dessen Aktionen gerade ausgeführt werden. Im Onlinebetrieb stellt TwinCAT aktive Schritte in blau dar.
- Initialschritt: Im ersten Zyklus nach Aufruf einer AS-POU wird der Initialschritt automatisch aktiv und die Schrittaktion wird ausgeführt.

- TwinCAT führt IEC-Aktionen mindestens zweimal aus. Das erste Mal nach Aktivwerden des Schrittes, das zweite Mal - aber erst im nächsten Zyklus - nach Deaktivieren des Schrittes.
- Alternative Verzweigungen: Wenn der Schritt vor der Verzweigung aktiv ist, wertet TwinCAT jeweils die erste Transition jedes alternativen Zweiges von links nach rechts aus. Im ersten Zweig, in dem TwinCAT eine Transition findet, die TRUE liefert, wird der darauf folgende Schritt aktiviert.
- Parallele Verzweigungen: Wenn der Schritt vor der Verzweigung (vor der horizontalen Doppellinie) aktiv ist und die Transition vor der Verzweigung TRUE liefert, aktiviert TwinCAT die ersten Schritte in allen Zweigen. Die Zweige werden dann gleichzeitig abgearbeitet. Der Schritt nach dem Ende der Verzweigung (nach der horizontalen Doppellinie) wird aktiviert, wenn alle letzten Schritte in den Zweigen aktiv sind und die Transition nach der Doppellinie TRUE liefert.

Abarbeitungsreihenfolge

1. **Reset der IEC-Aktionen**
   TwinCAT setzt die internen Aktionskontroll-Flags der Aktions-Qualifizierer (N, R, S, L, D, P, SD, DS, SL) zurück. Das sind die Flags, die eine IEC-Aktion steuern. Die Flags, die innerhalb von Aktionen aufgerufen werden, werden jedoch nicht zurückgesetzt!
2. **Ausgangsaktionen ausführen**
   TwinCAT überprüft daraufhin alle Schritte, ob sie die Bedingung für die Ausführung der Ausgangsaktionen erfüllen. Die Reihenfolge der Prüfung entspricht der Anordnung im AS-Diagramm: von oben nach unten und von links nach rechts.
   TwinCAT führt eine Ausgangsaktion aus, wenn der Schritt deaktiviert wird, d.h. wenn seine Eingangs- und Schrittaktionen (falls vorhanden) im vorangegangenen Zyklus ausgeführt worden sind und die Bedingung für den nachfolgenden Schritt TRUE liefert.
3. **Eingangsaktionen ausführen**
   TwinCAT überprüft daraufhin alle Schritte, ob sie die Bedingung für die Ausführung der Eingangsaktionen erfüllen. Die Reihenfolge der Prüfung entspricht der Anordnung im AS-Diagramm: von oben nach unten und von links nach rechts. Wenn dies der Fall ist, führt TwinCAT die Eingangsaktionen aus.
   TwinCAT führt eine Eingangsaktion aus, sobald die Abarbeitung bei der dem Schritt vorangehenden Transition angelangt ist und diese Transition TRUE liefert, der Schritt also aktiviert wird.
4. **Zeitenprüfung, Schrittaktionen ausführen**
   TwinCAT führt für alle Schritte in der Reihenfolge, in der sie im AS-Diagramm angeordnet sind, Folgendes durch:
   - TwinCAT kopiert die verstrichene Zeit des aktiven Schritts in die entsprechende implizite Schrittvariable <Schrittname>.t.
   - Bei einer Zeitüberschreitung bedient TwinCAT die entsprechenden Fehler-Flags.
   - Bei nicht-IEC-Schritten führt TwinCAT die Schrittaktion nun aus.
5. **IEC-Aktionen ausführen**
   TwinCAT führt die IEC-Aktionen in alphabetischer Reihenfolge aus. Dies erfolgt in zwei Durchläufen durch die Liste der Aktionen. Im ersten Lauf führt TwinCAT die IEC-Aktionen aller Schritte aus, die im vorangegangenen Zyklus deaktiviert wurden. Im zweiten Lauf werden die IEC-Aktionen aller Schritte ausgeführt, die aktiv sind.
6. **Transitionen-Check, Aktivierung der nachfolgenden Schritte**
   Die Transitionen werden ausgewertet: Wenn ein Schritt im aktuellen Zyklus aktiv ist und die nachfolgende Transition TRUE liefert (und die ggf. definierte Minimalzeit des Schrittes abgelaufen ist), wird der nachfolgende Schritt aktiviert.

Bei der Implementierung von Aktionen müssen Sie folgendes beachten:
Es kann vorkommen, dass eine Aktion innerhalb desselben Zyklus mehrfach ausgeführt wird, weil sie mit mehreren Abläufen verbunden ist. (Beispiel: Ein AS enthält die zwei IEC-Aktionen A und B, die beide in AS programmiert sind und die beide eine IEC-Aktion C aufrufen. Dann können A und B beide in IEC-Aktionen im gleichen Zyklus aktiv sein und darüber hinaus kann IEC-Aktion C in beiden Aktionen aktiv sein. In diesem Fall würde C zweimal aufgerufen werden.)
Wenn die gleiche IEC-Aktion gleichzeitig auf verschiedenen Ebenen eines AS-Diagramms verwendet wird, kann das zu unvorhersehbaren Effekten während der Abarbeitung führen. Deshalb wird eine entsprechende Fehlermeldung ausgegeben. Diese wird möglicherweise erscheinen, wenn mit Projekten gearbeitet wird, die mit einer älteren Version des Programmiersystems erstellt wurden.

Beachten Sie die Möglichkeit, Implizite Variablen zu verwenden, um den Abarbeitungsstatus von Schritten und Aktionen zu überwachen bzw. die Abarbeitung zu kontrollieren.

Siehe auch:

- Implizite Variablen [▶ 670]
- Qualifizierer für Aktionen in AS [▶ 670]

16.1.4.4 Qualifizierer für Aktionen in AS

Sie weisen Qualifizierer IEC-Schritten zu. Qualifizierer beschreiben, in welcher Art eine zum Schritt gehörige Aktion ausgeführt wird.

Die Qualifizierer werden vom Funktionsbaustein `SFCActionControl` der Systembibliothek verarbeitet. Die Bibliothek wird automatisch durch die SFC-Plug-Ins im Projekt eingebunden.

Verfügbare Qualifizierer:

N	Non-stored	Die Aktion ist so lange aktiv wie der Schritt.
R	overriding Reset	Die Aktion wird deaktiviert.
S	Set (Stored)	TwinCAT führt die Aktion aus, sobald der Schritt aktiv wird. Die Aktion wird weiter ausgeführt, auch wenn der Schritt schon deaktiviert wurde, bis sie einen Reset erhält.
L	time Limited	TwinCAT führt die Aktion aus, sobald der Schritt aktiv wird. Die Aktion wird so lange ausgeführt, bis der Schritt inaktiv wird oder die gegebene Zeitspanne abgelaufen ist.
D	time Delayed	TwinCAT startet die Ausführung der Aktion erst, wenn nach aktiv werden des Schrittes die gegebene Verzögerungszeit abgelaufen ist und der Schritt immer noch aktiv ist. Die Aktion wird ausgeführt, bis der Schritt deaktiviert wird.
P	Pulse	TwinCAT führt die Aktion genau zweimal aus: Einmal, wenn der Schritt aktiv wird und ein weiteres Mal im darauffolgenden Zyklus.
SD	Stored and time Delayed	TwinCAT startet die Ausführung der Aktion erst, wenn nach aktiv werden des Schrittes die gegebene Verzögerungszeit abgelaufen ist. Die Aktion wird so lange ausgeführt, bis sie einen Reset erhält.
DS	Delayed and Stored	TwinCAT startet die Ausführung der Aktion erst, wenn nach aktiv werden des Schrittes die gegebene Verzögerungszeit abgelaufen ist und der Schritt immer noch aktiv ist. Die Aktion wird so lange ausgeführt, bis sie einen Reset erhält.
SL	Stored and time limited	TwinCAT führt die Aktion aus, sobald der Schritt aktiviert wird. Sie wird so lange ausgeführt, bis die gegebene Zeit abgelaufen ist oder sie einen Reset erhält.

Sie müssen die Zeitangaben bei den Qualifizierern L, D, SD, DS und SL im Format einer TIME-Konstante angeben.

Wenn eine IEC-Aktion deaktiviert wird, wird sie ein weiteres Mal ausgeführt. Dies bedeutet, dass TwinCAT eine solche Aktion mindestens zweimal ausführt. Das betrifft auch Aktionen mit dem Qualifizierer P.

Siehe auch:

- Programmieren in Ablaufsprache (AS) [▶ 129]

16.1.4.5 Implizite Variablen

Jedes AS-Objekt stellt implizite Variablen bereit, mit denen Sie zur Laufzeit den Status von Schritten und IEC-Aktionen überwachen können. Diese impliziten Variablen werden von TwinCAT für jeden Schritt und jede IEC-Aktion automatisch angelegt.

Die impliziten Variablen sind Strukturinstanzen vom Typ SFCStepType (bei Schritten) bzw. SFCActionType (bei Aktionen). Die Variablen haben den Namen des Elementes, z. B. „step1“ für einen Schritt mit Schrittnamen „step1“. Die Strukturkomponenten beschreiben den Status eines Schritts bzw. einer Aktion oder die aktuell bereits abgelaufene Zeit in einem aktiven Schritt.

Schritt- und Aktionsstatus

Die Syntax für die implizit durchgeführte Variablendeklaration:

<stepname>:SFCStepType;

_<actionname>:SFCActionType;

Folgende implizite Variablen stehen Ihnen für Schritt- oder IEC-Aktionsstatus zur Verfügung:

Schritt	
<Schrittname>.x	Zeigt den Aktivationsstatus im aktuellen Zyklus. Wenn <Schrittname>.x = TRUE, führt TwinCAT den Schritt im aktuellen Zyklus aus.
<Schrittname>._x	Zeigt den Aktivationsstatus für den nächsten Zyklus. Wenn <Schrittname>._x = TRUE und <Schrittname>.x = FALSE, führt TwinCAT den Schritt im nächsten Zyklus aus, d.h. <Schrittname>._x wird zu Beginn eines Zyklus in <Schrittname>.x kopiert.
<Schrittname>.t	Das Flag t liefert die aktuelle Zeitspanne, die seit Aktivwerden des Schrittes verstrichen ist. Dies gilt nur für Schritte, unabhängig davon ob in den Schritteigenschaften eine Minimalzeit definiert ist oder nicht. Siehe auch AS-Flag SFCError.
<Schrittname>._t	Verwendung nur für interne Zwecke
IEC-Aktion	
_<Aktionsname>.x	TRUE, wenn die Aktion ausgeführt wird.
_<Aktionsname>._x	TRUE, wenn die Aktion aktiv ist.

Sie können die oben beschriebenen Variablen benutzten, um einen bestimmten Statuswert für einen Schritt zu erzwingen (forcen), d.h. um einen Schritt aktiv zu setzen. Beachten Sie jedoch, dass dadurch ein unkontrollierter Status des AS herbeigeführt wird.

Zugriff auf implizite Variablen

Syntax für den Zugriff:

Innerhalb der POU weisen Sie die implizite Variable direkt zu: <Variablenname>:=<Schrittname>.<implizite Variable> bzw. <Variablenname>:=_<Aktionsname>.<implizite Variable>

Beispiel:

```
status := step1._x;
```

Von einem anderen Baustein aus mit POU-Name: <Variablenname>:=<POU-Name>.<Schrittname>.<implizite Variable> bzw. <Variablenname>:=<POU-Name>._<Aktionsname>.<implizite Variable>

Beispiel:

```
status := SFC_prog.step1._x;
```

Siehe auch:

- AS-Elementeigenschaften [▶ 683]
- AS-Flags [▶ 672]

16.1.4.6 AS-Flags

AS-Flags sind implizit erzeugte Variablen mit vordefinierten Namen. Sie können sie verwenden, um die Abarbeitung eines AS-Diagramms zu beeinflussen. Mit diesen Flags können Sie zum Beispiel Zeitüberschreitungen anzeigen oder Schrittketten zurücksetzen. Weiterhin können Sie zum Beispiel den Tipp-Betrieb anschalten, um Transitionen gezielt zu schalten. Damit Sie auf diese Variablen Zugriff haben, müssen Sie sie deklarieren und aktivieren.

AS-Flags

Name	Datentyp	Beschreibung
SFCInit	Bool	TRUE: TwinCAT setzt den Ablauf auf den Initialschritt zurück. Die anderen AS-Flags werden ebenfalls zurückgesetzt (Initialisierung). Solange die Variable TRUE ist, bleibt der Initialschritt gesetzt (aktiv), wird aber nicht ausgeführt. Erst wenn Sie `SFCInit` wieder auf FALSE setzen, wird der Baustein normal weiterbearbeitet.
SFCReset	Bool	Verhält sich ähnlich wie `SFCInit`. Im Unterschied zu diesem arbeitet TwinCAT allerdings nach der Initialisierung den Initialschritt weiter ab. So könnten Sie beispielsweise im Init-Schritt das SFCReset-yyyyy-Flag gleich wieder auf FALSE setzen.
SFCError	Bool	Wird TRUE, wenn in einem AS-Diagramm eine Zeitüberschreitung aufgetreten ist. Wenn im Programm nach der ersten Zeitüberschreitung eine weitere auftritt, wird diese nicht mehr registriert, wenn Sie die Variable `SFCError` vorher nicht wieder zurückgesetzt haben. Die Deklaration von `SFCError` ist Voraussetzung für das Funktionieren der anderen Flag-Variablen zur Kontrolle des zeitlichen Ablaufs (`SFCErrorStep`, `SFCErrorPOU`, `SFCQuitError`).
SFCEnableLimit	Bool	Dient dem gezielten Einschalten (TRUE) und Ausschalten (FALSE) der Kontrolle für Zeitüberschreitungen in Schritten durch `SFCError`. Wenn Sie diese Variable deklarieren und aktivieren (AS-Einstellungen), müssen Sie sie auch auf TRUE setzen, damit `SFCError` arbeitet, ansonsten werden Zeitüberschreitungen ignoriert. Die Verwendung kann beispielsweise bei Inbetriebnahme oder Handbetrieb nützlich sein. Wenn Sie die Variable nicht deklarieren, arbeitet `SFCError` automatisch. Voraussetzung ist die Deklaration von `SFCError`.
SFCErrorStep	String	Speichert den Namen des Schritts, der eine durch `SFCError` registrierte Zeitüberschreitung verursacht hat. Der Name wird gespeichert, bis die registrierte Zeitüberschreitung durch `SFCQuitError` zurückgesetzt wird. Voraussetzung ist die Deklaration von `SFCError`.
SFCErrorPOU	String	Speichert im Falle einer Zeitüberschreitung den Namen des Bausteins, in dem eine durch `SFCError` registrierte Zeitüberschreitung aufgetreten ist. Der Name wird gespeichert, bis die registrierte Zeitüberschreitung durch `SFCQuitError` zurückgesetzt wird. Voraussetzung ist die Deklaration von `SFCError`.
SFCQuitError	Bool	Solange diese boolesche Variable TRUE ist, hält TwinCAT die Abarbeitung des AS-Diagramms an. Eine eventuelle Zeitüberschreitung, gespeichert in der Variablen `SFCError`, wird dabei zurückgesetzt. Wenn Sie die Variable wieder auf FALSE setzen, werden alle bisherigen Zeiten in den aktiven Schritten zurückgesetzt. Voraussetzung ist die Deklaration von `SFCError`.
SFCPause	Bool	Solange diese Variable TRUE ist, hält TwinCAT die Abarbeitung des AS-Diagramms an.
SFCTrans	Bool	Wird TRUE, wenn eine Transition schaltet.
SFCCurrentStep	String	Zeigt den Namen des gerade aktiven Schritts, unabhängig von der Zeitüberwachung. Bei einer Parallel-Verzweigung wird immer der Name des Schrittes des äußersten rechten Zweigs gespeichert.
SFCTip, SFCTipMode	Bool	Erlauben den „Tipp-Betrieb“ des AS-Bausteins. Wenn Sie den Tipp-Betrieb durch SFCTipMode=TRUE aktivieren, kann nur zum nächsten Schritt weiter geschaltet werden, indem Sie `SFCTip` auf TRUE setzen. Solange Sie `SFCTipMode` auf FALSE setzen, kann zusätzlich auch über die Transitionen weiter geschaltet werden.

Name	Datentyp	Beschreibung
SFCErrorAnalyzation		Enthält als Zeichenkette alle Variablen, die zum Gesamtwert TRUE von `SFCError` beitragen (Zeitüberschreitung in einem Schritt). `SFCError` muss dazu aktiviert sein. `SFCErrorAnalyzation` verwendet dazu implizit die Funktion des Bausteins AnaylzeExpression der Bibliothek Tc2_System.
SFCError AnalyzationTable		Enthält in einer Tabelle die Variablen, die zum Gesamtwert TRUE von `SFCError` beitragen (Zeitüberschreitung in einem Schritt). `SFCError` muss dazu aktiviert sein. `SFCErrorAnalyzationTable` verwendet dazu implizit die Funktion des Bausteins AnaylzeExpressionTable der Bibliothek Tc2_System.

Implizite Erzeugung der AS-Flags

AS-Flags werden von TwinCAT automatisch deklariert, wenn Sie die entsprechende Option aktiviert haben. Diese Option können Sie in der Ansicht **Eigenschaften** für eine einzelne AS-POU oder in den Projekteigenschaften in der Kategorie **AS** für alle AS-POUs im Projekt setzen.

Die AS-Einstellungen der AS-Flags von einzelnen POUs sind nur wirksam, wenn Sie die Option **Use default SFC settings** nicht aktiviert haben. Wenn Sie diese Option aktiviert haben, dann gelten die Einstellungen, die in den Projekteigenschaften definiert sind.

AS-Flags, die Sie im AS-Einstellungen-Dialog deklariert haben, sind nur in der Online-Ansicht des AS-Bausteins sichtbar!

Explizite Erzeugung der AS-Flags

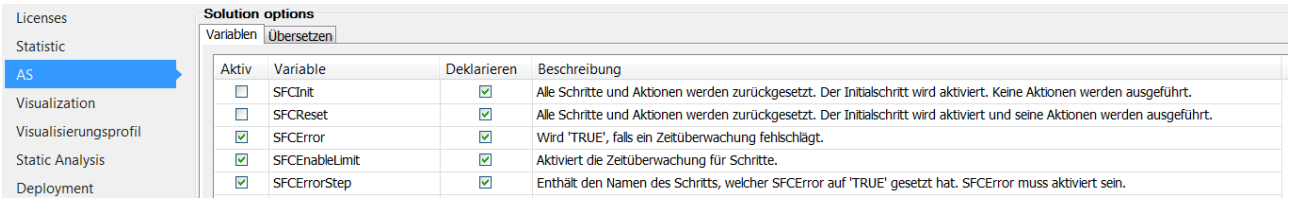
Händische Deklaration ist nur noch erforderlich, um Schreibzugriff von einem anderen Baustein aus zu ermöglichen. In diesem Fall müssen Sie folgendes beachten: Wenn Sie das Flag in einer globalen Variablenliste deklariert haben, müssen Sie dessen Einstellung „Deklarieren“ im AS-Einstellungen-Dialog deaktivieren. Andernfalls wird implizit ein lokales AS-Flag deklariert, das TwinCAT dann anstelle der globalen Variablen verwendet!

Anwendungsbeispiel für SFCError

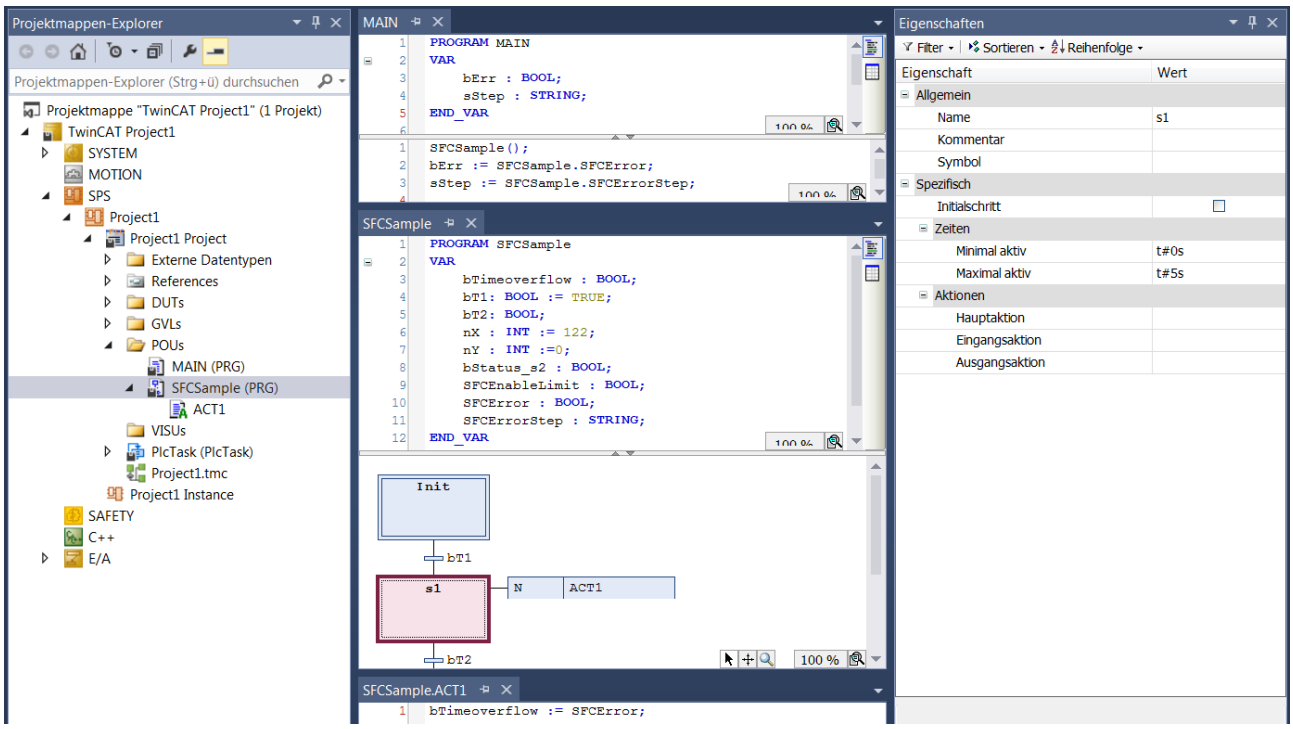
Sie haben einen AS-Baustein namens „SFCSample“ angelegt, der einen Schritt „s1“ enthält. In den Eigenschaften des Schritts haben Sie Zeitbegrenzungen definiert. Siehe Abbildung „Online-Ansicht von AS-Baustein SFCSample“.

Wenn aus irgendwelchen Gründen Schritt s1 länger aktiv bleibt als in seinen Zeiteigenschaften erlaubt ist (Zeitüberschreitung), setzt TwinCAT das AS-Flag `SFCError`, auf das das SPS-Programm zugreifen kann.

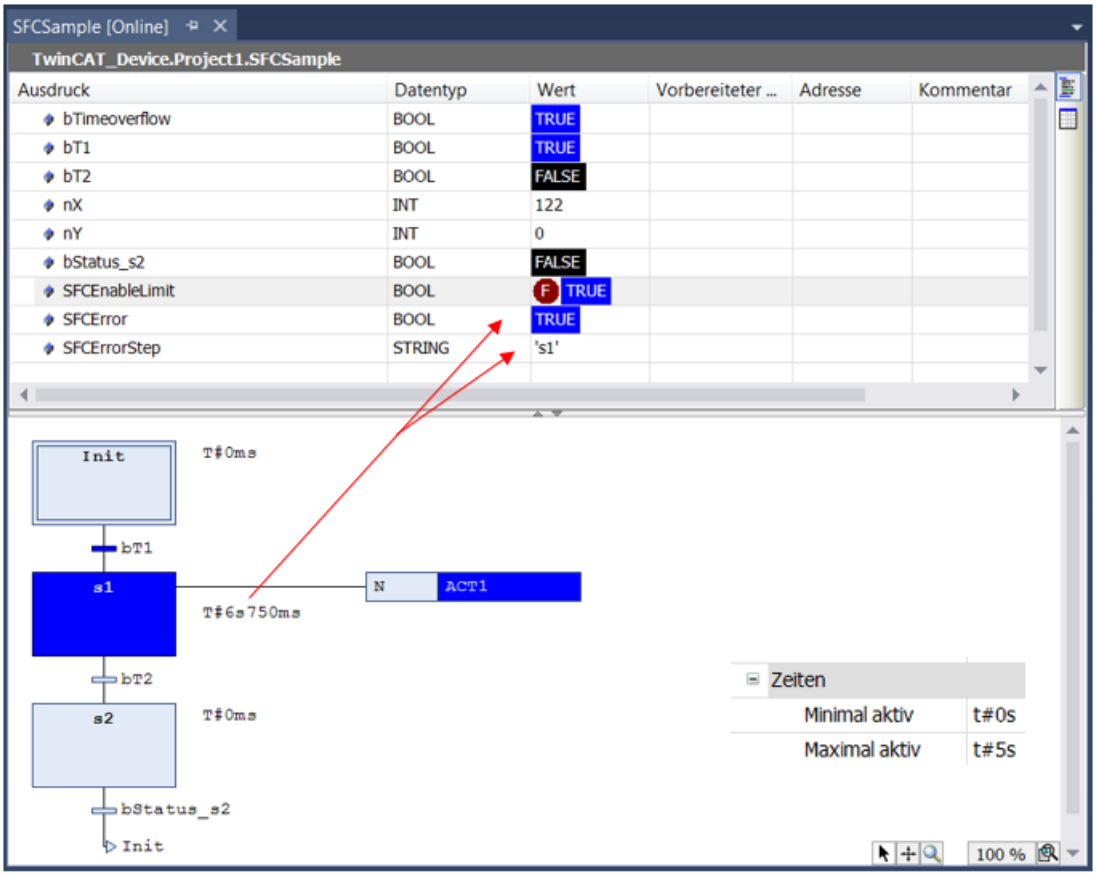
Um diesen Zugriff zu erlauben, müssen Sie das AS-Flag in den AS-Einstellungen aktivieren und deklarieren. Wenn Sie es nur deklarieren, wird das AS-Flag zwar in der Online-Ansicht von `SFCSample` im Deklarationsteil angezeigt, ist aber ohne Funktion.



Nun können Sie das AS-Flag innerhalb des Bausteins, beispielsweise in einer Aktion, oder von außerhalb des Bausteins ansprechen.



Online-Ansicht von AS-Baustein `SFCSample`:



`SFCError` wird TRUE, sobald eine Zeitüberschreitung innerhalb von `SFCSample` auftritt.

Beachten Sie die Möglichkeit mit Hilfe der Flags `SFCErrorAnalyzation` und `SFCErrorAnalyzationTable` die Komponenten des Ausdrucks zu ermitteln, der zum Wert TRUE des `SFCError` beiträgt.

Zugriff auf die Flags

Syntax für den Zugriff:

Innerhalb der POU weisen Sie das Flag direkt zu: <Variablenname>:=<AS-Flag>

Beispiel:

```
checkerror:=SFCerror;
```

Von einem anderen Baustein aus mit POU-Name: <Variablenname>:=<POU-Name>.<AS-Flag>

Beispiel:

```
checkerror:=SFC_prog.SFCerror;
```

Falls Sie einen Schreibzugriff von einem anderen Baustein aus benötigen, müssen Sie das AS-Flag zusätzlich explizit als eine VAR_INPUT Variable im AS-Baustein oder global in einer GVL deklarieren.

Beispiel:

Lokale Deklaration:

```
PROGRAM SFC_prog
VAR_INPUT
    SFCinit:BOOL;
END_VAR
```

oder globale Deklaration in einer globalen Variablenliste:

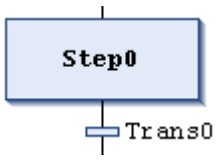
```
VAR_GLOBAL
    SFCinit:BOOL;
END_VAR

PROGRAM PLC_PRG
VAR
    setinit: BOOL;
END_VAR
SFC_prog.SFCinit:=setinit;  //Schreibzugriff auf SFCinit in SFC_prog
```

16.1.4.7 Elemente

16.1.4.7.1 AS-Elemente Schritt und Transition

Symbol Schritt:  , Symbol Transition: 

TwinCAT fügt Schritte und Transitionen grundsätzlich als Kombination ein. Wenn Sie einen Schritt ohne Transition oder eine Transition ohne Schritt einfügen, führt dies beim Übersetzen zu einem Fehler. Durch einen Doppelklick auf den Namen können Sie diesen verändern.



Schrittnamen müssen innerhalb des Gültigkeitsbereichs des „Vater“-Bausteins eindeutig sein. Bedenken Sie dies insbesondere, wenn Aktionen verwendet werden, die ebenfalls in AS programmiert sind.

Beachten Sie, dass Sie einen Schritt mit dem Befehl **Initialschritt** oder durch Setzen der entsprechenden Eigenschaft in den AS-Elementeigenschaften zum Initialschritt machen können.

Jeder Schritt wird durch die Schritteigenschaften definiert, die Sie abhängig von den eingestellten Optionen in der Ansicht **Eigenschaften** anzeigen und bearbeiten können.

Aktionen, die ausgeführt werden sollen, wenn der Schritt aktiv ist, müssen Sie dem Schritt hinzufügen. Es gibt „IEC-Aktionen“ und „Schrittaktionen“. Details hierzu finden Sie im Abschnitt zum AS-Element **Aktion**.

Es liegt in der Verantwortung des Benutzers, den gewünschten Ausdruck einer Transitionsvariable zuzuweisen, falls die Transition mehrfache Anweisungen enthält.

Transitionen, die aus einem Transitions- oder Eigenschaften-Objekt bestehen, werden durch ein kleines Dreieck in der rechten oberen Ecke des Transitionsrechtecks gekennzeichnet.

Siehe auch:

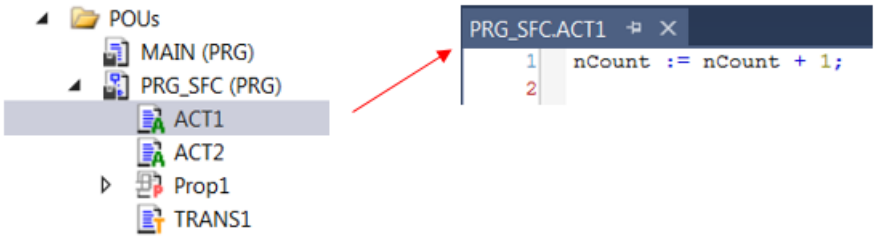
- Programmieren in Ablaufsprache (AS) [▶ 129]
- Objekt Transition [▶ 90]
- AS-Elementeigenschaften [▶ 683]
- AS-Element Aktion [▶ 678]
- Methodenaufruf - virtueller Funktionsaufruf [▶ 210]
- Dokumentation TC3 User Interface: Befehl Schritt-Transition einfügen [▶ 1047]
- Dokumentation TC3 User Interface: Befehl Initialschritt [▶ 1047]

16.1.4.7.2 AS-Element Aktion

Symbol:

Eine Aktion enthält eine oder mehrere Anweisungen in einer der gültigen Programmiersprachen. Sie können eine Aktion einem Schritt zuweisen.

Aktionen, die Sie in AS-Schritten verwenden, müssen als Bausteine im Projekt angelegt sein.



Ausnahme: Im Fall von IEC-Aktionen, die Sie als Aktionsassoziation einem Schritt hinzufügen, können Sie anstelle eines Aktionsobjekts auch eine boolsche Variable angeben. Der Wert dieser Variablen wird bei jeder Ausführung der Aktion zwischen FALSE und TRUE umgeschaltet.

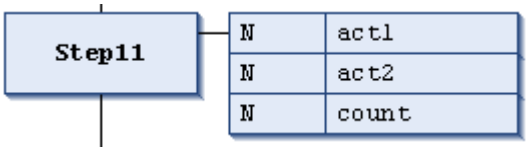
Sie müssen Schrittnamen innerhalb des Gültigkeitsbereichs des „Vater“-Bausteins eindeutig definieren. Eine in AS geschriebene Aktion darf keinen Schritt enthalten, der den gleichen Namen hat wie der Schritt, dem die Aktion zugewiesen ist.

Es gibt „IEC-Aktionen“ und „Schrittaktionen“:

1. IEC-Aktionen

IEC-Aktionen sind Aktionen gemäß der Norm IEC 61131-3. Sie werden entsprechend ihres Qualifizierers ausgeführt. IEC-Aktionen werden mindestens zweimal ausgeführt: das erste Mal, wenn der Schritt aktiv wird und ein zweites Mal, wenn er deaktiviert wird. Wenn Sie einem Schritt mehrere Aktionen zuweisen, wird die Aktionenliste von oben nach unten abgearbeitet.

Jede Aktionenbox enthält in der ersten Spalte den Qualifizierer und in der zweiten den Aktionsnamen. Beide können direkt editiert werden.



Sie können für IEC-Aktionen im Gegensatz zu Schrittaktionen unterschiedliche Qualifizierer verwenden. Ein weiterer Unterschied zu den Schrittaktionen ist, dass jede IEC-Aktion mit einem Kontroll-Flag versehen ist. Dies bewirkt, dass TwinCAT die Aktion - auch wenn sie gleichzeitig von einem weiteren Schritt aufgerufen wird - nur einmal zur selben Zeit ausführt. Dies kann für Schrittaktionen nicht garantiert werden.

Sie weisen einem Schritt IEC-Aktionen mit dem Befehl **Aktionsassoziation einfügen** im Menü **SFC** zu.

Assoziierte boolesche Variablen

Eine assoziierte boolesche Variable wird bei jedem Aufruf des AS-Bausteins gesetzt oder zurückgesetzt. Das heißt, ihr wird jedes Mal entweder der Wert TRUE oder FALSE neu zugewiesen, unabhängig davon, ob der zugehörige Schritt aktiv ist oder nicht.

Wenn in verschiedenen AS-Bausteinen dieselbe globale boolsche Variable als IEC-Aktion assoziiert ist, kann dies zu nicht erwünschten Überschreibungseffekten führen.

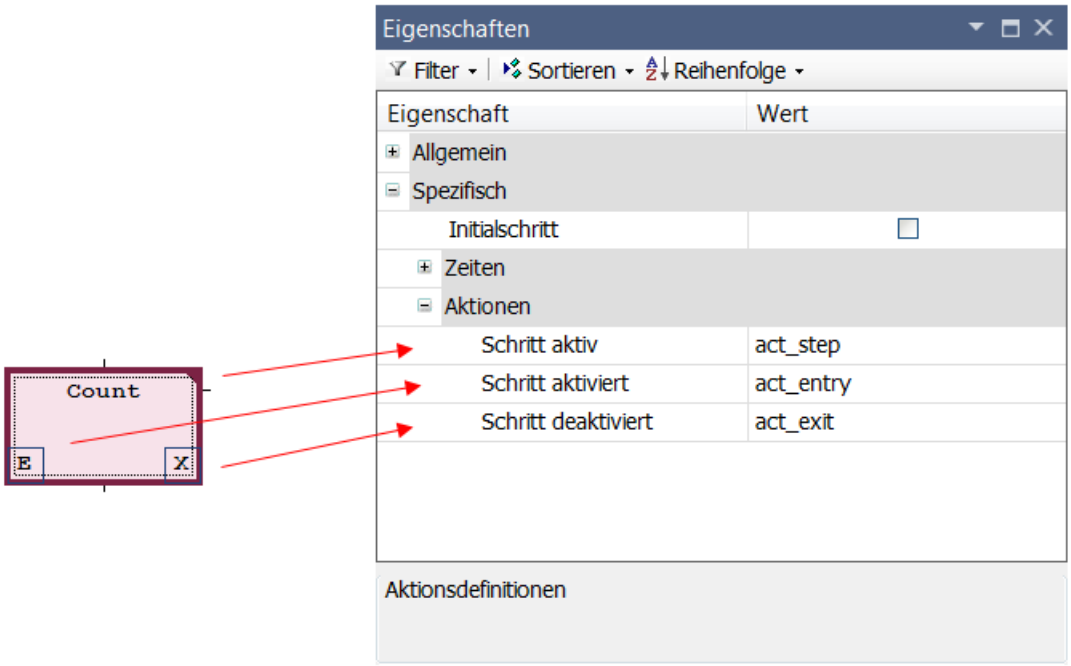
Siehe auch:

- Dokumentation TC3 User Interface: Befehl Aktionsassoziation einfügen [▶ 1051]
- Qualifizierer für Aktionen in AS [▶ 670]

2. Schrittaktionen

Dies sind Aktionen, die Sie in Erweiterung des IEC-Standards verwenden können.

- **Eingangsaktion**:
  TwinCAT führt diese Aktion einmal aus, nachdem der Schritt aktiviert wurde und bevor die Hauptaktion ausgeführt wird.
  Sie referenzieren eine neue oder bereits unter dem AS-Objekt angelegte Aktion aus einem Schritt heraus über die Elementeigenschaft **Eingangsaktion** (**Step entry**). Eine neue Aktion können Sie dem Schritt auch mit dem Befehl **Eingangsaktion hinzufügen** hinzufügen. Die Eingangsaktion wird durch ein „E“ in der unteren linken Ecke der Schritt-Box angezeigt.
- **Hauptaktion**:
  TwinCAT führt diese Aktion aus, nachdem der Schritt aktiviert wurde und eine eventuelle Eingangsaktion bereits ausgeführt wurde. Allerdings wird sie im Gegensatz zu einer IEC-Aktion (s. o.) kein zweites Mal ausgeführt, wenn der Schritt wieder deaktiviert wird. Sie können hier auch keine Qualifizierer verwenden.
  Eine bereits vorliegende Aktion fügen Sie einem Schritt über die Elementeigenschaft **Hauptaktion** (**Step active**) hinzu. Eine Hauptaktion wird durch ein gefülltes Dreieck in der rechten oberen Ecke der Schritt-Box angezeigt.
- **Ausgangsaktion**:
  TwinCAT führt diese Aktion einmal aus, wenn der Schritt deaktiviert wurde. Beachten Sie allerdings, dass die Ausführung nicht mehr im gleichen Zyklus, sondern zu Beginn des nächsten erfolgt!
  Sie referenzieren eine neue oder bereits unter dem AS-Objekt angelegte Aktion aus einem Schritt heraus über die Elementeigenschaft **Ausgangsaktion** (**Step exit**). Eine neue Aktion können Sie dem Schritt auch mit dem Befehl **Ausgangsaktion hinzufügen** hinzufügen. Die Ausgangsaktion wird durch ein „X“ in der unteren rechten Ecke der Schritt-Box angezeigt.



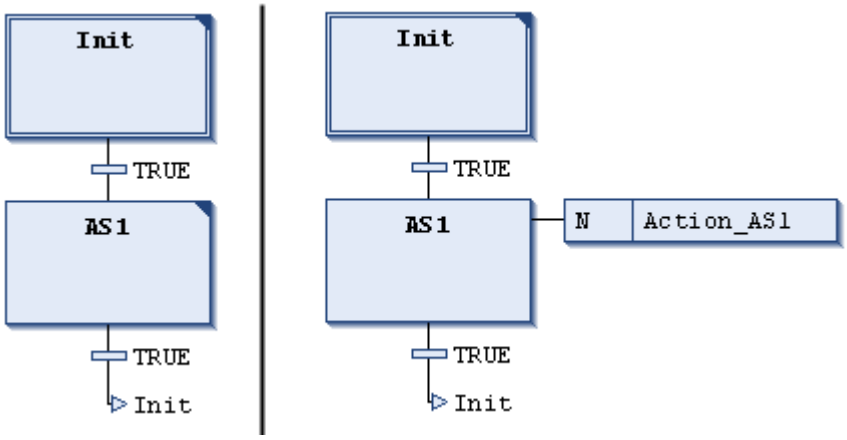
Siehe auch:

- AS-Elementeigenschaften [▶ 683]

Unterschied IEC-Aktion - Schrittaktion

Ausführung

Der wesentliche Unterschied zwischen Schrittaktionen und IEC-Aktionen mit Qualifizierer „N“ besteht darin, dass die IEC-Aktion (mindestens) zweimal ausgeführt wird: ein erstes Mal, wenn der Schritt aktiv ist und ein zweites Mal, wenn er deaktiviert wird. Siehe dazu folgendes Beispiel:



Sie haben dem Schritt AS1 die Aktion Action_AS1 einmal als Schrittaktion (links), einmal als IEC-Aktion mit Qualifizierer N beigefügt. Da in beiden Fällen zwei Transitionen geschaltet werden, dauert es jeweils zwei SPS-Zyklen, bis erneut der Initialisierungsschritt erreicht wird. Angenommen, in der Aktion Action_AS1 werde eine mit 0 initialisierte Zählvariable nCounter hochgezählt. Nach der erneuten Aktivierung des Schritts Init hat nCounter im linken Beispiel den Wert 1. Im rechten Beispiel hat sie jedoch den Wert 2, da die IEC-Aktion ein zweites Mal aufgrund der Deaktivierung von AS1 ausgeführt wird.

Duplizierung

Ein weiterer Unterschied ist, dass Schrittaktionen „eingebettet“ werden können. In diesem Fall können sie nur noch von dem betreffenden Schritt aufgerufen werden. Wenn Sie diesen Schritt kopieren, erzeugt TwinCAT automatisch neue Aktionsobjekte und kopiert jeweils den Implementierungscode.

Ob eine Schrittaktion „eingebettet“ wird, definieren Sie entweder beim Einfügen der ersten Aktion im Schritt oder über die Schritteigenschaft **Duplizieren oder kopieren.** Generell kann dieses Verhalten auch in den TwinCAT-Optionen in der Kategorie **AS-Editor** voreingestellt werden.

Boolsche Variable

Außerdem kann bei IEC-Aktionen anstelle eines Aktionsobjekts eine boolsche Variable angegeben werden. Dies ist bei Schrittaktionen nicht möglich.

Siehe auch:

- AS-Elementeigenschaften [▶ 683]

16.1.4.7.3 AS-Element Verzweigung

Symbol: 

Sie verwenden Verzweigungen, wenn Sie in der Ablaufsprache parallele oder alternative Abläufe programmieren.

Bei alternativen Verzweigungen führt TwinCAT immer nur einen der Zweige aus, abhängig von der vorausgehenden Transitionsbedingung. Parallele Verzweigungen werden gleichzeitig ausgeführt.

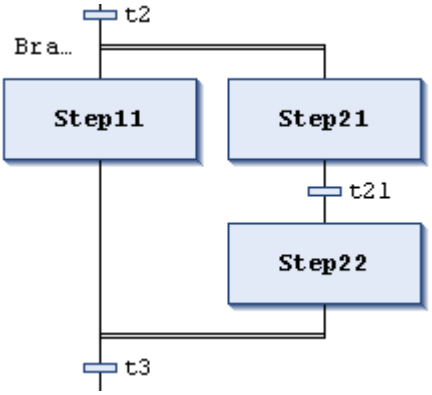
Siehe auch:

- Programmieren in Ablaufsprache (AS) [▶ 129]
- Abarbeitungsreihenfolge in AS [▶ 668]
- Dokumentation TC3 User Interface: Befehl Verzweigung rechts einfügen [▶ 1049]

Parallel Verzweigung

In einer parallelen Verzweigung müssen die Zweige mit Schritten beginnen und enden. Parallele Zweige können weitere Verzweigungen beinhalten.

Die horizontale Linie vor und nach der Verzweigung ist eine doppelte Linie.



Abarbeitung im Onlinebetrieb: Wenn die vorausgehende Transition (t2 im dargestellten Beispiel) TRUE ist, werden die ersten Schritte in allen parallelen Zweigen aktiv (Step11 und Step21). TwinCAT arbeitet die einzelnen Zweige parallel ab und wertet erst dann die nachfolgende Transition aus (t3).

Der horizontalen Linie, die den Beginn einer Verzweigung bildet, wird automatisch eine Sprungmarke Branch<n> hinzugefügt. Sie können diese Marke als Sprungziel angeben.

Beachten Sie, dass Sie eine parallele Verzweigung mit dem Befehl **Alternativ** in eine alternative Verzweigung umwandeln können.

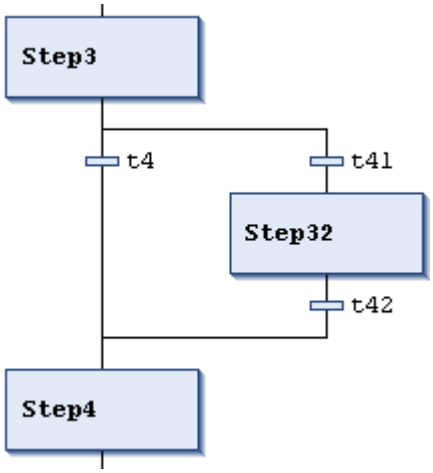
Siehe auch:

- Dokumentation TC3 User Interface: Befehl Alternativ [▶ 1049]

Alternative Verzweigung

Die horizontale Linie vor und nach der Verzweigung ist eine einfache Linie.

In einer alternativen Verzweigung müssen die Zweige mit Transitionen beginnen und enden. Die Zweige können weitere Verzweigungen beinhalten.



Wenn der Schritt, der der Verzweigung vorangeht, aktiv ist, wertet TwinCAT jeweils die erste Transition der alternativen Zweige von links nach rechts aus. Bei der ersten Transition, die TRUE liefert, wird der zugehörige Zweig „geöffnet“, d. h. der Transition folgende Schritt aktiviert.

Beachten Sie, dass Sie eine alternative Verzweigungen mit dem Befehl **Parallel** in eine parallele Verzweigung umwandeln können.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Parallel [▶ 1048]

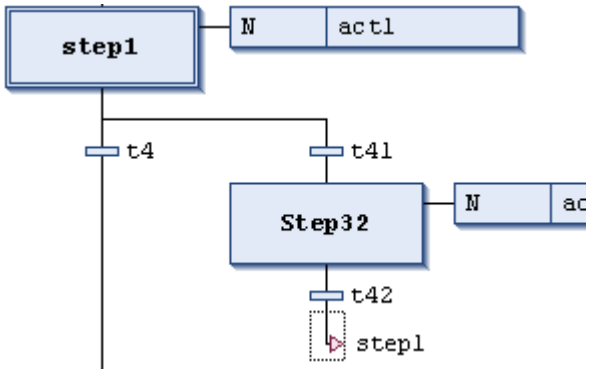
16.1.4.7.4 AS-Element Sprung

Symbol: 

Ein Sprung definiert, welcher Schritt als nächstes ausgeführt werden soll, sobald die dem Sprung vorausgehende Transition TRUE wird. Sprünge können nötig sein, weil die Ausführungslinien sich nicht kreuzen und nicht nach oben führen können.

Neben dem obligatorischen Sprung am Ende des Diagramms können Sie Sprünge nur am Ende einer Verzweigung eingeben.

Das Ziel eines Sprungs wird durch den hinzugefügten Textstring definiert, den Sie direkt editieren können. Das Sprungziel kann ein Schrittname oder die Marke einer parallelen Verzweigung sein.



Siehe auch:

- Programmieren in Ablaufsprache (AS) [▶ 129]
- Dokumentation TC3 User Interface: Befehl Sprung einfügen [▶ 1051]

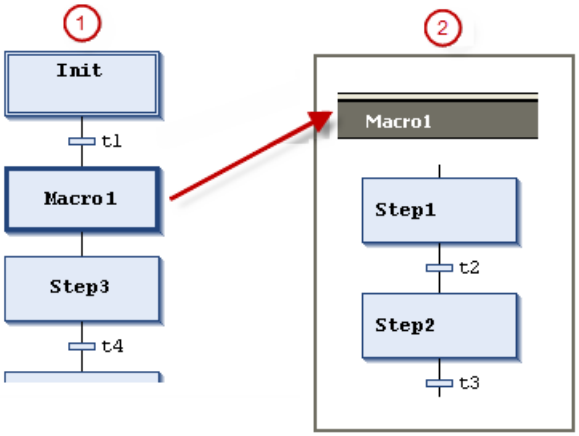
16.1.4.7.5 AS-Element Makro

Symbol: 

Ein Makro enthält einen Teil des AS-Diagramms, der in der Hauptansicht des Editors nicht ausführlich dargestellt wird.

Der Abarbeitungsfluss wird durch die Verwendung von Makros nicht beeinflusst. Makros dienen nur dazu, bestimmte Teile des Diagramms auszublenden um zum Beispiel die Übersichtlichkeit zu erhöhen.

Sie öffnen den Makro-Editor mit einem Doppelklick auf die Makro-Box oder mit dem Befehl **Makro anzeigen** im Menü **AS**. Sie können hier genauso programmieren wie in der Hauptansicht des AS-Editors. Um den Makro-Editor wieder zu schließen, verwenden Sie Befehl **Makro verlassen** im Menü **AS**.

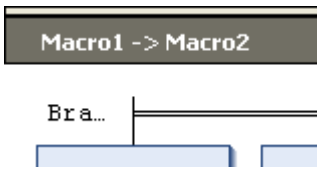


① Hauptansicht im SFC-Editor

② Ansicht im Makro-Editor für Macro1

Makros können wiederum Makros enthalten. Die Titelzeile des Makro-Editors zeigt immer, welchen Pfad das gerade geöffnete Makro innerhalb des Diagramms hat.

Beispiel:



Siehe auch:

- Programmieren in Ablaufsprache (AS) [▶ 129]
- Dokumentation TC3 User Interface: Befehl Makro anzeigen [▶ 1053]
- Dokumentation TC3 User Interface: Befehl Makro verlassen [▶ 1053]

16.1.4.7.6 AS-Elementeigenschaften

Sie bearbeiten die Eigenschaften eines AS-Elements in der Ansicht **Eigenschaften**. Sie öffnen diese Ansicht mit dem Befehl **Ansicht > Eigenschaftenfenster**. Es hängt vom gerade ausgewählten Element ab, welche Eigenschaften dargestellt werden.

Welche Eigenschaften neben dem Element im AS-Diagramm angezeigt werden, hängt von den Einstellungen in den TwinCAT-Optionen in der Kategorie **TwinCAT > SPS Programmierumgebung > AS-Editor**, Registerkarte **Ansicht** ab.

Allgemein

Eigenschaft	Beschreibung Wert
Name	Elementname, standardmäßig <Element><fortlaufende Zahl> , z. B. Schrittname „Schritt0“, „Schritt1“, Verzweigungsname „Verzweigung0“ etc.
Kommentar	Elementkommentar (Text), z. B. Zähler Reset. Sie können Zeilenumbrüche mit **[Strg] + [Eingabe]** einfügen.
Symbol	Für jedes AS-Element legt TwinCAT eine implizite Variable an, mit dem gleichen Namen wie das Element.

Spezifisch

Eigenschaft	Beschreibung Wert
Initialschritt	☑ : Diese Option ist immer nur für den Schritt aktiviert, der gerade als Initialschritt definiert ist. Standardmäßig ist dies der erste Schritt in einem AS-Diagramm. Wenn Sie diese Eigenschaft für einen anderen Schritt aktiviert haben, muss sie für den Schritt, der die Eigenschaft vorher hatte, deaktiviert werden, um Übersetzungsfehler zu vermeiden.
Duplizieren oder kopieren	Diese Option ist für Schritte verfügbar, die eine Schrittaktion (Eingangsaktion, Hauptaktion oder Ausgangsaktion) enthalten, und für Transitionen, die mit einem Transitionsobjekt verknüpft sind. ☑ : Beim Kopieren des Schritts/der Transition wird für die aufgerufenen Aktionen/Transitionen jeweils ein neues Objekt angelegt. Es enthält eine Kopie des Implementierungscodes des kopierten Objekts. ☐ : Beim Kopieren des Schritts oder der Transition wird für die zugehörigen Aktionen/Transitionen die Verknüpfung zum jeweiligen aufgerufenen Objekt beibehalten. Es entstehen keine neuen Objekte. Quelle und Kopien des Schritts oder der Transition rufen dieselbe Aktion/Transition auf. Eingebettete Objekte werden im SPS-Projektbaum mit einem Unterstrich im Namen angezeigt. Mit den Befehlen **Change duplication > Setzen** und **Entfernen** können Sie alle Schrittaktionen oder Transitionen, die in einem AS-Baustein aufgerufen werden, „einbetten“ oder die Einbettung aufheben.
Zeiten • Minimal aktiv • Maximal aktiv	Minimale Zeit, die der Schritt aktiv ist, auch wenn die nachfolgende Transition TRUE ist. Maximale Zeit, die der Schritt aktiv sein darf. Wird die Zeit überschritten, setzt TwinCAT die implizite Variable SFCError auf TRUE. Zeitangaben gemäß der IEC-Syntax (z. B. t#8s) oder TIME Variable; Default: t#0s.
Aktionen • Eingangsaktion • Schrittaktion • Ausgangsaktion	• Eingangsaktion: TwinCAT führt diese Aktion aus, nachdem der Schritt aktiviert wurde. • Schrittaktion: TwinCAT führt diese Aktion aus, wenn der Schritt aktiv ist und eine eventuelle Eingangsaktion bereits abgearbeitet ist. • Ausgangsaktion: Wenn der Schritt deaktiviert wird, führt TwinCAT diese Aktion im darauf folgenden Zyklus aus. Beachten Sie die Abarbeitungsreihenfolge.

Mithilfe der entsprechenden impliziten AS-Variablen und -Flags können Sie Informationen zum Status eines Schritts oder einer Aktion oder zu Zeitüberschreitungen erhalten.

Siehe auch:

- Implizite Variablen [▶ 670]

- Dokumentation TC3 User Interface: Dialog Optionen - AS-Editor [▶ 1022]

16.1.5 Funktionsplan/Kontaktplan/Anweisungsliste (FUP/KOP/AWL)

16.1.5.1 FUP/KOP/AWL-Editor

Der FUP/KOP/AWL-Editor ist ein kombinierter Editor der Programmiersprachen FUP, KOP und AWL.

AWL kann bei Bedarf über die TwinCAT-Optionen aktiviert werden.
(**Extras > Optionen > TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL > AWL**)

Es gibt einen gemeinsamen Satz an Befehlen und Elementen und TwinCAT konvertiert die drei Programmiersprachen automatisch intern ineinander.

Der Code im Implementierungsteil ist in allen drei Sprachen mithilfe von Netzwerken strukturiert.

Das Menü **FUP/KOP/AWL** enthält Befehle für das Arbeiten im Editor.

Im Offline- und Onlinebetrieb können Sie jederzeit über den Befehl zwischen den drei Editoransichten umschalten.

Das Verhalten des FUP/KOP/AWL-Editors ist durch die Einstellungen im Menü **Extras > Optionen**, Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL** definiert.

Es gibt einige spezielle Elemente, die TwinCAT nicht konvertieren kann und somit nur in der passenden Sprache anzeigt. Ebenso gibt es Konstrukte, die zwischen AWL und FUP nicht eindeutig konvertierbar sind und deshalb bei der Rückumwandlung in FUP „normalisiert“, also wieder aufgehoben werden. Dies betrifft: Negation von Ausdrücken und explizite/implizite Zuweisung von Funktionsbaustein-Eingängen und -Ausgängen.

Siehe auch:

- Generelle Funktionalitäten in allen grafischen Editoren [▶ 655]
- Dokumentation TC3 User Interface: FUP/KOP/AWL [▶ 1072]
- Dokumentation TC3 User Interface: Dialog Optionen - FUP, KOP und AWL [▶ 1014]

FUP- und KOP-Editor

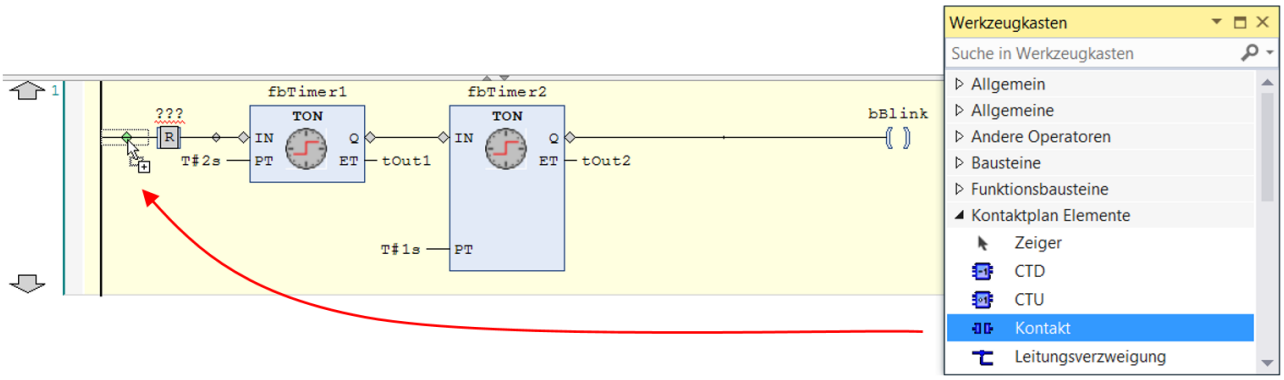
Einfügen und Anordnen von Elementen:

Sie können Elemente durch Ziehen mit der Maus aus der Ansicht **Werkzeugkasten** (Toolbox) in den Implementierungsteil des Editors ziehen. Alternativ können Sie die Befehle des Kontextmenüs oder des Menüs **FUP/KOP/AWL** verwenden.

Einstellungen zur Anzeige und Oberfläche definieren Sie in den TwinCAT-Optionen, Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL**.

Wenn Sie ein Element mit der Maus über ein Netzwerk im Editor ziehen, werden alle möglichen Einfügepositionen mit grauen rauten-, dreieck- oder pfeileförmigen Positionsmarken angezeigt. Sobald der Mauszeiger über einer dieser Marken steht, wird die Marke grün und durch Loslassen der Maustaste fügt TwinCAT das Element an der Position ein.

Beispiel:



Wenn Sie einen Funktionsbaustein oder Operator aus dem **Werkzeugkasten** oder einem Netzwerk an den linken Rand des Netzwerks auf einen der beiden Pfeile ziehen, geschieht Folgendes: TwinCAT erzeugt automatisch ein neues Netzwerk und fügt dort das Element ein.

Um ein Element zu ersetzen, ziehen Sie mit der Maus ein passendes anderes Element auf seine Position. Elemente, Sie durch das neue Element ersetzen können, kennzeichnet TwinCAT im Editor mit Textfeldern, zum Beispiel „Ersetzen“ oder „Eingang anhängen“.

Für das Ausschneiden, Kopieren, Einfügen und Löschen von Elementen können Sie die üblichen Befehle des Menüs **Bearbeiten** verwenden. Kopieren funktioniert auch mit Drag-and-drop bei gedrückt gehaltener Taste **[Strg]**.

Die Operatoren mit EN/ENO-Funktionalität können nur im FUP- und KOP-Editor eingefügt werden.

Selektieren von Elementen:

Mit einem Mausklick auf einen Baustein oder eine Verbindungslinie im Editor selektieren Sie diese(n), setzen also den Fokus darauf. Mehrfachselektion ist möglich, während die Taste **[Strg]** gedrückt gehalten wird. Ein selektiertes Element erhält eine rote Schattierung.

Tooltip:

Wenn der Cursor auf bestimmte Elemente zeigt, beispielsweise auf eine Variable oder auf einen Eingang, erscheint ein Tooltip mit Informationen zu diesem Element.

Von rot unterkringelten Elementen zeigt der Tooltip die Precompile-Fehlermeldung des Fehlers, der bei diesem Element auftritt.

Navigieren im Editor:

Mithilfe der im Folgenden beschriebenen Tasten und Befehle können Sie den Fokus innerhalb des Editors auf eine andere Cursorposition legen. Der Wechsel zwischen den Positionen funktioniert auch netzwerkübergreifend.	
[←] [→]	Wechsel zur benachbarten Cursorposition, entlang des Signalflusses, also von links nach rechts und umgekehrt.
[↑][↓]	Wechsel zur nächsten Cursorposition oberhalb oder unterhalb der aktuellen Position, wenn diese Nachbarposition zur gleichen logischen Gruppe gehört. Eine logische Gruppe bilden beispielsweise alle Anschlüsse eines Bausteins. Wenn eine solche logische Gruppe nicht existiert: Wechsel zur ersten Cursorposition im nächsten oberen oder unteren Nachbarelement. Im Fall von parallel verbundenen Elemente erfolgt die Navigation entlang des ersten Zweiges.
[Strg] + [Pos 1]	Wechsel ins erste Netzwerk; dieses wird selektiert.
[Strg] + [Ende]	Wechsel ins letzte Netzwerk; dieses wird selektiert.
[Bild nach oben]	Blättern um 1 Seite nach oben. Das oberste Netzwerk auf dieser Seite wird selektiert.
[Bild nach unten]	Blättern um 1 Seite nach unten. Das unterste Netzwerk auf dieser Seite wird selektiert.
Befehl "Gehe zu..."	Wechsel in ein bestimmtes Netzwerk.

Funktionsbaustein öffnen:

Wenn ein Funktionsbaustein im Editor eingefügt ist, können Sie dessen Implementierung durch einen Doppelklick oder mit dem Befehl **Gehe zu** des Kontextmenüs öffnen.

Siehe auch:

- Funktionsplan (FUP) [▶ 113]
- Programmieren in Funktionsplan (FUP) [▶ 115]
- Kontaktplan (KOP) [▶ 114]
- Programmieren in Kontaktplan (KOP) [▶ 116]
- Elemente [▶ 694]
- FUP/KOP/AWL-Editor im Onlinebetrieb [▶ 689]
- Dokumentation TC3 User Interface: Befehl Gehe zu [▶ 1085]

AWL-Editor

Einfügen und Anordnen von Elementen:

Sie können Elemente mithilfe der Befehle des Menüs **FUP/KOP/AWL** oder des Kontextmenüs einfügen. Ein neues Netzwerk können Sie auch zusätzlich per Drag-and-drop aus dem **Werkzeugkasten** in den Implementierungsteil des Editors ziehen.

Für das Ausschneiden, Kopieren, Einfügen und Löschen von Elementen können Sie die üblichen Befehle des Menüs Bearbeiten verwenden. Kopieren funktioniert auch mit Drag-and-drop bei gedrückt gehaltener Taste **[Strg]**.

Beachten Sie, dass Operatoren mit EN/ENO Funktionalität nur im FUP- und KOP-Editor eingefügt werden können.

Jede Programmzeile wird in einer Tabellenzeile eingetragen.

Struktur eines Netzwerks im AWL-Editor:

1. Zeile: Netzwerktitel Voraussetzung: Die Option ist in den TwinCAT-Optionen aktiviert		
2. Zeile: Netzwerkkommentar Voraussetzung: Die Option ist in den TwinCAT-Optionen aktiviert		
ab 3. Zeile:		
Spalte	**Inhalt**	**Beschreibung**
1	Operator	Enthält den AWL-Operator (LD, ST, CAL, AND, OR etc.) oder einen Funktionsnamen. Wenn Sie einen Funktionsbaustein aufrufen, müssen Sie hier zusätzlich die entsprechenden Parameter angeben; im vorhergehenden Feld müssen Sie in diesem Fall := oder => eingeben.
2	Operand	Enthält genau einen Operanden oder den Namen einer Sprungmarke. Bei mehreren Operanden, müssen Sie diese in mehreren Zeilen eingeben und dabei direkt hinter den einzelnen Operanden ein Komma einfügen. (Beispiel siehe unten)
3	Adresse	Enthält die Adresse des Operanden, wie bei dessen Deklaration definiert. nicht editierbar Sie können die Anzeige über die Option Symboladresse anzeigen aktivieren/deaktivieren. Wählen Sie dazu den Befehl **Extras > Optionen** und in der Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL** die Registerkarte **Allgemeines**.
4	Symbolkommentar	Enthält den Kommentar, der gegebenenfalls für den Operanden in der Deklaration angegeben wurde. nicht editierbar Sie können die Anzeige über die Option **Symbolkommentar anzeigen** aktivieren/deaktivieren, wenn Sie den Befehl **Extras > Optionen** auswählen und in der Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL** die Registerkarte **Allgemeines** auswählen.
5	Operandenkommentar	Kommentar für die aktuelle Programmzeile. Sie können die Anzeige über die Option **Operandenkommentar anzeigen** aktivieren/deaktivieren, wenn Sie den Befehl **Extras > Optionen** auswählen und in der Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL** die Registerkarte **Allgemeines** auswählen.

Beispiel:

```
CAL          fbTonInst1(
         IN:= bVar,
         PT:= tTime1,
         ET=> tOut1)
LD           fbTonInst1.Q          (* gets TRUE, delay time (PT) aft… *)
ST           fbTonInst2.IN         (* starts timer with rising edge,… *)
CAL          fbTonInst2(
         PT:= tTime2,
          Q=> bReady,              %Q*   (* for fbTonInst2 *)
         ET=> tOut2)
```

Navigieren im Editor:

Taste(n)/Befehl	Cursorbewegung
[↑][↓]	Springt zum oberhalb/unterhalb liegenden Feld.
[Tab]	Springt nach rechts zum nächsten Feld innerhalb der Zeile.
[Umschalttaste] + [Tab]	Springt nach links zum vorhergehenden Feld innerhalb der Zeile
[Leerzeichen]	Öffnet den Editierrahmen für das selektierte Feld. Alternativ können Sie einen Mausklick auf das Feld ausführen. Gegebenenfalls steht die Schaltfläche für den **Eingabehilfe**-Dialog zur Verfügung.
[Strg] + [Eingabe]	Fügt eine neue Zeile unterhalb der aktuellen Zeile ein.
[Entf]	Löscht die aktuelle Zeile.
[Strg] + [Pos 1]	Setzt den Fokus auf den Dokumentanfang und markiert das erste Netzwerk.
[Strg] + [Ende]	Setzt den Fokus auf das Dokumentende und markiert das letzte Netzwerk.
[Bild nach unten]	Blättert eine Seite hoch und markiert das oberste Rechteck.
[Bild nach oben]	Blättert eine Seite nach unten und markiert das oberste Rechteck.

Siehe auch:

- Anweisungsliste (AWL) [▶ 114]
- Programmieren in Anweisungsliste (AWL) [▶ 117]
- Modifikatoren und Operatoren in AWL [▶ 690]
- FUP/KOP/AWL-Editor im Onlinebetrieb [▶ 689]

16.1.5.2 FUP/KOP/AWL-Editor im Onlinebetrieb

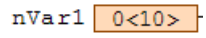
Im Onlinebetrieb wird im Editor hinter jeder Variablen ihr aktueller Wert dargestellt. Schreiben/Forcen und das Setzen von Haltepunkten ist möglich.

Wenn die Variable gerade geforct ist, wird dies durch F direkt vor dem geforcten Wert angezeigt. Wenn ein Wert für das Schreiben oder Forcen vorbereitet ist, wird dieser Wert direkt hinter dem aktuellen Wert in eckigen Klammern <value> angezeigt.

Beispiel:

Geforcte Variable:

Vorbereiteter Wert



In der Online-Ansicht eines Kontaktplans (KOP) sind die Anschlussleitungen farblich gekennzeichnet: Verbindungen mit dem Wert TRUE werden als fette blaue Linie dargestellt, Verbindungen mit dem Wert FALSE als fette schwarze Linie. Dagegen werden Verbindungen von unbekanntem oder analogem Wert normal dargestellt werden (dünne schwarze Linie).

Beachten Sie, dass der Wert der Verbindungen aus den gemonitorten Variablen berechnet wird. Es handelt sich dabei nicht um eine echte Ablaufkontrolle.

Haltepunkte

Mögliche Positionen für Haltepunkte sind grundsätzlich die Positionen, an denen sich Variablenwerte ändern können (Anweisungen), an denen sich das Programm verzweigt oder an denen ein anderer Baustein aufgerufen wird.

Mögliche Haltepunktpositionen:

- Auf dem gesamten Netzwerk: Bewirkt, dass der Haltepunkt an der ersten möglichen Position im Netzwerk gesetzt wird.
- Auf einer Bausteinbox, wenn der Baustein eine Zuweisung beinhaltet. Nicht möglich bei Operatoren-Bausteinen, beispielweise ADD, DIV.
- Auf Zuweisungen.
- Am Ende des Bausteins an der Position der Rückkehr zum aufrufenden Bausteins. Im Onlinebetrieb erscheint automatisch ein leeres Netzwerk an dieser Stelle, das anstelle einer Netzwerknummer mit „RET“ gekennzeichnet ist.

Gegenwärtig können Sie auf den ersten Baustein im Netzwerk nicht direkt einen Haltepunkt setzen. Wenn Sie jedoch einen Haltepunkt auf das gesamte Netzwerk setzen, wird diese Haltepunktmarkierung im Onlinebetrieb automatisch auf den ersten Baustein übertragen.

Haltepunkte in Methoden: TwinCAT setzt automatisch einen Haltepunkt in allen Methoden, die aufgerufen werden können. Wenn also eine durch eine Schnittstelle verwaltete Methode aufgerufen wird, werden Haltepunkte in allen Methoden gesetzt, die in Funktionsbausteinen vorkommen, die diese Schnittstelle implementieren; ebenso in allen abgeleiteten Funktionsbausteinen, die die Methode verwenden. Wenn eine Methode durch einen Zeiger auf einen Funktionsbaustein aufgerufen wird, setzt TwinCAT die Haltepunkte in der Methode des Funktionsbausteins und in allen abgeleiteten Funktionsbausteinen, die die Methode verwenden.

Siehe auch:

- Forcen und Schreiben von Variablen [▶ 225]
- Haltepunkte verwenden [▶ 222]

16.1.5.3 Modifikatoren und Operatoren in AWL

Modifikatoren:

Modifikator	Kombiniert mit Operator	Beschreibung
C	JMP, CAL, RET	Die Anweisung wird nur ausgeführt, wenn das Ergebnis des vorhergehenden Ausdrucks TRUE ist.
N	JMPC, CALC, RETC	Die Anweisung wird nur ausgeführt, wenn das Ergebnis des vorhergehenden Ausdrucks FALSE ist.
N	ansonsten	Negation des Operanden (nicht des Akkus).

Operatoren mit den möglichen Modifikatoren:

Operator	N	Bedeutung	Beispiel
LD	N	Lädt den (negierten) Wert des Operanden in den Akkumulator.	LD ivar
ST	N	Speichert den (negierten) Inhalt des Akkumulators in den Operanden.	ST iErg
S		Setzt den Operanden (Typ BOOL) auf TRUE, wenn der Inhalt des Akkumulators TRUE ist.	S bVar1
R		Setzt den Operanden (Typ BOOL) auf FALSE, wenn der Inhalt des Akkumulators TRUE ist.	R bVar1
AND	N,(	Bitweises AND des Akkumulatorwerts und des (negierten) Operanden	AND bVar2
OR	N,(	Bitweises OR des Akkumulatorwerts und des (negierten) Operanden	OR xVar
XOR	N,(	Bitweises, exklusives OR des Akkumulatorwerts und des (negierten) Operanden	XOR N,(bVar1,bVar2)
NOT		Bitweise Negation des Akkumulatorwerts	
ADD	(	Addition des Akkumulatorwerts und des Operanden; Ergebnis wird in den Akkumulator geschrieben.	ADD ivar1
SUB	(	Subtraktion des Operanden vom Akkumulator-Wert; Ergebnis wird in den Akkumulator geschrieben.	SUB iVar2
MUL	(	Multiplikation von Akkumulatorwert und Operand; Ergebnis wird in den Akkumulator geschrieben.	MUL ivar2
DIV	(	Division des Akkumulatorwerts durch den Operanden; Ergebnis wird in den Akkumulator geschrieben.	DIV 44
GT	(	Prüft, ob der Akkumulatorwert größer als der Operandenwert ist; Ergebnis (BOOL) wird in den Akkumulator geschrieben; >	GT 23
GE	(	Prüft, ob der Akkumulatorwert größer oder gleich dem Operandenwert ist; Ergebnis (BOOL) wird in den Akkumulator geschrieben.	GE iVar2
EQ	(	Prüft, ob der Akkumulatorwert gleich dem Operandenwert ist; Ergebnis (BOOL) wird in den Akkumulator geschrieben.	EQ iVar2
NE	(	Prüft, ob der Akkumulatorwert ungleich dem Operandenwert ist; Ergebnis (BOOL) wird in den Akkumulator geschrieben.	NE iVar1
LE	(	Prüft, ob der Akkumulatorwert kleiner oder gleich dem Operandenwert ist; Ergebnis (BOOL) wird in den Akkumulator geschrieben.	LE 5
LT	(	Prüfen, ob der Akkumulator-Wert kleiner als der Operandenwert ist; Ergebnis (BOOL) wird in den Akkumulator geschrieben.	LT cVar1
JMP	CN	Unbedingter (bedingter) Sprung zur angegebenen Sprungmarke	JMPN next
CAL	CN	(Bedingter) Aufruf eines Programms oder Funktionsbausteins (wenn der Akkumulator-Wert TRUE ist)	CAL prog1
RET		Verlassen des Bausteins und Rückkehr zum aufrufenden Baustein	RET
RET	C	Wenn der Akkumulatorwert TRUE ist: Verlassen des Bausteins und Rückkehr zum aufrufenden Baustein.	RETC
RET	CN	Wenn der Akkumulatorwert FALSE ist: Verlassen des Bausteins und Rückkehr zum aufrufenden Baustein.	RETCN
)		Auswerten der zurückgestellten Operation	

Beispiel:

```
1
    AND     TRUE    load TRUE to accumulator
    ANDN    bVar1   execute AND with negated value of bVar1
    JMPC    m1      if accum. is TRUE, jump to label "m1"

    LDN     bVar2   store negated value of bVar2...
    ST      bRes    ... in bRes

2   m1:

    LD      bVar2   store value of bVar2...
    ST      bRes    ... in bRes
```

Anwendung	Beschreibung	Beispiel

Mehrere Operanden für 1 Operator	Möglichkeiten
	• Sie geben die Operanden in aufeinander folgenden Zeilen, getrennt durch Komma in der 2. Spalte ein.
	• Sie wiederholen den Operator in aufeinander folgenden Zeilen.
		Variante 1:
		`LD 2`
		`ADD 3,`
		`4,`
		`6,`
		`ST nVar`
		Variante 2:
		`LD 2`
		`ADD 3`
		`ADD 4`
		`ADD 6`
		`ST nVar`

Komplexe Operanden	Für einen komplexen Operanden geben Sie in der ersten Spalte die öffnende Klammer (an. Die schließende Klammer geben Sie in der ersten Spalte in einer separaten Zeile im Anschluss an die Operandeneinträge der folgenden Zeilen ein.	Ein String wird jeden Zyklus um ein Zeichen rotiert:
		`LD sRotate`
		`RIGHT( sRotate`
		`LEN`
		`SUB 1`
		`)`
		`CONCAT( sRotate`
		`LEFT 1`
		`)`
		`ST sRotate`

Funktionsbaustein-Aufruf, Programmaufruf	Spalte 1: Operator CAL oder CALC
	Spalte 2: Name der Funktionsbausteins-Instanz oder des Programms und öffnende Klammer (. Wenn keine Parameter folgen, wird hier die schließende Klammer) eingegeben.
	darauf folgende Zeilen:
	Spalte 1: Parametername gefolgt von := für Eingabeparamer oder => für Ausgabeparameter
	Spalte 2: Parameterwert gefolgt von Komma, wenn weitere Parameter folgen. Nach dem letzten Parameter, wird die schließende Klammer) eingegeben.
	Als Einschränkung bezüglich des IEC-Standards können hier komplexe Ausdrücke nicht verwendet werden. Solche Konstrukte müssen Sie dem Funktionsbaustein oder dem Programm noch vor dem Aufruf zuweisen.
		`CAL ProgToCall(`
		`nCounter:= 1,`
		`nDecrement:= 1000,`
		`nError=> nResult)`
		`LD ProgToCall.bError`
		`ST bErr`

Funktionsaufruf	Zeile 1: Spalte 1: LD
	Spalte 2: Eingabevariable
	Zeile 2: Spalte 1: Funktionsname Spalte2: weitere Eingabeparameter durch Komma getrennt.
	TwinCAT schreibt den Rückgabewert in den Akkumulator.
	Zeile 3: Spalte 1: ST Spalte 2: Variable, in die der Rückgabewert geschrieben wird
		`LD nVar7`
		`F_GeomAverage 25`
		`ST nAve`

Aktionsaufruf	Wie Funktionsbaustein- oder Programmaufruf. Der Aktionsname wird an den FB-Instanz- oder Programmnamen angehängt.	`CAL ProgToCall.ResetAction`

Anwendung	Beschreibung	Beispiel
Sprung	Spalte 1: Operator JMP oder JMPC. Spalte 2: Name der Sprungmarke des Ziel-Netzwerks. Bei einem unbedingten Sprung muss die voraus gehenden Anweisungsfolge mit einem der folgenden Kommandos enden: ST,STN, S, R, CAL, RET, JMP Bei einem bedingten Sprung hängt die Ausführung des Sprungs vom geladenen Wert ab.	

Siehe auch:

- Anweisungsliste (AWL) [▶ 114]
- Programmieren in Anweisungsliste (AWL) [▶ 117]

16.1.5.4 Elemente

16.1.5.4.1 FUP/KOP/AWL-Element Netzwerk

Symbol: 

Ein Netzwerk ist die Basiseinheit eines FUP- oder KOP-Programms. Im FUP/KOP/AWL-Editor sind die Netzwerke in einer Liste untereinander angeordnet. Jedes Netzwerk ist an der linken Seite mit einer fortlaufenden Netzwerknummer versehen und kann Folgendes enthalten: logische und arithmetische Ausdrücke, Programm-, Funktions- und Funktionsbausteinaufrufe, eine Sprung- oder Return-Anweisung.

Ein AWL-Programm besteht aus mindestens einem Netzwerk. Dieses Netzwerk kann alle AWL-Anweisungen des Programms enthalten.

Sie können jedes Netzwerk mit einem Titel, einem Kommentar oder einer Marke versehen. In den TwinCAT-Optionen, Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL** können Sie definieren, ob Netzwerktitel, Netzwerkkommentar und Trennlinien zwischen den einzelnen Netzwerken im Editor angezeigt werden.

Um einen Netzwerktitel einzugeben, klicken Sie auf die 1. Zeile des Netzwerks. Zur Eingabe eines Kommentars klicken Sie auf die 2. Zeile des Netzwerks.

Siehe auch:

- Dokumentation TC3 User Interface: Dialog Optionen - FUP, KOP und AWL [▶ 1014]
- Dokumentation TC3 User Interface: Befehl Netzwerk einfügen [▶ 1072]

16.1.5.4.2 FUP/KOP/AWL-Element Baustein

Symbol: 

Ein Baustein und dessen Aufruf können zusätzliche Funktionen repräsentieren, beispielsweise IEC-Funktionsbausteine, IEC-Funktionen, Bibliotheksbausteine, Operatoren.

Ein Baustein kann beliebige Ein-und Ausgänge haben.

Wenn der Baustein eine Bilddatei mitliefert, wird das Bausteinsymbol innerhalb des Bausteins angezeigt. Voraussetzung ist, dass in den TwinCAT-Optionen, Kategorie **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL** die Option **Bausteinsymbol anzeigen** aktiviert ist.

Wenn Sie die Bausteinschnittstellen geändert haben, können Sie die Bausteinparameter mit dem Befehl **Parameter aktualisieren** im Menü **FUP/KOP/AWL** aktualisieren, ohne dass Sie den Baustein neu einfügen müssen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Parameter aktualisieren [▶ 1083]
- Dokumentation TC3 User Interface: Dialog Optionen - FUP, KOP und AWL [▶ 1014]
- Dokumentation TC3 User Interface: Befehl Bausteinaufruf einfügen [▶ 1073]

16.1.5.4.3 FUP/KOP/AWL-Element Zuweisung

Symbol:

Der FUP-Editor zeigt eine neu eingefügte Zuweisung als Linie mit nachfolgenden drei Fragezeichen an. Der KOP-Editor zeigt eine neu eingefügte Zuweisung als Spule mit darüber liegenden drei Fragezeichen an.

Nach den Einfügen können Sie den Platzhalter **???** durch den Namen der Variablen ersetzen, der das von links kommende Signal zugewiesen werden soll. Dafür steht Ihnen die **Eingabehilfe** zur Verfügung.

In AWL wird eine Zuweisung über die Operatoren LD und ST programmiert.

Siehe auch:

- FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation TC3 User Interface: Befehl Zuweisung einfügen [▶ 1073]

16.1.5.4.4 FUP/KOP/AWL-Element Baustein mit EN/ENO

Symbol: 

Das Element steht nur im FUP- und im KOP-Editor zur Verfügung.

Der Baustein entspricht generell dem FUP/KOP/AWL-Element **Baustein**. Zusätzlich enthält dieser Baustein jedoch einen EN-Eingang und einen ENO-Ausgang. EN und ENO haben den Datentyp BOOL.

Funktion des EN-Eingangs und ENO-Ausgangs: Wenn der Eingang EN zum Zeitpunkt des Bausteinaufrufs den Wert FALSE hat, werden die im Baustein definierten Operationen nicht ausgeführt. Andernfalls, also wenn EN den Wert TRUE hat, werden diese Operationen ausgeführt. Der ENO-Ausgang hat den gleichen Wert wie der EN-Eingang.

Siehe auch:

- FUP/KOP/AWL-Element Baustein [▶ 694]
- Dokumentation TC3 User Interface: Befehl Baustein mit EN/ENO einfügen [▶ 1074]

16.1.5.4.5 FUP/KOP/AWL-Element Eingang

Symbol: 

Die maximale Anzahl von Eingängen hängt vom Bausteintyp ab.

Ein neu hinzugefügter Eingang ist zunächst mit **???** belegt. Sie können die Zeichenfolge **???** mit einer Variablen oder einer Konstanten ersetzen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Bausteineingang einfügen [▶ 1076]

16.1.5.4.6 FUP/KOP/AWL-Element Sprungmarke

Die Marke ist ein optionaler Kennzeichner für ein Netzwerk in FUP und KOP, den Sie als Ziel für einen Sprung angeben können.

Wenn Sie in einem Netzwerk eine Sprungmarke einfügen, wird diese im Netzwerk als editierbares Feld **Label:** hinzugefügt.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Sprungmarke einfügen [▶ 1075]

16.1.5.4.7 FUP/KOP/AWL-Element Sprung

Symbol: 

In FUP oder KOP wird ein Sprung abhängig von der aktuellen Cursorposition entweder direkt vor einem Eingang, direkt nach einem Ausgang oder am Ende des Netzwerks eingefügt.

Direkt hinter dem Sprungelement geben Sie eine Sprungmarke als Sprungziel ein.

In AWL programmieren Sie einen Sprung mit der Anweisung JMP.

Siehe auch:

- FUP/KOP/AWL-Element Sprungmarke [▶ 696]
- Dokumentation TC3 User Interface: Befehl Sprung einfügen [▶ 1075]

16.1.5.4.8 FUP/KOP/AWL-Element Return

Das Element dient dazu, die Abarbeitung des Bausteins sofort abzubrechen, wenn der Eingang des RETURN-Elements TRUE wird.

In einem FUP- oder KOP-Netzwerk können Sie die Return-Anweisung parallel zu oder anschließend an die vorausgehenden Elemente platzieren.

In AWL steht Ihnen für diesen Zweck die Anweisung RET zur Verfügung.

Siehe auch:

- Modifikatoren und Operatoren in AWL [▶ 690]
- Dokumentation TC3 User Interface: Befehl Return einfügen [▶ 1075]

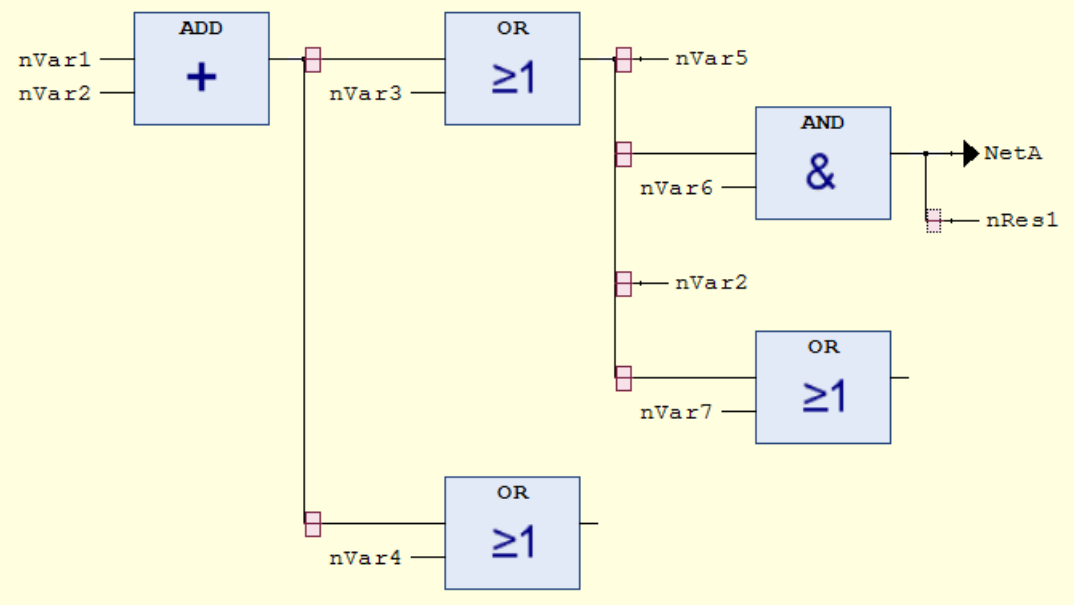
16.1.5.4.9 FUP/KOP/AWL-Element Leitungsverzweigung

Symbol: 

Das Element steht im KOP- und FUP-Editor zur Verfügung und stellt eine offene Leitungsverzweigung dar. Eine Leitungsverzweigung spaltet die Abarbeitungslinie ab der aktuellen Cursor-Position in 2 Subnetzwerke, die von oben nach unten nacheinander ausgeführt werden. Jedes Subnetzwerk können Sie weiter verzweigen, wodurch innerhalb eines Netzwerks Mehrfachverzweigungen entstehen.

Jedes Subnetzwerk erhält am Verzweigungspunkt ein Marker-Symbol (Rechteck), das Sie auswählen können, um weitere Befehle auszuführen.

Die Befehle Kopieren, Ausschneiden und Einfügen stehen für Subnetzwerke nicht zur Verfügung.



Um ein Subnetzwerk zu löschen, müssen Sie zuerst alle Elemente des Netzwerks und anschließend das Markersymbol des Subnetzwerks löschen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Leitungsverzweigung einfügen [▶ 1081]
- Dokumentation TC3 User Interface: Befehl Leitungsverzweigung oberhalb einfügen [▶ 1081]
- Dokumentation TC3 User Interface: Befehl Leitungsverzweigung unterhalb einfügen [▶ 1082]

16.1.5.4.10 FUP/KOP/AWL-Element Execute

Symbol: 

Das Element ist ein Baustein, der dazu dient, dass Sie im FUP- und im KOP-Editor direkt ST-Code eingeben können.

Sie können das Element **Execute** aus der Ansicht **Werkzeugkasten** mit der Maus in den Implementierungsteil Ihrer POU ziehen. Wenn Sie im Baustein auf das Feld **Enter ST-Code here…** klicken, öffnet sich ein Eingabefeld, wo Sie mehrzeiligen ST-Code eingeben können.

16.1.5.4.11 KOP-Element Kontakt

Symbol:  , im Editor: 

Das Element ist nur im KOP-Editor verfügbar.

Ein Kontakt gibt das Signal TRUE (ON) oder FALSE (OFF) von links nach rechts weiter, bis das Signal schließlich eine Spule im rechten Teil des Netzwerks erreicht. Zu diesem Zweck wird dem Kontakt eine boolesche Variable zugewiesen, die das Signal enthält. Dazu ersetzen Sie den Platzhalter **???** oberhalb des Kontakts mit dem Namen einer booleschen Variablen

Sie können mehrere Kontakte sowohl in Reihe als auch parallel anordnen. Bei zwei parallelen Kontakten muss nur einer den Wert TRUE erhalten, damit ON nach rechts weitergegeben wird. Wenn Kontakte hintereinander geschaltet sind, müssen alle Kontakte den Wert TRUE erhalten, damit vom letzten Kontakt der Reihe ON nach rechts weitergegeben wird. Sie können also mit KOP elektrische Parallel- und Reihenschaltungen programmieren.

Ein negierter Kontakt  gibt das Signal TRUE weiter, wenn der Variablenwert FALSE ist. Sie können einen eingefügten Kontakt mithilfe des Befehls **Negation** oder einen negierten Kontakt aus der Ansicht **Werkzeugkasten** einfügen.

Wenn Sie bei selektiertem Netzwerk mit der Maus auf einen Kontakt gehen und die linke Maustaste gedrückt halten, erscheint im Netzwerk die Schaltfläche In Spule umwandeln. Wenn Sie nun mit der weiterhin gedrückten Maustaste den Mauszeiger auf diese Schaltfläche bewegen und über der Schaltfläche die Maustaste loslassen, wandelt TwinCAT den Kontakt in eine Spule um.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Negation [▶ 1080]
- Dokumentation TC3 User Interface: Befehl Kontakt einfügen [▶ 1077]
- Dokumentation TC3 User Interface: Befehl Negierten Kontakt einfügen [▶ 1078]
- Dokumentation TC3 User Interface: Befehl Kontakt einfügen (rechts) [▶ 1072]
- Dokumentation TC3 User Interface: Befehl Kontakt parallel einfügen (oberhalb) [▶ 1078]
- Dokumentation TC3 User Interface: Befehl Kontakt parallel einfügen (unterhalb) [▶ 1077]

16.1.5.4.12 KOP-Element Spule

Symbol:  , im Editor: 

Das Element ist nur im KOP-Editor verfügbar.

Eine Spule nimmt den Wert auf, der von links geliefert wird und speichert ihn in der ihr zugewiesenen booleschen Variablen. Ihr Eingang kann den Wert TRUE (ON) oder FALSE (OFF) haben.

Mehrere Spulen in einem Netzwerk können nur parallel angeordnet werden.

In einer negierten Spule  wird der negierte Wert des eingehenden Signals in der booleschen Variablen gespeichert, die der Spule zugewiesen ist.

Set-Spule und Reset-Spule

Symbol:  ,  , im Editor:  , 

Set-Spule: Wenn der Wert TRUE an einer Set-Spule ankommt, behält die Spule den Wert TRUE. Solange das SPS-Programm läuft, kann der Wert an dieser Stelle nicht mehr überschrieben werden.

Reset-Spule: Wenn der Wert TRUE an einer Reset-Spule ankommt, behält die Spule den Wert FALSE. Solange das SPS-Programm läuft, kann der Wert an dieser Stelle nicht mehr überschrieben werden.

Sie können eine eingefügte Spule mithilfe des Befehls **Set/Rese**t im Menü **FUP/KOP/AWL** als Set- oder Reset-Spule definieren oder als Elemente Set-Spule und Reset-Spule aus der Ansicht **Werkzeuge** einfügen.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Set/Reset [▶ 1081]
- Dokumentation TC3 User Interface: Befehl Spule einfügen [▶ 1076]
- Dokumentation TC3 User Interface: Befehl Reset-Spule einfügen [▶ 1077]

16.1.5.4.13 KOP-Element Leitungsverzweigung Start/Ende

Symbol: 

Das Element dient der geschlossenen Leitungsverzweigung.

Siehe auch:

- Geschlossene Leitungsverzweigung [▶ 699]
- Dokumentation TC3 User Interface: Befehl Verzweigung Startpunkt setzen [▶ 1082]
- Dokumentation TC3 User Interface: Befehl Verzweigung Endpunkt setzen [▶ 1082]

16.1.5.4.14 Geschlossene Leitungsverzweigung

Eine geschlossene Leitungsverzweigung steht nur im KOP zur Verfügung und enthält einen Start- und einen Endpunkt. Sie dient der Implementierung einer parallelen Auswertung von logischen Elementen.

Einfügen einer geschlossenen Leitungsverzweigung:

- Dokumentation TC3 User Interface: Befehl Kontakt parallel einfügen (unterhalb) [▶ 1077]
- Dokumentation TC3 User Interface: Befehl Kontakt parallel einfügen (oberhalb) [▶ 1078]
- Dokumentation TC3 User Interface: Befehl Verzweigung Endpunkt setzen [▶ 1082]
- Dokumentation TC3 User Interface: Befehl Verzweigung Startpunkt setzen [▶ 1082]

Geschlossene Leitungsverzeigung an einem Kontakt

Wenn Sie einen oder mehrere Kontakt markiert haben und Sie einen Befehl **Kontakt parallel einfügen** ausführen, wird eine parallele Verzweigung mit einem einfachen vertikalen Strich eingefügt. Bei dieser Verzweigung durchläuft der Signalfluss beide Zweige. Es handelt sich um ein OR-Konstrukt der beiden Zweige.

Geschlossene Leitungsverzweigung an einem Baustein

Wenn Sie einen Baustein markiert haben und Sie einen Befehl "Kontakt parallel einfügen" ausführen, wird eine parallele Verzweigung mit einem doppelten vertikalen Strich eingefügt. Dies signalisiert, dass eine Kurzschlussauswertung („Short Circuit Evaluation“ – „SCE“) implementiert ist. Die Kurschlussauswertung ermöglicht, die Ausführung eines Funktionsbausteins mit einem booleschen Ausgang zu umgehen, wenn eine bestimmte Bedingung TRUE ist. Die Bedingung kann im KOP-Editor durch einen Funktionsbausteinzweig parallel geschalteten Zweig dargestellt werden. Die Kurzschlussbedingung wird durch einen oder mehrere Kontakte in diesem Zweig definiert, die parallel oder sequentiell verschaltet sind.

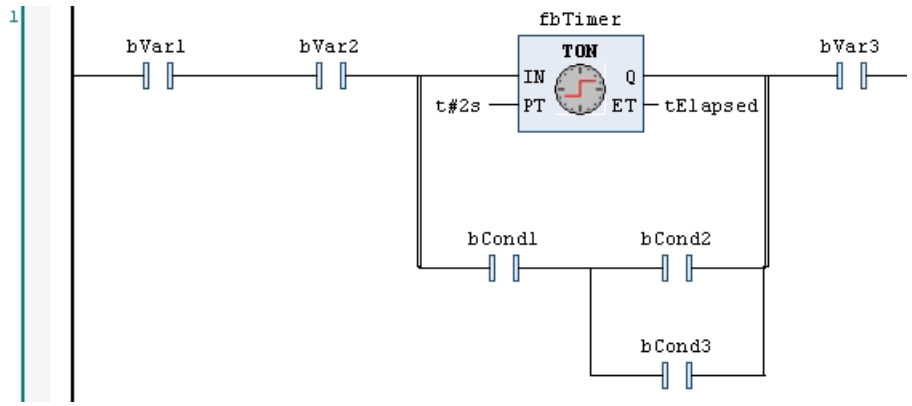
Funktionsweise:

Zuerst werden die Zweige betrachtet, die nicht den Funktionsbaustein enthalten. Wenn TwinCAT für einen dieser Zweige der Wert TRUE ermittelt, dann wird der Funktionsbaustein im parallelen Zweig nicht aufgerufen. In diesem Fall wird der Wert, der am Eingang des Funktionsbausteins anliegt, unmittelbar auf den Ausgang übertragen. Wenn TwinCAT für die Kurzschlussauswertungsbedingung FALSE ermittelt wird, dann wird der Baustein aufgerufen und das boolesche Ergebnis seiner Abarbeitung wird weitergegeben. Wenn alle Zweige Funktionsbausteine enthalten, werden sie von oben nach unten ausgewertet und ihre Ausgänge mit logischen OR-Operationen verknüpft. Wenn es keine Zweige mit Funktionsbausteinen gibt, werden normale OR-Operationen durchgeführt.

Beispiel:

Die Funktionsbaustein-Instanz fbTimer (TON) hat einen booleschen Eingang und einen booleschen Ausgang. Die Ausführung von fbTimer wird ausgelassen, wenn für die Bedingung in der parallelen Leitungsverzweigung TRUE ermittelt wurde. Der Bedingungswert resultiert aus den OR- und AND-Operationen, die die Kontakte bCond1, bCond2 und bCond3 verknüpfen.

Wenn der Bedingungswert aus der Verknüpfung der Kontakte bCond1, bCond2 und bCond3 FALSE ist, wird fbTimer ausgeführt.



(1) Die doppelt gezeichneten Vertikalverbindungen zeigen an, dass es sich um ein Konstrukt handelt, das einer Kurzschlussauswertung unterliegt.

(2) Die einfach gezeichneten Vertikalverbindungen zeigen an, dass es sich um ein OR-Konstrukt handelt.

Im Folgenden wird das oben gezeigt KOP-Beispiel als ST-Code gezeigt. Dabei sind bIN und bOUT die booleschen Werte am Eingang (Aufspaltungspunkt) und Ausgang (Wiedervereinigungspunkt) der parallelen Leitungsverzweigung.

```
bIN := bVar1 AND bVar2;

IF ((bIN AND bCond1) AND (bCond2 OR bCond3)) THEN
    bOUT := bIN;
ELSE
    fbTimer(IN := bIN, PT := {p 10}t#2s);
    tElapsed := fbTimer.ET;
    bOUT := fbTimer.Q;
END_IF

bRes := bOUT AND bVar3;
```

16.1.6 Continuous Function Chart (CFC) und Seitenorientierter CFC

Ein Funktionsplan besteht von außen betrachtet aus Eingängen und Ausgängen, zwischen denen Daten verarbeitet werden. Intern besteht ein Funktionsplan aus Bausteinen und deren Verbindungen, die Daten (Signale) repräsentieren und zeigen, wie Zuweisungsoperatoren in ST fungieren. Das Gesamtverhalten setzt sich zusammen aus dem Verhalten der eingefügten Bausteine, die andere Programmierbausteine oder Bibliotheksbausteine aufrufen.

Code in der Implementierungssprache Continuous Function Chart (CFC) veranschaulicht vor allem den Datenfluss durch das System. Ein Continuous Function Chart wird deshalb auch als Signalflussplan bezeichnet.

Im seitenorientierten CFC-Editor können Sie Programmierbausteine miteinander verdrahten und anschauliche Funktionspläne auf Seiten verteilt erstellen. Der seitenorientierte Editor verhält sich wie der CFC-Editor mit zusätzlicher Funktionalität:

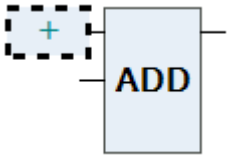
- Erstellen von Seiten
- Einstellen der Seitengröße
- Kopieren und Einfügen von Seiten im Seitennavigator
- Kopieren der Implementierung eines Programmierbausteins der Implementierungssprache CFC und Einfügen in einer Seite
- Übersichtliches und platzsparendes Arrangieren von Eingängen, Ausgängen und Verbindungsmarken in den Randbereichen
- Verbindung über Seiten hinweg mit Verbindungsmarken

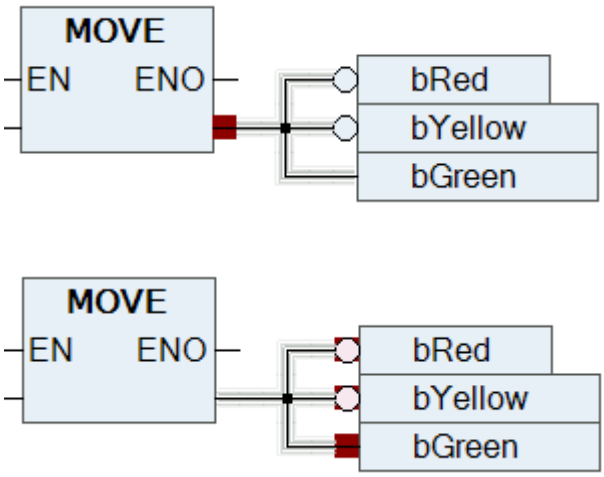
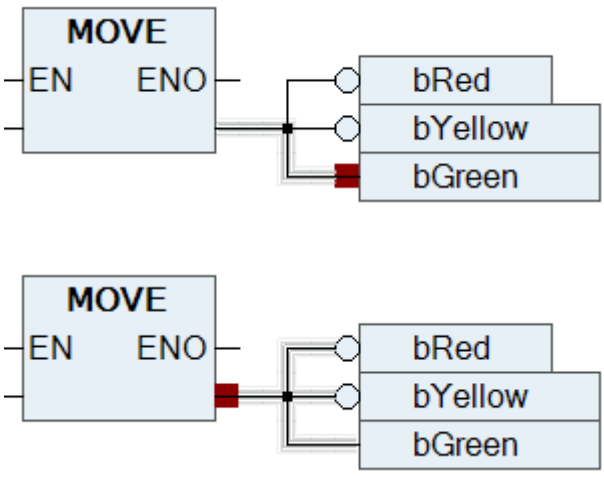
16.1.6.1 CFC-Editor

Editor konfigurieren

Sie können das Aussehen, Verhalten und Drucken in den TwinCAT-Optionen in der Kategorie **TwinCAT > SPS Programmierumgebung > CFC Editor** projektweit konfigurieren. Beispielsweise können Sie in der Registerkarte **Ansicht** die Farbe der Verbindungslinien abhängig vom Datentyp konfigurieren.

Editieren

Cursorsymbol	Voraussetzung: In der Ansicht **Werkzeugkasten** ist **Zeiger** selektiert. Das Symbol weist darauf hin, dass Sie im Editor editieren können. Selektieren Sie Elemente oder Verbindungen, um sie zu verschieben oder um Befehle auszuwählen.
Cursorsymbol	Voraussetzung: In der Ansicht **Werkzeugkasten** ist ein Element selektiert. Mit einem Mausklick in den Editor wird das selektierte Element eingefügt. Alternativ können Sie mit Drag-and-drop ein Element in den Editor ziehen.
Drag-and-drop einer Funktionsbaustein-Instanz aus der Deklaration in den Editor	Voraussetzung: In der Deklaration des CFCs ist eine Deklarationszeile selektiert. Die Instanz wird als Baustein mit Bausteinname und Bausteintyp und allen Anschlüssen eingefügt.
Drag-and-drop einer Variablen aus der Deklaration in den Editor an einen Bausteinanschluss	Die Variable wird als Eingang oder Ausgang mit Verbindung zum fokussierten Bausteinanschluss eingefügt. Tipp: Der Cursor zeigt an, ob Sie eine gültige Stelle für eine Variable fokussieren. 
Drag-and-drop einer Variablen aus dem Deklarationsteil in den Editor	Voraussetzung: In der Deklaration ist das jeweilige Element selektiert. • Funktionsbaustein-Instanz: Ein Baustein mit dem entsprechenden Datentyp wird erzeugt. • Deklaration von VAR_INPUT oder CONSTANT: Ein Eingangselement wird eingefügt. • Deklaration von VAR_OUTPUT: Ein Ausgangselement wird eingefügt. • Deklaration von VAR, VAR_GLOBAL: An der Einfügeposition öffnet sich ein Fenster, in dem Sie auswählen können, ob ein Eingangs- oder ein Ausgangselement eingefügt werden soll. Wenn eine Variable aus dem Deklarationsteil mittels Drag-and-drop auf ein bestehendes ersetzbares Element gezogen wird, wird das bestehende Element ersetzt.
Drag-and-drop eines Funktionsbausteins oder Programmierbausteins aus dem Projektbaum oder dem Bibliotheksverwalter in den Editor	Ein Bausteinelement mit dem entsprechenden Typ wird eingefügt. • Wenn ein Baustein mittels Drag-and-drop auf eine bestehende Verbindungslinie gezogen wird und sowohl ein Eingang als auch ein Ausgang des Bausteins zum Datentyp der Linie kompatibel sind, wird der Baustein auf der Linie eingefügt. Dabei werden sein erster passender Eingang und Ausgang mit den Elementen verbunden, die vorher durch die Verbindungslinie verbunden waren. • Wenn ein Baustein mittels Drag-and-drop auf einen bestehenden Baustein gezogen wird, wird der bestehende Baustein ersetzt.

Umsortieren der Reihenfolge von Eingängen und Ausgängen innerhalb eines Funktionsbausteins mittels Drag-and-drop.	Voraussetzung: Das Textfeld des Eingangs oder Ausgangs, die an eine andere Stelle umsortiert werden sollen, ist selektiert.
[Strg] + Klick in den Programmierbereich	Voraussetzung: In der Ansicht **Werkzeugkasten** ist ein Element selektiert. Solange Sie **[Strg]** gedrückt halten, wird bei jedem Klick in den Programmierbereich ein selektiertes Element erzeugt.
[Strg] + Pfeil nach rechts	Voraussetzung: Im CFC-Programm ist bei einem Element **genau** ein Ausgangsanschluss (Ausgangsspin) selektiert. Die Selektion wird so verschoben, dass der Eingangsanschluss (Eingangspin) am Ende der Verbindungslinie selektiert ist. Bei mehreren Anschlüssen sind alle selektiert. Beispiel: 
[Strg] + Pfeil nach links	Voraussetzung: Im CFC Programm ist bei einem Element **genau** ein Eingangsanschluss (Eingangspin) selektiert. Die Selektion wird so verschoben, dass der Ausgangsanschluss am Anfang der Verbindungslinie selektiert ist. Bei mehreren Anschlüssen sind alle selektiert. Beispiel: 

Siehe auch:

- Generelle Funktionalitäten in allen grafischen Editoren [▶ 655]

Verbinden

Sie können zwischen Elementanschlüssen Verbindungslinien einfügen. Verbindungslinien werden mit Autorouting eingefügt, sodass sie automatisch optimal und möglichst kurz sind. Die Verbindungslinien werden auf Kollisionen überprüft.

Drag-and-drop von einem Anschluss zu einem anderen	Zwischen den 2 Elementanschlüssen wird eine Verbindungslinie eingefügt.
Drag-and-drop von einem Anschluss zu einem Funktionsbaustein	Loslassen (Drop) kann auf einem Anschluss oder auf dem Textfeld eines Anschlusses durchgeführt werden. Bei erweiterbaren Operatoren (zum Beispiel ADD) kann das Loslassen auch innerhalb des Bausteins erfolgen. Dabei gilt folgendes Verhalten: Wenn es noch unverbundene Eingangsanschlüsse gibt, so wird der oberste freie Anschluss verschaltet. Wenn es keine unverbundenen Eingangsanschlüsse mehr gibt, dann wird automatisch ein neuer Anschluss unten eingefügt.
Befehl **Selektierte Anschüsse verbinden**	Voraussetzung: Sie haben mehrere Anschlüsse selektiert. Die Anschlüsse sind rot gekennzeichnet.
Eingefügtes Element so verschieben, dass es den Anschluss eines anderen Elements berührt.	Voraussetzung: Option **Automatisches Verknüpfen aktivieren** ist aktiviert. Die sich berührenden Anschlüsse werden automatisch miteinander verbunden.
	Das Verbindungsicon befindet sich in der rechten oberen Ecke im Editor. Ein grünes Icon weist auf kollisionsfreie Verbindungen hin. Ein rotes Icon weist auf Kollisionen hin. Mit Klick auf das Icon öffnet sich ein Menü mit Befehlen zur Kollisionsbearbeitung, beispielsweise dem Befehl **Nächste Kollision zeigen**.
	Voraussetzung: Sie haben eine Verbindung selektiert und den Befehl **Verbindungsmarke** gewählt. Statt durch eine lange Verbindungslinie wird eine Verbindung durch Verbindungsmarken dargestellt.

Siehe auch:

- Befehl Selektierte Anschlüsse verbinden [▶ 1064]
- Befehl Nächste Kollision zeigen [▶ 1065]
- Befehl Verbindungsmarke [▶ 1069]

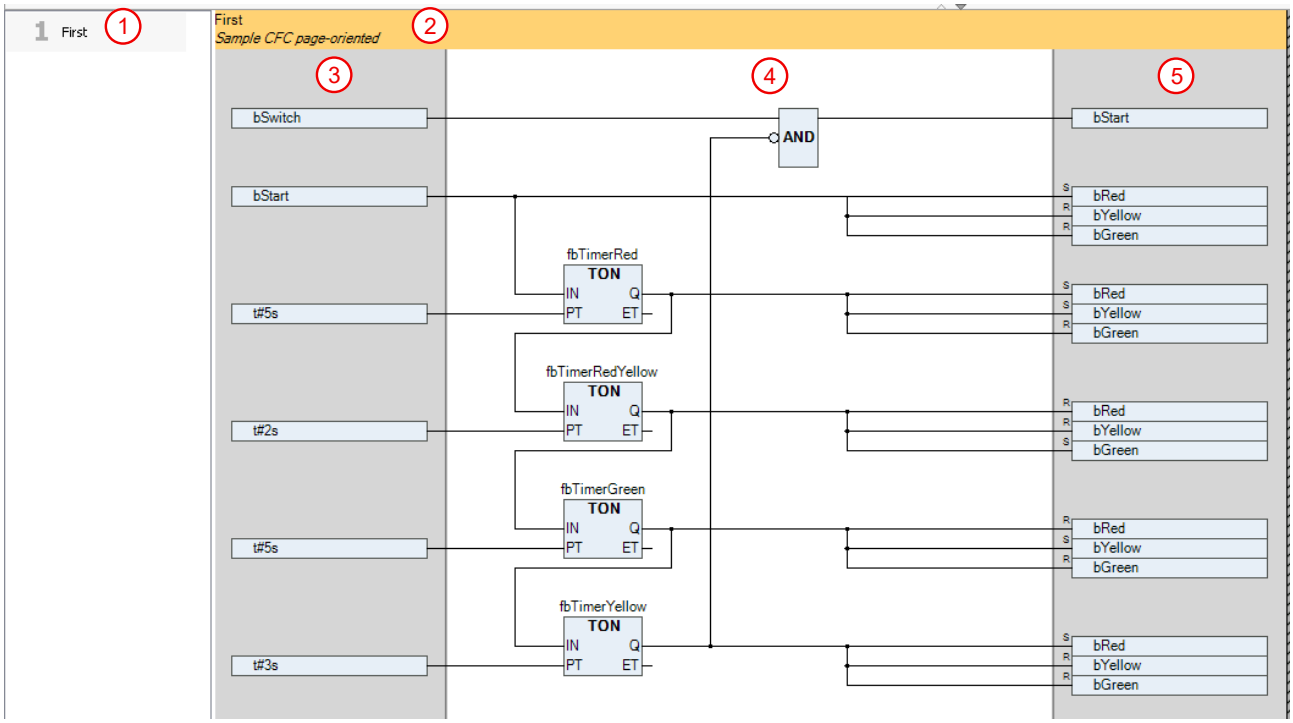
Siehe auch:

- Generelle Funktionalitäten in allen grafischen Editoren [▶ 655]
- Programmieren in CFC [▶ 119]
- Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Elemente [▶ 711]
- Dokumentation TC3 User Interface: CFC [▶ 1057]
- Dokumentation TC3 User Interface: Dialog Optionen - CFC-Editor [▶ 1011]

16.1.6.2 CFC-Editor seitenorientiert

Sie können Programmierbausteine, die in der Implementierungssprache **Continous Function Chart (CFC) - seitenorientiert** erzeugt wurden, nicht in Programmierbausteine der Implementierungssprache **Continous Function Chart (CFC)** umwandeln und umgekehrt.

Sie können die Elemente zwischen diesen beiden Editoren mit dem Befehlen **Kopieren** und **Einfügen** (über Zwischenablage) oder über die Drag-and-drop-Funktion kopieren.



1. Seitennavigator
2. Kopfzeile der Seite mit Seitennamen und Seitenbeschreibung
3. Linker Randbereich reserviert für Eingänge und Zielverbindungsmarken
4. Programmbereich
5. Rechter Randbereich reserviert für Ausgänge und Quellverbindungsmarken

Editieren

Sie können ein Element **Seite** aus dem **Werkzeugkasten** in die Seitennavigation ziehen, dann wird eine weitere Seite eingefügt.

Bestehende Seiten können Sie in der Seitennavigation selektieren und mit den Befehlen **Kopieren** und **Einfügen** vervielfachen.

Die Größe der Seite verändern Sie über den Befehl **Seitengröße bearbeiten**.

Verbindungen über Seiten hinweg erstellen Sie mit Hilfe der Elemente **Verbindungsmarke - Quelle** und **Verbindungsmarke - Ziel**. Wenn Sie eine Verbindungslinie von einem Eingangspin oder einem Ausgangspin auf den Randbereich ziehen, wird automatisch eine neue Verbindungsmarke erzeugt. Die Eingabeunterstützung **Komponenten auflisten** bietet alle bereits definierten Verbindungsmarken-Quellen an.

Wenn Sie im Editor ein Element selektiert haben, können Sie mit Hilfe der Pfeiltasten die Selektion von einem zum nächsten Element umsetzen und so durch die Schaltung navigieren. Wenn Sie eine Verbindungsmarke selektiert haben und noch eine Pfeiltaste drücken, wird die zugehörige Verbindungsmarke der nächsten/vorherigen Seite selektiert.

Sie können Netzwerke oder Elemente aus einem CFC-Programmierbaustein mit den Befehlen **Kopieren** und **Einfügen** in den Programmbereich eines seitenorientierten CFCs übertragen. Alternativ können Sie Drag-and-drop verwenden.

Ausführungsreihenfolge

Die Ausführungsreihenfolge wird automatisch nach der Reihenfolge der Seiten festgelegt, so wie sie im Seitennavigator des Editors sortiert sind. Innerhalb einer Seite verhält sich ein seitenorientiertes CFC-Objekt so wie ein CFC-Objekt. Sie können somit zwischen den Modi **Automatischer Datenfluss-Modus** und **Expliziter Ausführungsreihenfolge-Modus** umschalten.

Zusätzliche Befehle in CFC seitenorientiert

- Befehl Seitengröße bearbeiten [▶ 1058]
- Befehl Arbeitsblatt bearbeiten [▶ 1057]

Siehe auch:

- Generelle Funktionalitäten in allen grafischen Editoren [▶ 655]
- Programmieren in CFC [▶ 119]
- Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Elemente [▶ 711]
- Dokumentation TC3 User Interface: CFC [▶ 1057]
- Dokumentation TC3 User Interface: Dialog Optionen - CFC-Editor [▶ 1011]

16.1.6.3 CFC-Editor im Onlinebetrieb

Im Onlinebetrieb können Sie im CFC-Editor Variablenwerte der Steuerung überwachen und diese auch verändern. Weiterhin können Sie die Debugging-Funktionalitäten von TwinCAT wie Haltepunkte und schrittweises Ausführen nutzen.

Monitoring

Sie können wie gewohnt sowohl im Deklarationsteil als auch im Implementationsteil (mit Inline-Monitoring) Werte beobachten.

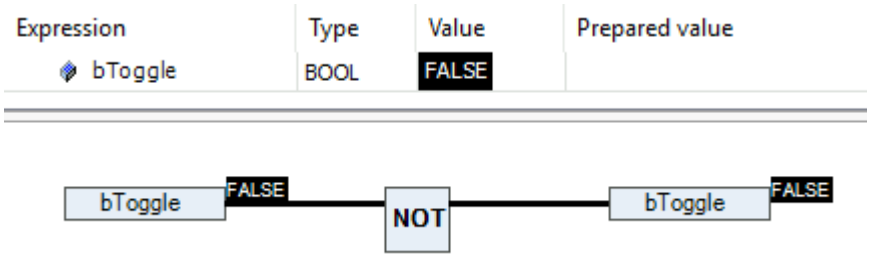
Inline-Monitoring bei einem Funktionsbaustein ist nur möglich, wenn eine Instanz des Funktionsbausteins geöffnet ist. In der Ansicht der Basisimplementierung werden keine Werte angezeigt.

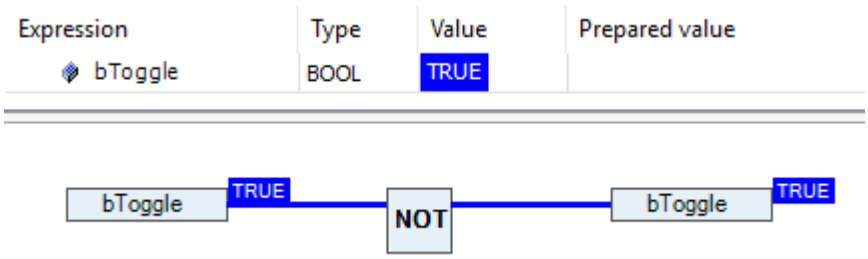
Boolsche Variablen monitoren

Die Verbindungen zwischen booleschen Variablen werden ihrem Istwert entsprechend farblich dargestellt: TRUE in blau und FALSE in schwarz. Die Elementanschlüsse werden mit dem Istwert dekoriert.

Beispiel

Eine Applikation enthält einen CFC-Programmierbaustein. Dort wird eine interne boolesche Variable umgeschaltet: Die Variable `bToggle` wechselt mit jedem Buszyklus ihren Zustand von TRUE auf FALSE.

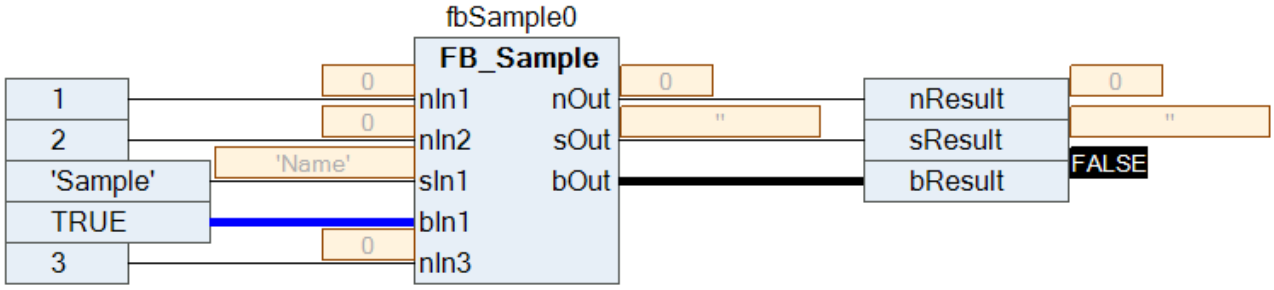




Skalare Variablen monitoren

Bei skalaren Variablen werden die Elementanschlüsse mit den Istwerten dekoriert.

Beispiel



Variablen forcen und schreiben

Im Onlinebetrieb können Sie einen Wert für das Forcen oder Schreiben einer gemonitorten Variablen im Deklarationseditor vorbereiten.

Wenn Sie die Option **Werte im Implementierungsteil vorbereiten** in den TwinCAT-Optionen in der Kategorie **TwinCAT > SPS Programmierumgebung > CFC Editor** aktiviert haben, können Sie auch im Implementierungsteil Werte vorbereiten. Dazu öffnen Sie den Dialog **Wert vorbereiten** durch einen Doppelklick auf die Monitoring-Box neben dem jeweiligen Element oder direkt auf das Element. Für boolesche Variablen erscheint kein Dialog, sondern es wird bei jedem Mausklick auf den neben der Variablen dargestellten Wert zwischen den Werten TRUE und FALSE umgeschaltet.

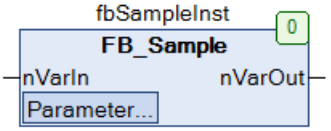
Vorbereitete Werte werden in spitzen Klammern angezeigt. Nach dem Ausführen des Schreibens oder Forcens erscheint in der Monitoring-Box ein rotes „F“.

Konstante Eingangsparameter von Funktionsbausteininstanzen ändern

Sie können Eingangsparameter von Funktionsbausteininstanzen vom Typ `VAR_CONSTANT` auch im Onlinebetrieb verändern und somit die Parameter justieren. Nach dem Ausloggen übernehmen Sie dann diese Parameter mit dem Befehl **Vorbereitete Parameterwerte übernehmen** in Ihr Projekt.

✓ Ein CFC-Editor ist aktiv. Es ist ein Funktionsbaustein instanziiert, der in seiner Deklaration `VAR_INPUT CONSTANT`-Variablen hat.

1. Öffnen Sie die POU mit dem Aufruf der Funktionsbausteininstanz im Editor.
2. Loggen Sie sich auf der Steuerung ein.
3. Klicken Sie in das Feld **Parameter** einer der Funktionsbausteininstanzen.

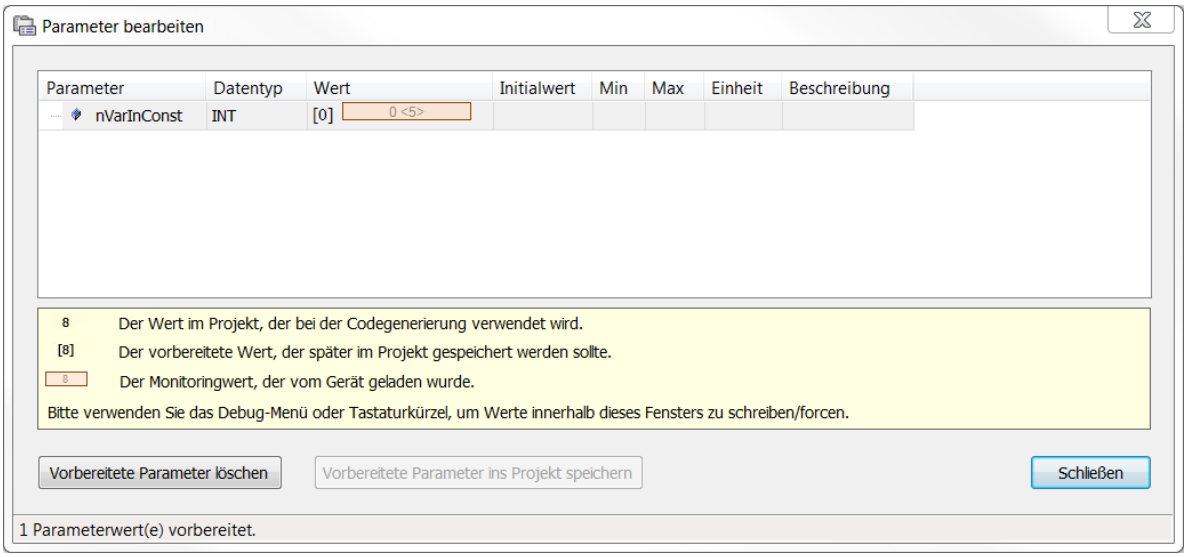


⇨ Der Dialog **Parameter bearbeiten** öffnet sich.

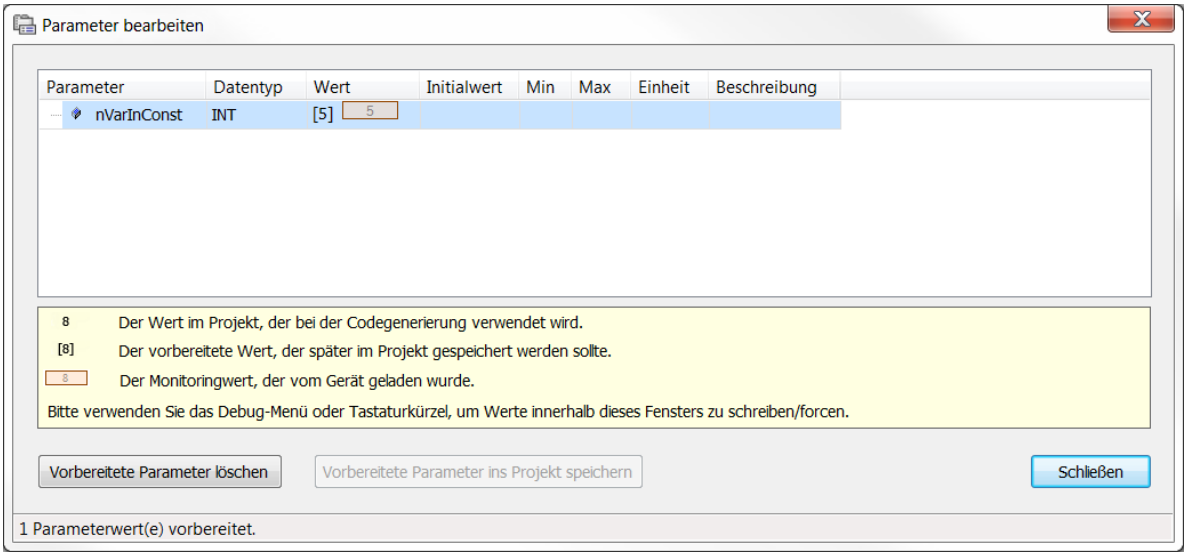
4. Klicken Sie in der Spalte **Wert** in ein Inline-Monitoring-Feld eines Parameters.

⇨ Der Dialog **Wert vorbereiten** öffnet sich.

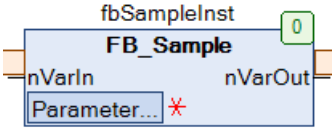
5. Geben Sie einen Wert in das Feld **Einen neuen Wert für die nächste Schreib- oder Force-Operation vorbereiten** ein.
6. Bestätigen Sie die Eingabe mit **OK**.
   ⇨ Der vorbereitete Wert wird neben dem aktuellen Wert in spitzen Klammern dargestellt (im Beispiel <5>)



7. Schreiben Sie den vorbereiteten Wert mit dem Befehl **Werte schreiben** im Menü **PLC** oder in der Symbolleiste.
   ⇨ Der vorbereitete Wert wird geschrieben. Der Parameter ist geändert und wird in eckigen Klammern dargestellt.



   Neben dem Parameterfeld der Funktionsbausteininstanz wird der Unterschied der beiden Werte durch ein rotes Kreuz dargestellt



8. Schließen Sie den Dialog **Parameter bearbeiten**.
9. Loggen Sie sich aus.
10. Klicken Sie in das Feld **Parameter** einer der Funktionsbausteininstanzen oder markieren die Funktionsbausteininstanz und wählen Sie den Befehl **Parameter bearbeiten** im Menü **CFC**.
    ⇨ Der Dialog **Parameter bearbeiten** öffnet sich erneut.

11. Wählen Sie den Befehl **Vorbereitete Parameter im Projekt speichern**.

⇨ Der geänderte Parameterwert wird in das Projekt übernommen. Der Stern neben dem Parameterfeld verschwindet.

Siehe auch:

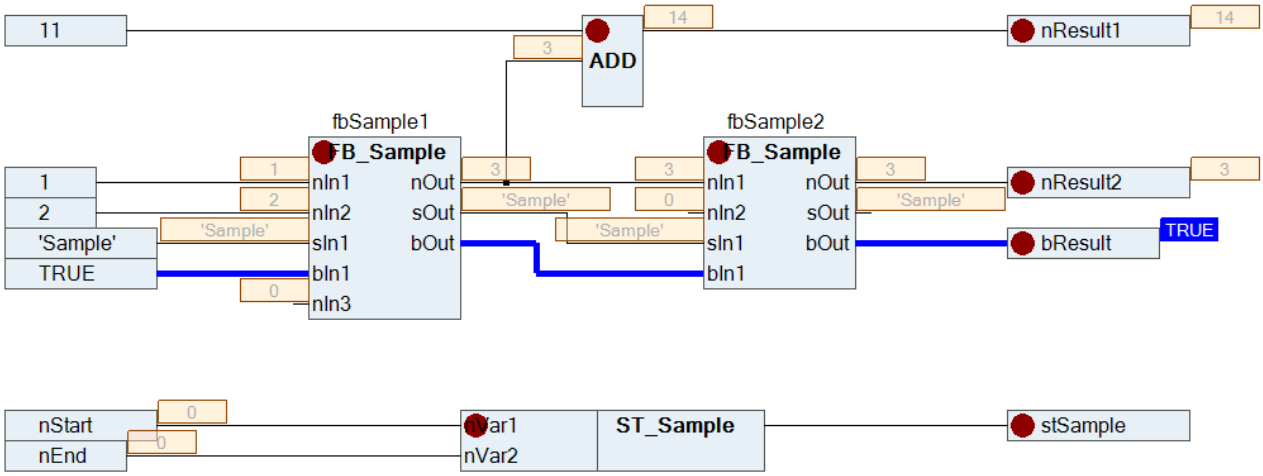
- Dokumentation TC3 User Interface: Befehl Parameter bearbeiten [▶ 1070]
- Dokumentation TC3 User Interface: Befehl Vorbereitete Parameter im Projekt speichern [▶ 1072]

Haltepunkt-Positionen

Mögliche Positionen eines Haltepunkts

- Element **Ausgang**
  Variablen werden beschrieben.
- Element **Baustein**
  Programmierbausteine werden aufgerufen.
- Element **RETURN**
  Der Programmverlauf verzweigt sich.
- Element **Kompositor**
  Strukturelemente werden beschrieben.

Wählen Sie den Befehl **Haltepunkt umschalten** im Menü **Debuggen**, um einen Haltepunkt zu setzen oder einen bestehenden zu löschen. Rote Kreise im Blockschaltbild stellen einen aktivierten Haltepunkt dar.



Ein Haltepunkt wird automatisch in allen Methoden gesetzt, die aufgerufen werden können. Somit gilt: Wenn eine Methode aufgerufen wird, die über eine Schnittstelle definiert ist, werden in allen Methoden von Funktionsbausteinen, die diese Schnittstelle implementieren, Haltepunkte gesetzt. Dies gilt ebenfalls in allen abgeleiteten Funktionsbausteinen, die die Methode definieren.

Programmierbaustein schrittweise ausführen

Sie können einen Programmierbaustein im Debug-Betrieb schrittweise abarbeiten. Ein Programmierbaustein, der aufgerufen wird, wird intern um einen RETURN am Anfang vor dem Element mit der Nummer 0 und am Ende nach dem letzten Element ergänzt. Bei der schrittweisen Abarbeitung werden diese automatisch angesprungen.

Siehe auch:

- Monitoring von Werten [▶ 234]
- Forcen und Schreiben von Variablen [▶ 225]
- Haltepunkte verwenden [▶ 222]
- Schrittweises Abarbeiten eines Programms (Stepping) [▶ 224]

Sehen Sie dazu auch

- Befehl Parameter bearbeiten [▶ 1070]
- Befehl Vorbereitete Parameter im Projekt speichern [▶ 1072]

16.1.6.4 Tastaturkürzel für die CFC-Editoren festlegen

In den **Optionen** können Sie unter **Umgebung > Tastatur** für viele Befehle im CFC-Editor individuelle Tastaturkürzel festlegen, die Ihnen das Bearbeiten vereinfachen:

Befehl	Kommando für Tastaturkürzel
Alles selektieren	CFC.SelectAll
Elemente einfügen:	
Baustein einfügen. Zur Auswahl des Bausteins öffnet sich der Dialog **Eingabehilfe**.	CFC.InsertBoxFromInputAssistant
Leeren Baustein einfügen.	CFC.InsertBox
Baustein mit EN/ENO einfügen. Zur Auswahl des Bausteins öffnet sich der Dialog **Eingabehilfe**.	CFC.InsertBoxWithENENOFromInputAssistant
Eingang einfügen. Fügt ein Eingangselement ein.	CFC.InsertInput
Ausgang einfügen. Fügt ein Ausgangselement ein.	CFC.InsertOutput
Sprung einfügen.	CFC.InsertJump
Bereits eingefügte Elemente bearbeiten:	
Negieren	CFC.Negate
Umschalten zwischen Set, Reset, REF und Kein.	CFC.SwitchSRRefNone
Anschlüsse zurücksetzen.	CFC.ResetPins

Siehe auch

- Dokumentation TC3 User Interface: CFC [▶ 1057]
- Dokumentation TC3 User Interface: Tastaturkürzel anpassen

16.1.6.5 Elemente

16.1.6.5.1 CFC-Element Seite

Symbol: 

Das Element fügt eine neue Seite in den Editor ein. Es ist nur im seitenorientierten CFC-Editor verfügbar. Die Nummer der Seite wird automatisch nach ihrer Position vergeben. Sie können den Namen und die Beschreibung der Seite in die orangene Kopfzeile eingeben. Die Seitengröße passen Sie mit dem Befehl **Seitengröße bearbeiten** an.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Seitengröße bearbeiten [▶ 1058]

16.1.6.5.2 CFC-Element Kontrollpunkt

Symbol: 

Verwenden Sie einen Kontrollpunkt, um Punkte einer Verbindung zu fixieren, bevor Sie die Linienführung anpassen. Dazu ziehen Sie das Element an die gewünschte Position auf einer Verbindungslinie. Verbindungslinien mit Kontrollpunkten werden nicht mehr automatisch geroutet.

Siehe auch:

- Programmieren in CFC [▶ 119]
- Dokumentation TC3 User Interface: Befehl Kontrollpunkt erzeugen [▶ 1068]
- Dokumentation TC3 User Interface: Befehl Kontrollpunkt entfernen [▶ 1068]

16.1.6.5.3 CFC-Element Eingang

Symbol:

TwinCAT fügt ein Eingangselement standardmäßig mit dem Text **???** ein. Sie können dieses Feld durch Anklicken direkt editieren und einen konstanten Wert oder einen Variablennamen eingeben. Alternativ können Sie durch Anklicken von  den Eingabeassistenten öffnen um eine Variable auszuwählen.

16.1.6.5.4 CFC-Element Ausgang

Symbol:

TwinCAT fügt ein Ausgangselement standardmäßig mit dem Text **???** ein. Sie können dieses Feld durch Anklicken direkt editieren und einen konstanten Wert oder einen Variablennamen eingeben. Alternativ können Sie durch Anklicken von  den Eingabeassistenten öffnen um eine Variable auszuwählen.

16.1.6.5.5 CFC-Element Baustein

Symbol: 

Sie verwenden das Element, um einen Operator, eine Funktion, einen Funktionsbaustein oder ein Programm einzufügen. TwinCAT fügt das Element standardmäßig mit dem Namen **???** ein. Sie können dieses Feld durch Anklicken direkt editieren und einen Bausteinnamen eingeben. Alternativ können Sie durch Anklicken von  den Eingabeassistenten öffnen und einen Baustein auswählen.

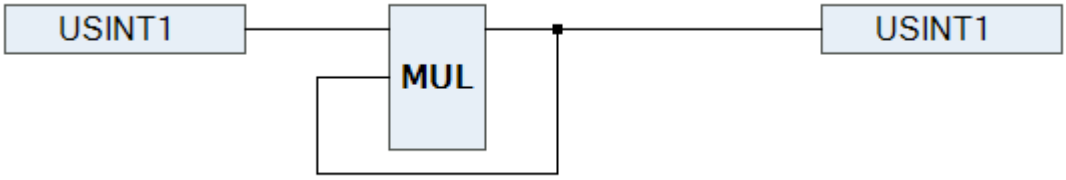
Im Fall eines Funktionsbausteins zeigt TwinCAT zusätzlich oberhalb des Bausteinsymbols ein Eingabefeld an **(???).** Diesen Namen müssen Sie durch den Namen der Funktionsbaustein-Instanz ersetzen. Wenn Sie einen Funktionsbaustein mit konstanten Eingangsparametern instanzieren, zeigt das Bausteinelement in der linken unteren Ecke das Feld **Parameter...** an. Die Parameter bearbeiten Sie durch Anklicken dieses Felds.

Um eine existierende Box zu ersetzen, ersetzen Sie nur den aktuell eingefügten Bezeichner mit dem gewünschten neuen Namen. Beachten Sie dabei, dass TwinCAT die Anzahl an Ein- und Ausgangspins entsprechend der Definition der POU anpasst und deshalb möglicherweise existierende Zuweisungen gelöscht werden.

Da im CFC Rückführungen erlaubt sind, werden am Ausgang eines Bausteins implizite Variablen mit Datentyp der Eingangsvariablen erzeugt (im Beispiel `temp_USINT`). Wenn das Ergebnis der Operation eines Bausteins ein Wert ist, der den Zahlenbereich des Datentyps der Eingangsvariablen überschreitet, wird der Überlauf in die implizite Variable geschrieben. Die eigentliche Ausgangsvariable erhält den Wert der impliziten Variablen, also den Überlauf und nicht das eigentliche Ergebnis der Operation (siehe Beispiel).

Beispiel

Implizit erzeugte Variablen am Bausteinausgang:



Implizit generierter Code:

```
temp_USINT := USINT1 * temp_USINT;
UDINT1     := temp_USINT;
```

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Parameter bearbeiten [▶ 1070]

16.1.6.5.6 CFC-Element Sprung

Symbol:

Sie verwenden das Element, um eine Position festzulegen, an der die Programmabarbeitung fortsetzen soll. Diese Zielposition müssen Sie durch eine Marke definieren. Dazu geben Sie den Namen der Marke in das Eingabefeld **???** ein. Falls Sie die entsprechende Marke bereits eingefügt haben, können Sie diese auch über den Eingabeassistenten (...) auswählen.

Siehe auch:

- CFC-Element Marke [▶ 713]

16.1.6.5.7 CFC-Element Marke

Symbol:

Eine Marke definiert eine Position, zu der die Programmabarbeitung mithilfe eines Sprung-Elements springt.

Im Onlinebetrieb fügt TwinCAT automatisch eine RETURN-Marke am Ende eines CFC-Bausteins ein.

Siehe auch:

- CFC-Element Sprung [▶ 713]

16.1.6.5.8 CFC-Element Return

Symbol:

Verwenden Sie das Element, um den Baustein zu verlassen.

Beachten Sie, dass im Onlinebetrieb im CFC-Editor ein Return-Element automatisch vor der ersten Zeile und nach dem letzten Element eingefügt wird. Bei der schrittweisen Abarbeitung springt TwinCAT das Return-Element am Ende automatisch an, bevor der Baustein verlassen wird.

16.1.6.5.9 CFC-Element Kompositor

Symbol:

Das Komposer-Element dient der Handhabung von Strukturkomponenten. Dabei werden Ihnen die einzelnen Komponenten einer Struktur als Eingang zur Verfügung gestellt. Zu diesem Zweck müssen Sie das Kompositorelement wie die betreffende Struktur nennen (Ersetzen der **???**).

Das Kompositorelement ist das Gegenstück zum Selektorelement.

Siehe auch:

- CFC-Element Selektor [▶ 714]

16.1.6.5.10 CFC-Element Selektor

Symbol: 

Das Selektorelement dient der Handhabung von Strukturkomponenten. Dabei werden Ihnen die einzelnen Komponenten einer Struktur als Ausgang zur Verfügung gestellt. Zu diesem Zweck müssen Sie das Selektorelement wie die betreffende Struktur nennen (Ersetzen der **???**).

Das Selektorelement ist das Gegenstück zum Kompositorelement.

Siehe auch:

- CFC-Element Kompositor [▶ 713]

16.1.6.5.11 CFC-Element Kommentar

Symbol:

Mit diesem Element geben Sie einen Kommentar im CFC-Editor ein. Ersetzen Sie den Platzhaltertext im Element durch den Kommentartext. Ein Zeilenumbruch kann mithilfe der Tastenkombination **[Strg] + [Eingabe]** eingefügt werden.

16.1.6.5.12 CFC-Element Verbindungsmarke-Quelle/Ziel

Symbol: 

Sie können Verbindungsmarken anstelle einer Verbindungslinie zwischen Elementen verwenden. Das hilft Ihnen, komplexe Diagramme übersichtlicher darzustellen.

Für eine gültige Verbindung müssen Sie ein Element **Verbindungsmarke - Quelle** mit dem Ausgang eines Elements und ein Element **Verbindungsmarke - Ziel** mit dem Eingang eines anderen Elements verbinden. Beide Marken müssen den gleichen Namen tragen. Dabei wird die Groß-/Kleinschreibung nicht berücksichtigt.

Hinweise zur Namensgebung:

- Der Standardnamen für Verbindungsmarken ist C-<nr>. Dabei ist <nr> eine fortlaufende Nummer, beginnend mit 1.
- Sie können den Standardnamen umbenennen. Dabei müssen Sie beachten, dass die Quellmarke und die Zielmarke den gleichen Namen haben.
- Wenn Sie den Namen der Quellmarke verändern, wird der Zielname automatisch umbenannt.
- Wenn Sie den Namen der Zielmarke verändern, bleibt der Quellname erhalten.

Beachten Sie den Befehl Verbindungsmarke zur automatischen Umwandlung einer bestehenden Verbindung.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Verbindungsmarke [▶ 1069]

16.1.6.5.13 CFC-Element Bausteineingang

Symbol: 

Abhängig vom Bausteintyp können Sie einem eingefügten Bausteinelement weitere Eingänge hinzufügen. Dazu müssen Sie das Bausteinelement selektieren und das Bausteineingangelement auf den Bausteinrumpf ziehen.

Sie können eine Eingangs- oder Ausgangsverbindung bei gedrückt gehaltener Taste **[Strg]** auf eine andere Position am Baustein ziehen.

Siehe auch:

- CFC-Element Bausteinausgang [▶ 715]

16.1.6.5.14 CFC-Element Bausteinausgang

Symbol: 

Abhängig vom Bausteintyp können Sie einem eingefügten Bausteinelement weitere Ausgänge hinzufügen. Dazu müssen Sie das Bausteinelement selektieren und das Bausteinausgangselement auf den Bausteinrumpf ziehen.

Sie können eine Eingangs- oder Ausgangsverbindung bei gedrückt gehaltener Taste **[Strg]** auf eine andere Position am Baustein ziehen.

Siehe auch:

- CFC-Element Bausteineingang [▶ 715]

16.1.7 Ladder Diagram (LD) (Beta)

16.1.7.1 Ladder-Editor

Der neue Ladder-Editor ist in einer Beta-Version verfügbar ab TC3.1 Build 4026.

Der Ladder-Editor ist ein netzwerkbasierter Editor für die IEC-Programmiersprache Kontaktplan (KOP). Im Implementierungsteil programmieren Sie in einem oder mehreren Netzwerk-Elementen einen „Stromlaufplan“. Dazu können Sie die benötigten Elemente aus der Werkzeugbox oder über Menübefehle einfügen, verknüpfen und mit Ein- und Ausgangsvariablen und Modifizierern versehen.

Jedes Netzwerk kann mit einem Titel, einem Kommentar und/oder einer Sprungmarke versehen werden. Sie können Netzwerke auskommentieren.

Das Einfügen und Ersetzen von Elementen erfolgt durch das Ziehen eines Elements mit der Maus aus der Werkzeugbox auf eine der angebotenen Einfügepositionen.

Die Einfügepositionen werden mit folgenden Symbolen angezeigt, während Sie das Element über den Implementierungsteil ziehen:

- grau hinterlegtes Quadrat innerhalb eines bereits vorhandenen Elementsymbols
- Raute auf einer Verbindungslinie
- Dreieck, nach unten oder oben gerichtet, für ein Einfügen oberhalb oder unterhalb

Eine mögliche Position "leuchtet auf", der Mauszeiger erhält ein Pluszeichen  , und beim Loslassen der Maustaste wird das Element eingefügt.

Aktuell selektierte Bereiche im Editor sind rot hinterlegt und umrandet.

Passende Befehle zum Modifizieren (Flankenerkennung, Set/Reset, EN/ENO), Löschen, Refactoring, Suchen etc.) finden Sie jeweils im Kontextmenü.

Im Onlinebetrieb [▶ 716] sind Monitoring und Fehlersuche mit Hilfe von Breakpoints und dem Schreiben und Forcen von Werten möglich.

Die Darstellung im Editor wird in den TwinCAT-Optionen, Kategorie Ladder [▶ 1028]-Editor festgelegt. Dies betrifft beispielsweise die Anzeige von Kommentaren, Adressen, oder ob Netzwerke mit Zeilenumbrüchen versehen werden.

Programme, die im FUP/KOP-Editor erstellt wurden, können in den Ladder-Editor eingelesen und weiterbearbeitet werden.

Siehe auch:

- Programmieren im Ladder-Editor [▶ 133]
- Dokumentation Benutzeroberfläche: Ladder-Editor [▶ 1085]
- Dokumentation Benutzeroberfläche: Dialog Optionen - Ladder-Editor [▶ 1028]

16.1.7.2 Ladder-Editor im Onlinebetrieb

Im Onlinebetrieb bietet der Editor ein Werte-Monitoring und unterstützt das Schreiben und Forcen der aktuellen Werte. Sie können Haltepunkte setzen und die farbliche Darstellung der Verbindungen ermöglicht eine Art Ablaufkontrolle.

Monitoring

Im Onlinebetrieb wird im Editor bei jeder Variablen ihr Istwert dargestellt. Konstante Variablen erhalten ein grünes C-Symbol. Die Darstellung der Werte ist in den TwinCAT-Optionen, Kategorie Ladder-Editor [▶ 1028] festgelegt.

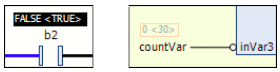
Schreiben/Forcen von Werten

Wenn eine Variable gerade geforct ist, wird dies durch F direkt vor dem geforcten Wert angezeigt. Wenn ein Wert für das Schreiben oder Forcen vorbereitet ist, wird dieser Wert direkt hinter dem Istwert in spitzen Klammern angezeigt: <*value*>.

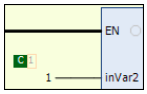
Beispiele:

Geforcte Variable:

Vorbereiteter Wert:



Konstanter Wert:

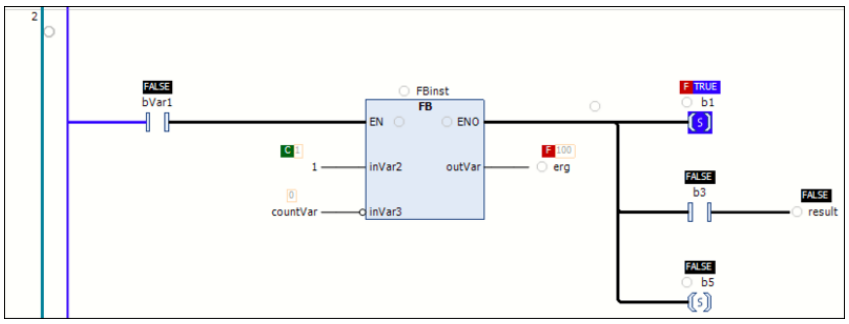


Farbliche Darstellung von Verbindungen

In der Onlineansicht eines Ladder-Diagramms sind die Anschlussleitungen farblich gekennzeichnet: Verbindungen mit dem Wert *TRUE* werden als fette blaue Linie dargestellt, Verbindungen mit dem Wert *FALSE* als fette schwarze Linie. Dagegen werden Verbindungen von unbekanntem oder analogem Wert normal dargestellt werden (dünne schwarze Linie).

Der Wert der Verbindungen wird nicht aus den gemonitorten Variablen abgelesen, sondern im Programmiersystem berechnet. Es handelt sich dabei nicht um eine echte Ablaufkontrolle.

Beispiel: Verbindungslinien und Haltepunktpositionen



Haltepunkte

Haltepunkte sind grundsätzlich an Positionen möglich, an denen sich Variablenwerte ändern können (Anweisungen), an denen sich das Programm verzweigt, oder an denen ein anderer Programmierbaustein aufgerufen wird.

Im Editor werden mögliche Haltepunktpositionen mit einem leeren grauen Kreissymbol angezeigt. Gesetzte Haltepunkte erscheinen als rot gefüllter Kreis. Sehen Sie dazu oben die Abbildung "Beispiel: Verbindungslinien und Haltepunktpositionen".

Mögliche Haltepunktpositionen:

- Auf einem aufrufbaren Baustein (Funktionsbaustein, Funktion, Programm, Aktion, Methode). Nicht möglich bei Operatoren-Bausteinen, beispielsweise *ADD, DIV*
- Auf Zuweisungen
- Vor Parallelverzweigungen
- Am Ende des Bausteins an der Position der Rückkehr zum aufrufenden Baustein. Im Onlinebetrieb erscheint automatisch ein leeres Netzwerk an dieser Stelle, das anstelle einer Netzwerknummer mit **RET** gekennzeichnet ist.
- Auf **EN**-Eingang und **ENO**-Ausgang eines Bausteins
- Auf dem gesamten Netzwerk. Zeigt nur an, dass im Netzwerk ein Haltepunkt gesetzt ist. Es kann kein Haltepunkt auf das gesamte Netzwerk gesetzt werden.

Haltepunkte in Methoden: TwinCAT setzt automatisch einen Haltepunkt in allen Methoden, die aufgerufen werden können. Wenn also eine durch eine Schnittstelle verwaltete Methode aufgerufen wird, werden Haltepunkte in allen Methoden gesetzt, die in Funktionsbausteinen vorkommen, die diese Schnittstelle implementieren; ebenso in allen abgeleiteten Funktionsbausteinen, die die Methode verwenden. Wenn eine Methode durch einen Pointer auf einen Funktionsbaustein aufgerufen wird, setzt TwinCAT die Haltepunkte in der Methode des Funktionsbausteins und in allen abgeleiteten Funktionsbausteinen, die die Methode verwenden.

16.1.7.3 Leitungsverzweigung

Eine Leitungsverzweigung spaltet die Abarbeitungslinie in 2 oder mehrere Zweige, die von oben nach unten nacheinander ausgeführt werden. Jeden Zweig können Sie weiter verzweigen, wodurch innerhalb eines Netzwerks Mehrfachverzweigungen entstehen.

Jeder Zweig erhält am Verzweigungspunkt ein Markersymbol (kleines Rechteck), das Sie auswählen können, um weitere Aktionen/Befehle für das Subnetz ausführen zu können.

Sehen Sie für weitere Informationen: Programmieren im Ladder-Editor [▶ 133]

16.1.7.4 Navigation im Ladder-Editor

Mit Hilfe der im Folgenden beschriebenen Tasten(-kombinationen) können Sie im Editor zwischen Cursorpositionen wechseln. Der Wechsel funktioniert auch netzwerkübergreifend.

→ ←	Wechsel zur benachbarten Cursorposition, entlang des Signalflusses, also von links nach rechts und umgekehrt.
↑ ↓	Wechsel zur nächsten Cursorposition oberhalb oder unterhalb der aktuellen Position, wenn diese Nachbarposition zur gleichen logischen Gruppe gehört. Eine logische Gruppe bilden beispielsweise alle Anschlüsse eines Bausteins. Wenn eine solche logische Gruppe nicht existiert: Wechsel zur ersten Cursorposition im nächsten oberen oder unteren Nachbarelement. Im Fall von parallel verbundenen Elementen erfolgt die Navigation entlang des ersten Zweiges.
Aktuell noch nicht implementiert. Strg + Pos1	Wechsel ins erste Netzwerk; dieses wird selektiert.
Strg + Ende	Wechsel ins letzte Netzwerk; dieses wird selektiert.
Bild↑	Um eine Seite nach oben blättern. Das oberste Netzwerk auf dieser Seite wird selektiert.
Bild↓	Um eine Seite nach unten blättern. Das unterste Netzwerk auf dieser Seite wird selektiert.

16.1.7.5 Elemente

Die Elemente, die im Ladder-Editor bereitstehen, werden auf den Hilfeseiten zu den Menübefehlen mit beschrieben, die für das Einfügen der einzelnen Elemente bereitstehen.

16.2 Variablen

Der Gültigkeitsbereich einer Variablen definiert, wie und wo Sie die Variable verwenden können. Sie legen den Gültigkeitsbereich bei der Variablendeklaration fest.

16.2.1 Lokale Variablen - VAR

Die Deklaration lokaler Variablen erfolgt im Deklarationsteil von Programmierobjekten zwischen den Schlüsselwörtern VAR und END_VAR.

Sie können lokale Variablen mit einem Attribut-Schlüsselwort erweitern.

Über den Instanzpfad können Sie von außerhalb des Programmierobjekts lesend auf lokale Variablen zugreifen. Ein Schreibzugriff von außerhalb des Programmierobjekts ist nicht möglich, da dies vom Compiler als Fehler ausgegeben wird.

Um die vorgesehene Datenkapselung zu wahren, wird dringend empfohlen, auf lokale Variablen einer POU nicht von außerhalb der POU zuzugreifen – weder lesend noch schreibend. (Andere Hochsprachen-Compiler geben auch lesende Zugriffe auf lokale Variablen als Fehler aus.) Bei Bibliotheksbausteinen ist zudem nicht sichergestellt, dass die lokalen Variablen eines Funktionsbausteins bei späteren Updates unverändert bleiben. Dadurch könnte das Applikationsprojekt nach dem Bibliotheksupdate nicht mehr fehlerfrei übersetzt werden.

Beachten Sie hierzu auch die Regel SA0102 vom Static Analysis, die lesende Zugriffe von außen auf lokale Variablen ermittelt.

Beispiel:

```
VAR
    nVar1 : INT;
END_VAR
```

Siehe auch:

- Dokumentation TE1200 | PLC Static Analysis: Konfiguration > Regeln
- Variablentypen - Attribut-Schlüsselwörter [▶ 731]

16.2.2 Eingabevariablen - VAR_INPUT

Eingabevariablen sind Eingangsvariablen für einen Baustein.

VAR_INPUT-Variablen deklarieren Sie im Deklarationsteil von Programmierobjekten zwischen den Schlüsselwörtern VAR_INPUT und END_VAR.

Sie können Eingabevariablen mit einem Attribut-Schlüsselwort erweitern.

Beispiel:

```
VAR_INPUT
    nIn1 : INT; //1st input variable
END_VAR
```

Siehe auch:

- Variablentypen - Attribut-Schlüsselwörter [▶ 731]

16.2.3 Ausgabevariablen - VAR_OUTPUT

Ausgabevariablen sind Ausgangsvariablen eines Bausteins.

VAR_OUTPUT-Variablen deklarieren Sie im Deklarationsteil von Programmierobjekten zwischen den Schlüsselwörtern VAR_OUTPUT und END_VAR. TwinCAT liefert die Werte dieser Variablen an den aufrufenden Baustein zurück. Dort können Sie die Werte abfragen und weiter verwenden.

Sie können Ausgabevariablen mit einem Attribut-Schlüsselwort erweitern.

Beispiel:

```
VAR_OUTPUT
  nOut1 : INT; //1st output variable
END_VAR
```

Ausgabevariablen in Funktionen und Methoden

Gemäß der Norm IEC 61131-3 können Funktionen (und Methoden) zusätzliche Ausgänge haben. Die zusätzlichen Ausgänge müssen Sie beim Aufruf der Funktion wie im folgenden Beispiel zuweisen.

Beispiel:

```
F_Fun(nIn1 := 1, nIn2 := 2, nOut1 => nLoc1, nOut2 => nLoc2);
```

Siehe auch:

- Variablentypen - Attribut-Schlüsselwörter [▶ 731]

16.2.4 Eingabe-/Ausgabevariablen - VAR_IN_OUT, VAR_IN_OUT CONSTANT

Eine VAR_IN_OUT-Variable ist eine Ein- und Ausgabevariable, die Teil einer Bausteinschnittstelle ist und als formaler Durchgangsparameter dient.

Die VAR_IN_OUT-Variablen eines Bausteins müssen bei dem Aufruf des Bausteins zugewiesen werden.

Syntax:

```
<keyword> <POU name>
VAR_IN_OUT
    <variable name> : <data type> ( := <initialization value> )? ;
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

Sie können eine Ein- und Ausgabevariable in den Programmierbausteinen PRG, FUNCTION_BLOCK, METHOD oder FUNCTION im Deklarationsabschnitt VAR_IN_OUT deklarieren. Als Initialisierungswert kann optional eine Konstante des deklarierten Datentyps zugewiesen werden. Die VAR_IN_OUT-Variable kann gelesen und beschrieben werden.

Verwendung:

- **Aufruf:** Bei Aufruf des Programmierbausteins erhält die formale VAR_IN_OUT-Variable als Argument die tatsächliche Variable, die so genannte „Durchgangsvariable“. Zur Laufzeit bei der Parameterübergabe wird dann keine Kopie erzeugt, sondern die formale Variable erhält eine Referenz auf die von außen übergebene tatsächliche Variable. Die referenziellen Variablen enthalten intern als Wert eine Speicheradresse auf den eigentlichen Wert (Übergabe als Pointer, Call-by-Reference). Es ist nicht möglich, als Argument direkt eine Konstante (Literal), eine konstante Variable oder eine Bitvariable anzugeben.
- **Schreib-/Lesezugriff innerhalb des Programmierbausteins:** Wenn innerhalb des Programmierbausteins auf die Variable geschrieben wird, wirkt dies auf die übergebene Variable durch. Wenn der Programmierbaustein verlassen wird, bleiben vorgenommene Änderungen somit erhalten. Das bedeutet, dass ein Programmierbaustein seine VAR_IN_OUT-Variablen so verwendet, wie der aufrufende Programmierbaustein seine Variablen verwendet. Ein Lesezugriff ist immer erlaubt.
- **Schreib-/Lesezugriff von außen:** VAR_IN_OUT-Variablen können nicht via <function block instance name>.<variable name> direkt von außen gelesen oder beschrieben werden. Dies funktioniert nur bei VAR_INPUT- und VAR_OUTPUT-Variablen.
- **Übergabe von Stringvariablen:** Wenn als Argument eine Stringvariable übergeben wird, sollten die tatsächliche Variable und die formale Variable die gleiche Länge haben. Andernfalls kann der übergebene String unbeabsichtigt manipuliert werden. Bei VAR_OUTPUT CONSTANT-Parametern tritt dieses Problem nicht auf.
- **Übergabe von Bitvariablen:** Eine Bitvariable kann nicht direkt an eine VAR_IN_OUT-Variable übergeben werden, sondern benötigt eine Zwischenvariable.
- **Übergabe von Properties:** Ist nicht erlaubt.

Übergabe von Zeichenketten an VAR_IN_OUT CONSTANT

Wenn eine Zeichenkette als Variable oder auch als Konstante an eine formale VAR_IN_OUT CONSTANT-Variable übergeben wird, kann die Stringlänge variabel sein. Es wird aber maximal die Stringlänge der VAR_IN_OUT CONSTANT-Variablen verarbeitet. Weitere Informationen hierzu erhalten Sie unten auf dieser Seite.

Übergabe von Eigenschaften nicht möglich

An eine VAR_IN_OUT- oder eine VAR_IN_OUT CONSTANT-Variable kann keine Eigenschaft (Property) übergeben werden.

Ausnahme: Eine Zuweisung auf VAR_IN_OUT (CONSTANT) ist möglich, falls die Eigenschaft den Rückgabetyp REFERENCE besitzt. Zu beachten ist hierbei allerdings, dass die Eigenschaft per REFERENCE nur eine Adresse zurückliefern darf, die über den Eigenschaftenaufruf hinaus gültig ist. Dies wäre z.B. nicht der Fall, wenn die Eigenschaft die Referenz auf eine temporäre Variable zurückliefern würde.

Beispiel:

Funktionsbaustein FB_Sample

```
FUNCTION_BLOCK FB_Sample
VAR_IN_OUT
    bInOut    : BOOL;
END_VAR
```

Programm MAIN:

```
VAR
    bTest     : BOOL;
    fbSample  : FB_Sample;
END_VAR
```

```
fbSample(bInOut := bTest);  // OK
fbSample();                 // NOK: not possible as the VAR_IN_OUT variable is not assigned in this
FB call
fbSample.bInOut := bTest;   // NOK: direct access to VAR_IN_OUT variable from the outside not
possible
```

Beispiel für einen Workaround für das Zuweisen einer Bitvariablen auf einen VAR_IN_OUT-Eingang:

Deklaration der Bitvariablen (bBit0):

```
VAR_GLOBAL
    bBit0 AT %MX0.1 : BOOL;
    bTemp           : BOOL;
END_VAR
```

Funktionsbaustein mit VAR_IN_OUT Eingang bInOut:

```
FUNCTION_BLOCK FB_Test
VAR_INPUT
    bIn    : BOOL;
END_VAR
VAR_IN_OUT
    bInOut : BOOL;
END_VAR
```

```
IF bIn THEN
    bInOut := TRUE;
END_IF
```

Programm, das den Funktionsbaustein aufruft. Direktes Zuweisen der Bitvariablen auf den VAR_IN_OUT Eingang (Fehler) und Zuweisen mithilfe einer Zwischenvariablen (Workaround):

```
PROGRAM MAIN
VAR
    bIn      : BOOL;
    fbTest1  : FB_Test;
    fbTest2  : FB_Test;
END_VAR
```

```
// Error C0201: Type 'BIT' doesn't correspond to the type 'BOOL' of VAR_IN_OUT 'bInOut'
fbTest1(bIn := bIn, bInOut := bBit0);

// Workaround
//bTemp := bBit0;
//fbTest2(bIn := bIn, bInOut := bTemp);
//bBit0 := bTemp;
```

Übergabevariable VAR_IN_OUT CONSTANT

Eine VAR_IN_OUT CONSTANT-Variable dient als konstanter Durchgangsparameter, der gelesen, aber nicht beschrieben werden kann.

Syntax:

```
<keyword> <POU name>
VAR_IN_OUT CONSTANT
    <variable name> : <data type>; // formal parameter
END_VAR
<keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG
```

VAR_IN_OUT CONSTANT-Variablen werden deklariert, ohne einen Initialisierungswert zuzuweisen.

Verwendung:

- Es sind alle Datentypen erlaubt.
- Schreibzugriffe auf die VAR_IN_OUT CONSTANT-Variable sind nicht erlaubt.
- Die Übergabe von Properties ist nicht erlaubt.
- Wenn die VAR_IN_OUT CONSTANT-Variable vom Typ STRING/WSTRING ist, kann bei Aufruf des Programmierbausteins eine Variable, eine konstante Variable oder eine Konstante (Literal) übergeben werden. Die Stringlänge der übergebenen Variablen/Konstanten kann beliebig sein und hängt nicht von der Stringlänge der VAR_IN_OUT CONSTANT-Variablen ab. Beachten Sie, dass maximal die Stringlänge der VAR_IN_OUT CONSTANT-Variablen verarbeitet wird.
- Wenn die VAR_IN_OUT CONSTANT-Variable nicht vom Typ STRING/WSTRING ist, kann bei Aufruf des Programmierbausteins eine Variable übergeben werden. Wenn die Compileroption **Konstanten ersetzen** in der Kategorie **Übersetzen** in den SPS-Projekteigenschaften deaktiviert ist, kann außerdem eine konstante Variable übergeben werden.

Wenn die Compileroption **Konstanten ersetzen** in der Kategorie **Übersetzen** in den SPS-Projekteigenschaften aktiviert ist, dann erzeugt die Parameterübergabe einer Konstanten mit Basisdatentyp ungleich STRING oder einer konstanten Variablen mit Basisdatentyp ungleich STRING einen Compilerfehler.

Beispiel:

Im Code werden Strings über verschiedene VAR_IN_OUT-Variablen an die Funktion F_Manipulate übergeben. Bei Übergabe eines Literals an eine VAR_IN_OUT-Variable wird ein Compilerfehler ausgegeben. Bei Übergabe eines Literals an eine VAR_IN_OUT CONSTANT-Variable wird korrekter Code erzeugt, ebenso bei der Übergabe von Stringvariablen.

Des Weiteren ist zu erkennen, dass die Übergabe einer zu kurzen STRING-Variablen an eine VAR_IN_OUT-Variable nicht möglich ist (Compiler-Fehler), bei der Übergabe an eine VAR_IN_OUT CONSTANT-Variable ist dies hingegen möglich.

Funktion F_Manipulate:

```
FUNCTION F_Manipulate : BOOL
VAR_IN_OUT
    sReadWrite   : STRING(16);  (* Can be read or written here in POU *)
    nReadWrite   : DWORD;       (* Can be read or written here in POU *)
END_VAR
VAR_IN_OUT CONSTANT
    cReadOnly    : STRING(16);  (* Constant string variable can only be read here in POU *)
END_VAR
```

```
sReadWrite := 'String_from_POU';
nReadWrite := STRING_TO_DWORD(cReadOnly);
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    sVar10 : STRING(10) := '1234567890';
    sVar16 : STRING(16) := '1234567890123456';
    sVar20 : STRING(20) := '12345678901234567890';
    nVar   : DWORD;
END_VAR
```

```
// The following line of code causes the compiler error:
// VAR_IN_OUT parameter 'sReadWrite' of 'F_Manipulate' needs variable with write access as input.
F_Manipulate(sReadWrite := '1234567890123456', cReadOnly := '1234567890123456', nReadWrite := nVar);

// The following line of code causes the compiler error:
// String variable 'sVar10' too short for VAR_IN_OUT parameter 'sReadWrite' of 'F_Manipulate'
F_Manipulate(sReadWrite := sVar10, cReadOnly := sVar10, nReadWrite := nVar);

// Correct code
F_Manipulate(sReadWrite := sVar16, cReadOnly := '1234567890', nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := '1234567890123456', nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := '12345678901234567890', nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := sVar10, nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar16, cReadOnly := sVar16, nReadWrite := nVar);
F_Manipulate(sReadWrite := sVar20, cReadOnly := sVar20, nReadWrite := nVar);
```

16.2.5 Globale Variablen - VAR_GLOBAL

Globale Variablen sind „normale“ Variablen, Konstanten oder remanente Variablen, die im gesamten Projekt bekannt sind.

Sie deklarieren globale Variablen in globalen Variablenlisten zwischen den Schlüsselwörtern VAR_GLOBAL und END_VAR.

Das System erkennt eine globale Variable, wenn Sie dem Variablennamen ein Punkt voranstellen, zum Beispiel „.nGlobVar1“.

Eine lokal in einem Baustein deklarierte Variable, die denselben Namen hat wie eine globale Variable, hat innerhalb des Bausteins Vorrang.

Globale Variablen werden immer vor den lokalen Variablen von POUs initialisiert.

Beispiel:

```
VAR_GLOBAL
    nVarGlob1 : INT;
END_VAR
```

Siehe auch:

- Objekt Globale Variablenliste [▶ 75]
- Globaler Namensraum [▶ 751]

16.2.6 Temporäre Variablen - VAR_TEMP

Diese Funktionalität ist eine Erweiterung bezüglich der Norm IEC 61131-3.

Sie deklarieren temporäre Variablen lokal zwischen den Schlüsselwörtern VAR_TEMP und END_VAR.

VAR_TEMP-Deklarationen sind nur in Programmen und Funktionsbausteinen möglich.

TwinCAT initialisiert temporäre Variablen bei jedem Aufruf des Bausteins neu.

Auf die temporären Variablen kann die Anwendung nur im Implementierungsteil eines Programms oder eines Funktionsbausteins zugreifen.

Beispiel:

```
VAR_TEMP
    nVarTmp1 : INT; //1st temporary variable
END_VAR
```

16.2.7 Statische Variablen - VAR_STAT

Diese Funktionalität ist eine Erweiterung bezüglich der Norm IEC 61131-3.

Statische Variablen deklarieren Sie lokal zwischen den Schlüsselwörtern VAR_STAT und END_VAR. Die statischen Variablen initialisiert TwinCAT beim ersten Aufruf des jeweiligen Bausteins.

Auf statische Variablen können Sie nur innerhalb des Namensraums, in dem die Variablen deklariert sind, zugreifen (wie bei statischen Variablen in C). Jedoch behalten statische Variablen ihren Wert, wenn die Anwendung den Baustein verlassen hat. Sie können statische Variablen beispielsweise als Zähler für Funktionsaufrufe verwenden.

Sie können statische Variablen mit einem Attribut-Schlüsselwort erweitern.

Statische Variablen existieren jeweils nur einmal. Dies gilt auch für statische Variablen eines Funktionsblocks oder einer Funktionsblockmethode, auch wenn der Funktionsblock mehrfach instanziiert wird.

Beispiel:

```
VAR_STAT
    nVarStat1 : INT;
END_VAR
```

Siehe auch:

- Variablentypen - Attribut-Schlüsselwörter [▶ 731]

16.2.8 Externe Variablen - VAR_EXTERNAL

Externe Variablen sind globale Variablen, die in einen Baustein „importiert“ werden.

Die Variablen deklarieren Sie zwischen den Schlüsselwörtern VAR_EXTERNAL und END_VAR. Wenn die globale Variable nicht existiert, wird eine Fehlermeldung ausgegeben.

In TwinCAT 3 PLC ist es nicht notwendig, Variablen als extern zu deklarieren, um sie in einer POU zu verwenden. Das Schlüsselwort existiert, um die Kompatibilität zu IEC 61131-3 zu wahren.

Syntax:

```
<POU keyword> <POU name>
VAR_EXTERNAL
    <variable name> : <data type>;
END_VAR
```

Eine Initialisierung ist nicht erlaubt.

Achten Sie darauf, die Adressierung der allokierten Variablen (mit AT %I bzw. AT %Q) nur in der globalen Variablenliste vorzunehmen. Bei einer zusätzlichen Adressierung der lokalen Instanzen der Variablen treten Dopplungen im Prozessabbild auf.

Beispiel:

```
FUNCTION_BLOCK FB_Sample
VAR_EXTERNAL
    nVarExt1 : INT;     // 1st external variable
END_VAR
```

Siehe auch:

- Objekt Globale Variablenliste [▶ 75]

16.2.9 Instanzvariablen - VAR_INST

TwinCAT legt eine VAR_INST-Variable einer Methode nicht wie VAR-Variablen auf dem Methoden-Stack, sondern auf dem Stack der Funktionsbaustein-Instanz ab. Dies bedeutet, dass sich die VAR_INST-Variable wie andere Variablen der Funktionsbaustein-Instanz verhält und nicht bei jedem Aufruf der Methode neu initialisiert wird.

VAR_INST-Variablen sind nur in Methoden eines Funktionsbausteins erlaubt und Sie können nur innerhalb der Methode auf eine solche Variable zugreifen. Die Variablenwerte von Instanzvariablen monitoren Sie im Deklarationsteil der Methode.

Sie können Instanzvariablen nicht mit einem Attribut-Schlüsselwort erweitern.

Beispiel:

```
METHOD MethLast : INT
VAR_INPUT
    nVar  : INT;
END_VAR
VAR_INST
    nLast : INT := 0;
END_VAR
```

```
MethLast := nLast;
nLast    := nVar;
```

Siehe auch:

- Objekt Methode [▶ 93]

16.2.10 Konstante Variablen - CONSTANT

Die Deklaration konstanter Variablen erfolgt in den globalen Variablenlisten oder im Deklarationsteil von Programmierobjekten. In Implementierungen kann auf konstante Variablen über den Instanzpfad lesend zugegriffen werden, aber nicht schreibend.

Syntax

```
<scope> CONSTANT
    <identifier> : <data type> := <initial value> ;
END_VAR

<scope> : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
<data type>: <elementary data type> | <user defined data type> | <function block>
<initial value> : <literal value> | <identifier> | <expression>
```

Weisen Sie bei der Deklaration einer konstanten Variablen immer einen Initialisierungswert zu. Danach kann die Konstante nicht mehr beschrieben werden.

Beispiel:

Deklaration:

```
VAR CONSTANT
    cTaxFactor : REAL := 1.19;
END_VAR
```

Aufruf:

```
rPrice := rValue * cTaxFactor;
```

Sie können in einer Implementierung nur lesend auf konstante Variablen zugreifen. Konstante Variablen stehen rechts vom Zuweisungsoperator.

Siehe auch:

- Eingabe-/Ausgabevariablen - VAR_IN_OUT, VAR_IN_OUT CONSTANT [▶ 719]
- Operanden [▶ 780]

16.2.11 Generische konstante Variablen - VAR_GENERIC CONSTANT

Eine generische Konstante ist eine Variable im Deklarationsbereich VAR_GENERIC CONSTANT eines Funktionsbausteins, der erst bei der Deklaration der Funktionsbaustein-Instanz definiert wird.

Verfügbar ab TC3.1 Build 4026

Syntax

Deklaration des Funktionsbausteins:

```
FUNCTION_BLOCK <function block name>
VAR_GENERIC CONSTANT
    <generic constant name> : <integer data type> := <initial value>; //Initialwert wird
überschrieben
END_VAR
```

Die generische Konstante (ganzzahliger Datentyp) kann innerhalb eines Funktionsbausteins wie üblich verwendet werden. Beispielsweise können Sie die Konstante für die Definition der Größe von Arrays oder der Länge von Strings verwenden. Der Initialwert wird nur für Compile-Prüfungen benötigt. Zur Laufzeit wird der Wert überschrieben.

Deklaration des Funktionsbaustein-Instanz:

```
PROGRAM MAIN
VAR
    <fb instance name> : <function block name> < <literal> >;
    <fb instance name> : <function block name> <( <expression> )>;
END_VAR
```

Bei der Deklaration der Funktionsbaustein-Instanzen wird der Konstanten ein spezifischer Wert zugewiesen, der nur für diese Instanz gültig ist. Dafür wird dem als Datentyp fungierenden Funktionsbaustein der Wert (<literal>) in spitzen Klammern angehängt.

Alternativ dazu kann ein Ausdruck (<expression>) angehängt werden. Dieser muss allerdings in runden Klammern stehen, da es bei einem Ausdruck erlaubt ist, Symbole wie < oder > zu verwenden. Sonst ist die Eindeutigkeit des Codes nicht mehr gewährleistet.

Beispiel

Generischer Funktionsbaustein mit parametrierbarer Arrayvariable

Der folgende Code zeigt, wie ein Funktionsbaustein zu definieren ist, der Arrays beliebiger Länge verarbeiten kann. Der Funktionsbaustein verfügt über ein Array mit einer generischen, aber konstanten Länge. Mit "konstant" ist gemeint, dass, obwohl jede Funktionsbaustein-Instanz in ihrer Arraylänge variiert, diese während der Lebensdauer des Objekts konstant ist.

Ein solches Konstrukt ist beispielsweise für einen Bibliotheksprogrammierer von Vorteil, der einen generischen Bibliotheksbaustein implementieren möchte.

```
FUNCTION_BLOCK FB_Sample
VAR_GENERIC CONSTANT
    nMaxLen : UDINT := 1;
END_VAR
VAR
    aSample : ARRAY[0..nMaxLen-1] OF BYTE
END_VAR
```

```
PROGRAM MAIN
VAR CONSTANT
    cConst    : DINT := 10;
END_VAR
VAR
    fbSample1 : FB_Sample<100>;
    fbSample2 : FB_Sample<(2*cConst)>;
    aSample   : ARRAY[0..5] OF FB_Sample<10>;
END_VAR
```

Vererbung

Ein Funktionsbaustein mit generischen Konstanten kann auch die Vererbungskonstrukte EXTENDS und IMPLEMENTS verwenden.

Achten Sie beim Implementieren darauf, dass erst die Deklaration der generischen Konstanten eingefügt wird und dann EXTENDS und IMPLEMENTS nachfolgen. Das hat den Grund, dass generische Konstanten auch bei Basisklassen verwendet werden können.

Beispiel

```
INTERFACE I_Sample
```

```
FUNCTION_BLOCK FB_Sample
VAR_GENERIC CONSTANT
    nMaxLen   : UDINT := 1;
END_VAR
IMPLEMENTS I_Sample
VAR
    aSample   : ARRAY[0..nMaxLen-1] OF BYTE
END_VAR
```

```
FUNCTION_BLOCK FB_Sub_1 EXTENDS FB_Sample<100>
```

```
FUNCTION_BLOCK FB_Sub_2
VAR_GENERIC CONSTANT
    nMaxLen2  : UDINT := 1;
END_VAR
EXTENDS FB_Sample<nMaxLen2>
```

```
PROGRAM MAIN
VAR
    fbSample1 : FB_Sample<10>;
    fbSub1    : FB_Sub_1;
    fbSub2    : FB_Sub_2<20>;
END_VAR
```

16.2.12 Remanente Variablen - PERSISTENT, RETAIN

Remanente Variablen können ihren Wert über die übliche Programmlaufzeit hinaus behalten. Sie können remanente Variablen als RETAIN-Variablen oder noch strenger als PERSISTENT-Variablen im SPS-Projekt deklarieren.

Voraussetzung für die volle Funktionalität von RETAIN-Variablen ist ein entsprechender Speicherbereich auf der Steuerung (NovRam). Persistente Variablen werden nur beim Shutdown von TwinCAT geschrieben. Dazu bedarf es in der Regel einer entsprechenden USV. Ausnahme: Persistente Variablen können auch mit dem Baustein FB_WritePersistentData geschrieben werden.

Wenn es den entsprechenden Speicherbereich nicht gibt, gehen bei einem Stromausfall auch die Werte von RETAIN- oder PERSISTENT-Variablen verloren.

Die AT-Deklaration dürfen Sie nicht in Kombination mit VAR RETAIN oder VAR PERSISTENT verwenden.

Persistente Variablen

Sie deklarieren persistente Variablen, indem Sie im Deklarationsteil von Programmierobjekten hinter dem Schlüsselwort für den Variablentyp (VAR, VAR_GLOBAL etc.) das Schlüsselwort PERSISTENT hinzufügen.

PERSISTENT-Variablen behalten ihren Wert nach einem unkontrollierten Beenden, einem Reset kalt oder einem erneuten Download des SPS-Projekts.
Bei einem erneuten Start des Programms arbeitet das System mit den gespeicherten Werten weiter. In diesem Fall initialisiert TwinCAT andere „normale“ Variablen neu, entweder mit ihren explizit vorgegebenen Initialwerten oder mit den Standardinitialisierungen.
TwinCAT initialisiert PERSISTENT-Variablen nur bei einem Reset Ursprung neu.

Ein Anwendungsbeispiel für persistente Variablen ist ein Betriebsstundenzähler, der nach einem Stromausfall und auch nach einem erneuten Download des SPS-Projekts weiter zählen soll.

Übersichtstabelle zum Verhalten von PERSISTENT-Variablen

Nach Online-Befehl	VAR PERSISTENT
Reset kalt	Werte werden beibehalten
Reset Ursprung	Neuinitialisierung der Werte
Download	Werte werden beibehalten
Online-Change	Werte werden beibehalten

Beispiel:

Persistente Variablenliste:

```
VAR_GLOBAL PERSISTENT
    nVarPers1 : DINT; (* 1. Persistent variable *)
    bVarPers2 : BOOL; (* 2. Persistent variable *)
END_VAR
```

- Vermeiden Sie die Verwendung des Datentyps POINTER TO in persistenten Variablenlisten, da sich die Adresswerte bei erneutem Download des SPS-Projekts verändern können. TwinCAT gibt entsprechende Compiler-Warnungen aus.
- Wenn Sie eine lokale Variable in einer Funktion als PERSISTENT deklarieren, bleibt dies ohne Auswirkung. Die Verwendung von Datenpersistenz ist hier nicht möglich.
- Das Verhalten bei einem **Reset kalt** kann mit dem Pragma ‘TcInitOnReset [▶ 865]‘ beeinflusst werden

RETAIN-Variablen

Sie deklarieren RETAIN-Variablen, indem Sie im Deklarationsteil von Programmierobjekten hinter dem Schlüsselwort für den Variablentyp (VAR, VAR_GLOBAL etc.) das Schlüsselwort RETAIN hinzufügen.

Als RETAIN deklarierte Variablen werden zielsystemabhängig, aber typischerweise in einem eigenen Speicherbereich verwaltet, der gegen Stromausfall gesichert sein muss. Der sogenannte Retain-Handler sorgt dafür, dass die RETAIN-Variablen am Ende eines SPS-Zyklus und nur bei Änderung in den entsprechenden Bereich des NovRams geschrieben werden. Der Umgang mit dem Retain-Handler ist in der Dokumentation C/C++ im Kapitel „Retain Daten“ beschrieben.

RETAIN-Variablen behalten ihren Wert nach einem unkontrollierten Beenden (Spannungsausfall). Bei einem erneuten Start des Programms arbeitet das System mit den gespeicherten Werten weiter. In diesem Fall initialisiert TwinCAT andere „normale“ Variablen neu, entweder mit ihren explizit vorgegebenen Initialwerten oder mit den Standardinitialisierungen.
TwinCAT initialisiert RETAIN-Variablen neu bei einem Reset Ursprung.

Eine mögliche Anwendung ist ein Stückzähler in einer Fertigungsanlage, der nach einem Stromausfall weiter zählen soll.

Übersichtstabelle zum Verhalten von RETAIN-Variablen

Nach Online-Befehl	VAR RETAIN
Reset kalt	Werte werden beibehalten
Reset Ursprung	Neuinitialisierung der Werte
Download	Werte werden beibehalten

Beispiele:

In einer POU:

```
VAR RETAIN
    nRem1 : INT;
END_VAR
```

In einer GVL:

```
VAR_GLOBAL RETAIN
    nVarRem1 : INT;
END_VAR
```

- Wenn Sie eine lokale Variable in einem Programm oder Funktionsbaustein als RETAIN deklarieren, speichert TwinCAT genau diese Variable im Retain-Bereich (wie eine globale RETAIN-Variable).
- Wenn Sie eine lokale Variable in einer Funktion als RETAIN deklarieren, hat dies keine Auswirkung. TwinCAT speichert die Variable nicht im Retain-Bereich.

Gesamte Übersichtstabelle

Der Grad der Werteerhaltung von RETAIN-Variablen ist automatisch in der von PERSISTENT-Variablen enthalten.

Nach Online-Befehl	VAR	VAR RETAIN	VAR PERSISTENT
Reset kalt	Neuinitialisierung der Werte	Werte werden beibehalten	Werte werden beibehalten
Reset Ursprung	Neuinitialisierung der Werte	Neuinitialisierung der Werte	Neuinitialisierung der Werte
Download	Neuinitialisierung der Werte	Werte werden beibehalten	Werte werden beibehalten
Online-Change	Werte werden beibehalten	Werte werden beibehalten	Werte werden beibehalten

Siehe auch:

- Datenpersistenz [▶ 171]
- Reset des SPS-Projekts durchführen [▶ 230]

16.2.13 SUPER

SUPER ist eine spezielle Variable und dient der objektorientierten Programmierung.

SUPER ist der Zeiger eines Funktionsbausteins auf die Basis-Funktionsbaustein-Instanz, aus die der Funktionsbaustein erzeugt wurde. Der SUPER-Zeiger erlaubt somit auch den Zugriff auf die Implementierung der Methoden des Basis-Funktionsbausteins (Basisklasse). Für jeden Funktionsbaustein steht automatisch ein SUPER-Zeiger zur Verfügung.

Sie können SUPER nur in Methoden- und in den zugehörigen Funktionsbaustein-Implementierungen verwenden.

Dereferenzierung des Zeigers: SUPER^

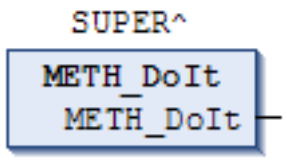
Verwendung des SUPER-Zeigers: Mithilfe des Schlüsselworts SUPER rufen Sie die Implementierung der Basisklasse oder eine Methode auf, die in der Instanz der Basisklasse gültig ist.

Beispiele:

ST:

```
SUPER^();                  // Call of FB-body of base class
SUPER^.METH_DoIt();        // Call of method METH_DoIt that is implemented in base class
```

FUP/CFC/LD:



SUPER ist für die Anweisungsliste (AWL) nicht implementiert.

Verwendung von SUPER- und THIS-Zeiger:

Funktionsbaustein FB_Base:

```
FUNCTION_BLOCK FB_Base
VAR_OUTPUT
    nCnt : INT;
END_VAR
```

Methode FB_Base.METH_DoIt:

```
METHOD METH_DoIt : BOOL
```

```
nCnt := -1;
```

Methode FB_Base.METH_DoAlso:

```
METHOD METH_DoAlso : BOOL
```

```
METH_DoAlso := TRUE;
```

Funktionsbaustein FB_1:

```
FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_OUTPUT
    nBase: INT;
END_VAR
```

```
THIS^.METH_DoIt();      // Call of the methods of FB_1
THIS^.METH_DoAlso();

SUPER^.METH_DoIt();     // Call of the methods of FB_Base
SUPER^.METH_DoAlso();
nBase := SUPER^.nCnt;
```

Methode FB_1.METH_DoIt:

```
METHOD METH_DoIt : BOOL
```

```
nCnt := 1111;
METH_DoIt := TRUE;
```

Methode FB_1.METH_DoAlso:

```
METHOD METH_DoAlso : BOOL
```

```
nCnt := 123;
METH_DoAlso := FALSE;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbMyBase : FB_Base;
    fbMyFB_1 : FB_1;
    nTHIS    : INT;
    nBase    : INT;
END_VAR
```

```
fbMyBase();
nBase := fbmyBase.nCnt;
fbMyFB_1();
nTHIS := fbMyFB_1.nCnt;
```

Siehe auch:

- Datentypen [▶ 793]
- THIS [▶ 730]

16.2.14 THIS

THIS ist eine spezielle Variable und dient der objektorientierten Programmierung.

THIS ist der Zeiger eines Funktionsbausteins auf seine eigene Funktionsbaustein-Instanz. Für jeden Funktionsbaustein steht automatisch ein THIS-Zeiger zur Verfügung.

Sie können THIS nur in Methoden und in Funktionsbausteinen verwenden. THIS steht für die Implementierung in der **Eingabehilfe** in der Kategorie **Schlüsselwörter** zur Verfügung.

Dereferenzierung des Zeigers: THIS^

Verwendung des THIS-Zeigers:

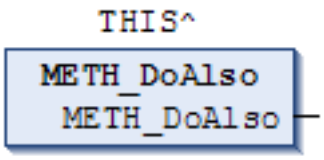
- Wenn in einer Methode eine lokale Variable eine Funktionsbaustein-Variable verschattet, können Sie die Funktionsbaustein-Variable mit dem THIS- Zeiger setzen. Siehe unten Beispiel (1)
- Wenn der Zeiger auf die eigene Funktionsbaustein-Instanz zur Verwendung in einer Funktion referenziert wird. (Siehe unten Beispiel (2))

Beispiele:

ST:

```
THIS^.METH_DoIt();
```

FUP/CFC/LD:



THIS ist für die Anweisungsliste (AWL) nicht implementiert.

Beispiele:

(1) Die lokale Variable nVarB verschattet die Funktionsbaustein-Variable nVarB.

```
FUNCTION_BLOCK FB_A
VAR_INPUT
    nVarA: INT;
END_VAR

nVarA := 1;

FUNCTION_BLOCK FB_B EXTENDS FB_A
VAR_INPUT
    nVarB : INT := 0;
```

```
END_VAR

nVarA := 11;
nVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
    nVarB : INT;
END_VAR

nVarB := 22; // The local variable nVarB is set.
THIS^.nVarB := 222; // The function block variable nVarB is set even though nVarB is obscured.

PROGRAM MAIN
VAR
   fbMyfbB : FB_B;
END_VAR

fbMyfbB(nVarA:=0, nVarB:= 0);
fbMyfbB.DoIt();
```

(2) Ein Funktionsaufruf benötigt die Referenz auf die eigene Instanz.

```
FUNCTION F_FunA
VAR_INPUT
    fbMyFbA : FB_A;
END_VAR
...;

FUNCTION_BLOCK FB_A
VAR_INPUT
    nVarA: INT;
END_VAR
...;

FUNCTION_BLOCK FB_B EXTENDS FB_A
VAR_INPUT
    nVarB: INT := 0;
END_VAR

nVarA := 11;
nVarB := 2;

METHOD DoIt : BOOL
VAR_INPUT
END_VAR
VAR
    nVarB: INT;
END_VAR

nVarB := 22;  //The local variable nVarB is set.
F_FunA(fbMyFbA := THIS^);  //F_FunA is called via THIS^.

PROGRAM MAIN
VAR
    fbMyFbB: FB_B;
END_VAR

fbMyFbB(nVarA:=0 , nVarB:= 0);
fbMyFbB.DoIt();
```

Siehe auch:

- Pointer [▶ 804]
- SUPER [▶ 728]

16.2.15 Variablentypen - Attribut-Schlüsselwörter

In der Variablendeklaration können Sie dem Variablentyp folgende Schlüsselwörter hinzufügen:

- RETAIN: für remanente Variablen vom Typ RETAIN
- PERSISTENT: für remanente Variablen vom Typ PERSISTENT
- CONSTANT: für Konstanten

Siehe auch:

- Konstanten [▶ 780]
- Remanente Variablen - RETAIN, PERSISTENT [▶ 726]

16.3 Operatoren

TwinCAT 3 PLC unterstützt alle Operatoren der Norm IEC-61131-3. Diese Operatoren sind implizit überall im Projekt bekannt. Neben den IEC-Operatoren unterstützt TwinCAT 3 PLC auch einige Operatoren, die nicht in der Norm IEC 61131-3 beschrieben sind.

Operatoren werden in einem Baustein wie Funktionen benutzt.

Operationen mit Gleitkomma-Datentypen

Bei Operationen mit Gleitkomma-Datentypen ist das Rechenergebnis abhängig von der verwendeten Zielsystem-Hardware.

Operationen mit Über- oder Unterlauf

Bei Operationen mit Überlauf oder Unterlauf im Datentyp ist das Rechenergebnis abhängig von der verwendeten Zielsystemhardware.

Informationen zur Abarbeitungsreihenfolge (Bindungsstärke) der ST-Operatoren finden Sie im Abschnitt „ST-Ausdrücke [▶ 657]“.

Überlauf/Unterlauf im Datentyp

Der TwinCAT 3 Compiler generiert Code für das jeweilige Zielgerät und berechnet dabei Zwischenergebnisse immer mit der nativen Größe, die durch das Zielsystem vorgegeben ist. Beispielsweise wird auf den Systemen x86 und ARM mindestens mit 32 Bit Zwischenwerten gerechnet und auf x64-Systemen immer mit 64 Bit Zwischenwerten. Dies bietet deutliche Vorteile bei der Rechengeschwindigkeit und produziert häufig auch das erwartete Ergebnis. Dies bedeutet aber auch, dass ein Überlauf oder ein Unterlauf im Datentyp unter Umständen nicht abgeschnitten wird.

Beispiel 1

Das Ergebnis dieser Addition wird nicht abgeschnitten und das Ergebnis in dwVar ist 65536.

```
VAR
    nVarWORD  : WORD;
    nVarDWORD : DWORD;
END_VAR
```

```
nVarWORD  := 65535;
nVarDWORD := nVarWORD + 1;
```

Beispiel 2

Der Überlauf und Unterlauf im Datentyp wird nicht abgeschnitten und die Ergebnisse (bVar1, bVar2) beider Vergleiche sind auf einer 32 Bit und auf einer 64 Bit Hardware FALSE.

```
VAR
    nVar1 : WORD;
    nVar2 : WORD;
    bVar1 : BOOL;
    bVar2 : BOOL;
END_VAR
```

```
nVar1 := 65535;
nVar2 := 0;
bVar1 := (nVar1 + 1) = nVar2;
bVar2 := (nVar2 - 1) = nVar1;
```

Beispiel 3

Durch die Zuweisung auf nVar3 wird der Wert auf den Zieldatentyp WORD abgeschnitten und das Ergebnis bVar1 ist TRUE.

```
VAR
    nVar1 : WORD;
    nVar2 : WORD;
    nVar3 : WORD;
    bVar1 : BOOL;
END_VAR
```

```
nVar1 := 65535;
nVar2 := 0;
nVar3 := (nVar1 + 1);
bVar1 := (nVar3 = nVar2);
```

Beispiel 4

Um den Compiler zu einem Abschneiden des Zwischenergebnisses zu zwingen, kann eine Konvertierung eingefügt werden.

Durch die Typkonvertierung wird sichergestellt, dass beide Vergleiche nur 16 Bit vergleichen und die Ergebnisse (bVar1, bVar2) beider Vergleiche jeweils TRUE sind.

```
VAR
    nVar1 : WORD;
    nVar2 : WORD;
    bVar1 : BOOL;
    bVar2 : BOOL;
END_VAR
```

```
nVar1 := 65535;
nVar2 := 0;
bVar1 := (TO_WORD(nVar1 + 1) = nVar2);
bVar2 := (TO_WORD(nVar2 - 1) = nVar1);
```

Adressoperatoren

- ADR [▶ 736]
- BITADR [▶ 737]
- Inhaltsoperator [▶ 736]

Arithmetische Operatoren

- ADD [▶ 737]
- SUB [▶ 738]
- MUL [▶ 739]
- DIV [▶ 740]
- MOD [▶ 741]
- MOVE [▶ 742]
- INDEXOF [▶ 742]
- SIZEOF [▶ 742]
- XSIZEOF [▶ 743]

Aufrufoperatoren

- CAL [▶ 743]

Auswahloperatoren

- LIMIT [▶ 744]
- MAX [▶ 744]
- MIN [▶ 744]
- MUX [▶ 745]
- SEL [▶ 745]

Bitshift-Operatoren

- ROL [▶ 747]
- ROR [▶ 748]
- SHL [▶ 746]
- SHR [▶ 747]

Bitstring-Operatoren

- AND [▶ 749]
- AND_THEN [▶ 749]
- NOT [▶ 749]
- OR [▶ 750]
- OR_ELSE [▶ 750]
- XOR [▶ 751]

Namensraumoperatoren

Die Namensraumoperatoren sind eine Erweiterung der IEC 61131-3 Operatoren. Sie bieten Ihnen Möglichkeiten, den Zugriff auf Variablen oder Module eindeutig zu gestalten, auch wenn Sie den gleichen Variablen- oder Modulnamen im Projekt mehrmals verwenden.

- Bibliotheksnamensraum [▶ 752]
- Enumerationsnamensraum [▶ 752]
- Globaler Namensraum [▶ 751]
- Namensraum für Globale Variablenlisten [▶ 751]

Numerische Operatoren

- ABS [▶ 752]
- ACOS [▶ 753]
- ASIN [▶ 753]
- ATAN [▶ 753]
- COS [▶ 754]
- EXP [▶ 754]
- EXPT [▶ 755]
- LN [▶ 755]
- LOG [▶ 756]
- SIN [▶ 756]
- SQRT [▶ 756]
- TAN [▶ 757]

Typkonvertierungsoperatoren

Sie können explizit Typkonvertierungsoperatoren aufrufen. Für getypte Konvertierungen von einem elementaren Typ in einen anderen elementaren Typ und auch für Überladungen stehen die unten beschriebenen Typkonvertierungsoperatoren zur Verfügung. Konvertierungen von einem „kleineren“ Typ auf einen „größeren“ Typ, wie beispielsweise von BYTE nach INT oder von WORD nach DINT, sind aber auch implizit möglich.

Getypte Konvertierung: `<elementary data type>_TO_<another elementary data type>`

Überladene Konvertierung: `TO_<elementary data type>`

Elementare Datentypen:

```
<elementary data type> =
__UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DINT | DT | DWORD | INT | LDATE | LDT | LINT
| LREAL | LTIME | LTOD | LWORD | REAL | SINT | TIME | TOD | UDINT | UINT | ULINT | USINT | WORD
```

Die Schlüsselwörter TIME_OF_DAY und DATE_AND_TIME sind alternative Schreibweisen für die Datentypen TOD und DT. TIME_OF_DAY und DATE_AND_TIME werden nicht als Typkonvertierungsbefehl abgebildet.

Wenn bei einem Typkonvertierungsoperator der Operandenwert außerhalb des Wertebereichs des Zieldatentyps liegt, ist die Ergebnisausgabe undefiniert. Dies ist beispielsweise der Fall, wenn ein negativer Operandenwert von LREAL in den Zieldatentyp UINT konvertiert wird.

Bei der Typkonvertierung von größeren auf kleinere Typen können Informationen verloren gehen.

Stringmanipulation bei Konvertierung nach STRING oder WSTRING

Bei einer Typkonvertierung nach STRING oder WSTRING wird der getypte Wert als Zeichenfolge linksbündig abgelegt und bei Überlänge abgeschnitten. Deklarieren Sie deshalb die Rückgabevariablen für die Typkonvertierungsoperatoren <type>_TO_STRING und <type>_TO_WSTRING ausreichend lang, so dass die Zeichenfolge ohne Manipulation Platz findet.

- Überladung [▶ 758]
- <INT type>_TO_<INT type> [▶ 760]
- <type>_TO_BOOL [▶ 759]
- BOOL_TO_<type> [▶ 759]
- DATE/DT_TO_<type> [▶ 764]
- REAL/LREAL_TO_<type> [▶ 761]
- STRING_TO_<type> [▶ 762]
- TO_STRING/TO_WSTRING für Enumerationsvariablen [▶ 762]
- TIME/TOD_TO_<type> [▶ 763]
- TRUNC [▶ 765]
- TRUNC_INT [▶ 765]

Vergleichsoperatoren

Die Vergleichsoperatoren sind boolesche Operatoren, die jeweils zwei Eingänge (erster und zweiter Operand) miteinander vergleichen.

- EQ [▶ 766]
- GE [▶ 766]
- GT [▶ 767]
- LE [▶ 767]
- LT [▶ 767]
- NE [▶ 768]

Weitere Operatoren

- __DELETE [▶ 771]
- __ISVALIDREF [▶ 772]
- __NEW [▶ 768]
- __QUERYINTERFACE [▶ 772]
- __QUERYPOINTER [▶ 773]

- __VARINFO [▶ 777]
- __POUNAME [▶ 779]
- __POSITION [▶ 779]

16.3.1 Adressoperatoren

16.3.1.1 ADR

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

ADR liefert die Adresse seines Arguments in einem PVOID bzw. abhängig vom Laufzeitsystem in einem DWORD (32-Bit-Systeme) oder LWORD (32- und 64-Bit-Systeme). Um die Unabhängigkeit von der Laufzeitsystemarchitektur zu gewährleisten, empfiehlt es sich, PVOID als Datentyp zu verwenden.

⚠ VORSICHT

Verschiebung der Inhalte von Adressen durch Online-Change

Wenn Sie einen Online-Change anwenden, können sich Inhalte von Adressen verschieben. Dadurch könnten POINTER-Variablen auf einen ungültigen Speicherbereich zeigen. Um Probleme zu vermeiden, stellen Sie sicher, dass TwinCAT den Wert von Pointern bei einem Online-Change aktualisiert.

Syntax:

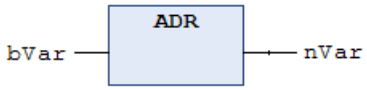
```
VAR
    <address name> : PVOID | DWORD | LWORD | __XWORD | POINTER TO < basis data type>;
END_VAR
<address name> := ADR(<variable name>);
```

Beispiele:

ST:

```
nVar := ADR(bVAR);
```

FUP:



Beispiel mit verschiedenen Datentypen:

```
FUNCTION_BLOCK FB_Address
VAR
    nVar      : INT := 10;
    pNumber   : POINTER TO INT;
    nAddress1 : PVOID;
    nAddress2 : DWORD;
    nAddress3 : LWORD;
    nAddress4 : __XWORD;
END_VAR
```

```
pNumber   := ADR(nVar);   // pNumber wird der Adresse von nVar zugewiesen
nAddress1 := ADR(nVar);   // PVOID   : für 32- und 64-Bit-Systeme
nAddress2 := ADR(nVar);   // DWORD   : nur für 32-Bit-Systeme
nAddress3 := ADR(nVar);   // LWORD   : für 32- und 64-Bit-Systeme
nAddress4 := ADR(nVar);   // __XWORD : für 32- und 64-Bit-Systeme
```

Siehe auch:

- Zeiger / POINTER [▶ 804]
- Spezialdatentypen UXINT, XINT, XWORD und PVOID [▶ 804]

16.3.1.2 Inhaltsoperator

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator ermöglicht das Dereferenzieren eines Pointers. Sie hängen den Operator als ^ an den Pointer-Bezeichner an.

⚠ VORSICHT

Verschiebung der Inhalte von Adressen durch Online-Change

Wenn Sie einen Online-Change anwenden, können sich Inhalte von Adressen verschieben.

Beispiel:

ST:

```
pSample : POINTER TO INT;
nInt1   : INT;
nInt2   : INT;
pSample := ADR(nInt1);
nInt2   := pSample^;
```

16.3.1.3 BITADR

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

BITADR liefert den Bit-Offset innerhalb des Segments in einem DWORD.

Der höchstwertige Nibble (4 Bit) in diesem DWORD beschreibt den Speicherbereich:

Merker: 16x40000000

Eingang: 16x80000000

Ausgang: 16xC0000000

⚠ VORSICHT

Verschiebung der Inhalte von Adressen durch Online-Change

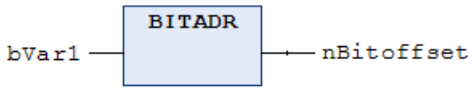
Wenn Sie einen Online-Change anwenden, können sich Inhalte von Adressen verschieben.

Beispiele:

ST:

```
VAR
    bVar1 AT %IX2.3 : BOOL;
    nBitoffset : DWORD;
END_VAR

nBitoffset := BITADR(bVar1); (*Result if byte addressing=TRUE: 16x80000013, if byte addressing = FAL
SE : 16x80000023*)
```

FUP:



16.3.2 Arithmetische Operatoren

16.3.2.1 ADD

Der IEC-Operator addiert Variablen.

Erlaubte Datentypen: __UXINT, __XINT, __XWORD, BYTE, DATE, DATE_AND_TIME, DINT, DT, DWORD, INT, LDATE, LDATE_AND_TIME, LDT, LINT, LREAL, LTIME, LTOD, LWORD, REAL, SINT, TIME, TIME_OF_DAY, TOD, UDINT, UINT, ULINT, USINT, WORD

Mögliche Kombinationen für Zeitdatentypen:

- `TIME + TIME = TIME`

- `TIME + LTIME = LTIME`
- `LTIME + LTIME = LTIME`

Mögliche Kombinationen für Datums- und Uhrzeitdatentypen:

- `TOD + TIME = TOD`
- `DT + TIME = DT`
- `TOD + LTIME = LTOD`
- `DT + LTIME = LDT`
- `LTOD + TIME = LTOD`
- `LDT + LTIME = LDT`
- `LTOD + LTIME = LTOD`
- `LDT + LTIME = LDT`

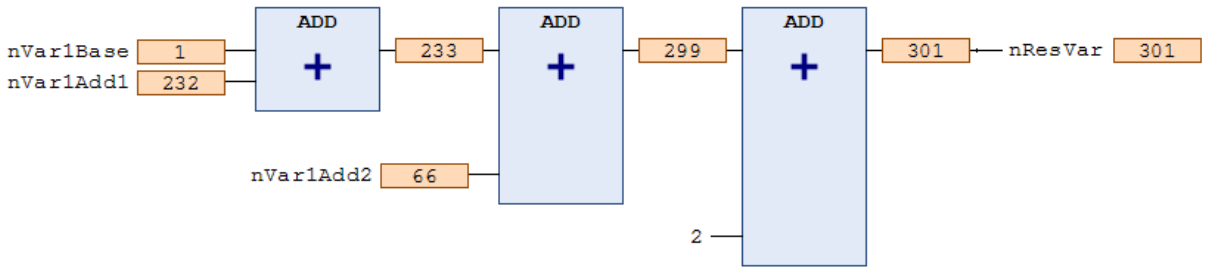
Besonderheit im FUP/KOP-Editor: Sie können den ADD-Operator um Bausteineingänge erweitern. Die Anzahl der zusätzlichen Bausteineingänge ist begrenzt.

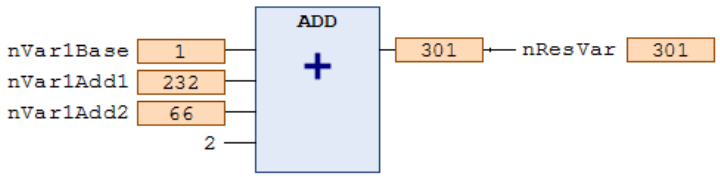
Beispiele:

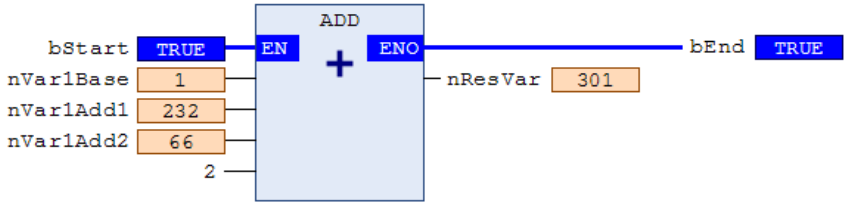
ST:

```
nVar := 7+2+4+7;
```

FUP:







16.3.2.2 SUB

Der IEC-Operator subtrahiert Variablen.

Erlaubte Datentypen: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME, TIME_OF_DAY, TOD, LTIME, LTIME_OF_DAY, LTOD, DATE, DATE_AND_TIME, DT, LDATE, LDATE_AND_TIME, LDT

Mögliche Kombinationen für Zeitdatentypen:

- TIME – TIME = TIME
- LTIME – LTIME = LTIME

Mögliche Kombinationen für Datums- und Uhrzeitdatentypen:

- DATE – DATE = TIME
- LDATE – LDATE = LTIME
- TOD – TIME = TOD
- LTOD – LTIME = LTOD
- TOD – TOD = TIME
- LTOD – LTOD = LTIME
- DT – TIME = DT
- LDT – LTIME = LDT
- DT – DT = TIME
- LDT – LDT = LTIME

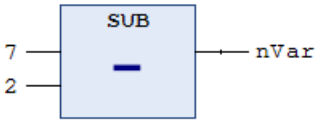
Negative TIME/LTIME-Werte sind undefiniert.

Beispiele:

ST:

```
nVar := 7-2;
```

FUP:



16.3.2.3 MUL

Der IEC-Operator dient der Multiplikation von Variablen.

Erlaubte Datentypen: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME

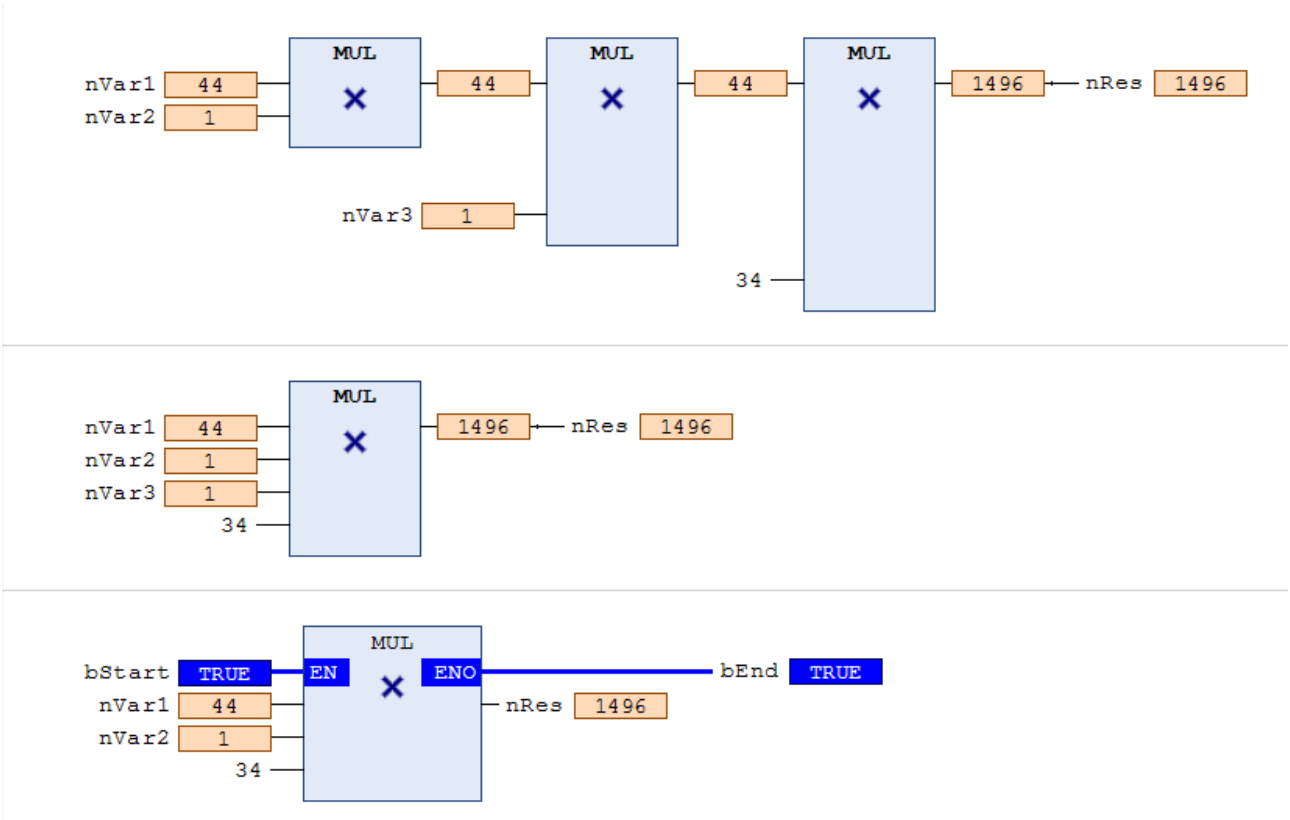
Besonderheit im FUP/KOP-Editor: Sie können den MUL-Operator um zusätzliche Bausteineingänge erweitern. Die Anzahl der zusätzlichen Bausteineingänge ist begrenzt.

Beispiele:

ST:

```
nVar := 7*2*4*7;
```

FUP:



16.3.2.4 DIV

Der IEC-Operator dient der Division von Variablen.

Erlaubte Datentypen: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, TIME

Division durch Null führt in TwinCAT immer zu einer Exception (Ausnahme) und dem Stopp des entsprechenden Tasks.

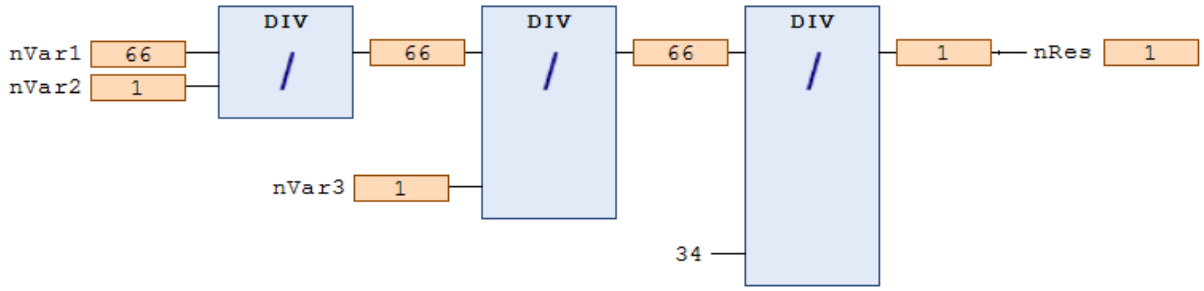
Beispiele:

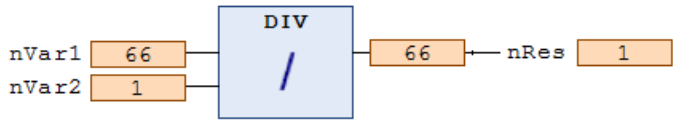
ST:

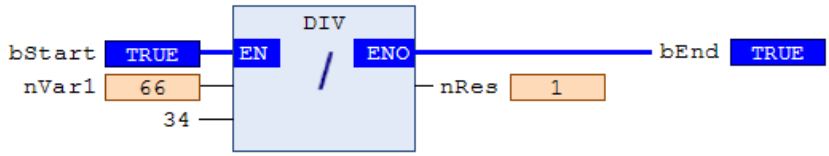
```
nVar := 8/2;
```

FUP:

1. Reihe von DIV-Bausteinen, 2. Einzelner DIV-Baustein, 3. DIV-Baustein mit EN/ENO-Parametern







Beachten Sie die Möglichkeit, eine Division durch 0 während der Laufzeit durch die impliziten Überwachungsfunktionen CheckDivInt, CheckDivLint, CheckDivReal und CheckDivLReal zu überwachen.

Siehe auch:

- Division Checks (POUs CheckDivDInt, CheckDivLInt, CheckDivReal, CheckDivLReal) [▶ 175]
- POU CheckDivLInt
- POU CheckDivLReal
- POU CheckDivReal

16.3.2.5 MOD

Der IEC-Operator dient der Modulo-Division.

Das Ergebnis der Funktion ist der ganzzahlige Rest der Division.

Erlaubte Datentypen: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT

Division durch Null führt in TwinCAT immer zu einer Exception (Ausnahme) und dem Stopp des entsprechenden Tasks.
Aber das Ergebnis von Mod 0 ist 0.

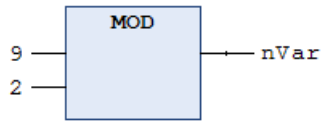
Beispiele:

Ergebnis: nVar ist 1.

ST:

```
nVar := 9 MOD 2;
```

FUP:



16.3.2.6 MOVE

Der IEC-Operator dient der Zuweisung einer Variablen auf eine andere Variable eines entsprechenden Typs.

Da MOVE in den CFC-, FUP- und KOP-Editoren als Baustein verfügbar ist, können Sie dort die EN/EN0-Funktionalität auch auf eine Variablenzuweisung anwenden.

Beispiele:

Ergebnis: var2 erhält den Wert von var1.

CFC in Verbindung mit der EN/EN0 Funktion:

Nur wenn en_i TRUE ist, weist TwinCAT den Wert der Variablen var1 Variable var2 zu.

ST:

```
ivar2 := MOVE(ivar1);
```

Dies entspricht:

```
ivar2 := ivar1;
```

16.3.2.7 INDEXOF

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Anstelle des Operators INDEXOF steht in TwinCAT der ADR-Operator zur Verfügung, um einen Zeiger auf den Index eines Bausteins zu erhalten.

Siehe auch:

- ADR [▶ 736]

16.3.2.8 SIZEOF

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator dient dazu, die Anzahl Bytes zu bestimmen, die die angegebene Variable x benötigt. Der SIZEOF-Operator liefert immer einen vorzeichenlosen Wert. Der Typ der Rückgabevariablen passt sich der gefundenen Größe von Variable x an.

Rückgabewert von SIZEOF(x)	Datentyp der Konstanten, die TwinCAT implizit für die gefundene Größe verwendet.
0 <= Größe von x < 256	USINT
256 <= Größe von x < 65536	UINT
65536 <= Größe von x < 4294967296	UDINT
4294967296 <= Größe von x	ULINT

Beispiele:

Ergebnis: nVar ist 10.

ST:

```
aArr1 : ARRAY[0..4] OF INT;
nVar  : INT;

nVar := SIZEOF(aArr1); (*nVar := USINT#10;*)
```

Sehen Sie dazu auch

XSIZEOF [▶ 743]

16.3.2.9 XSIZEOF

Verfügbar ab TC3.1 Build 4026

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator XSIZEOF bestimmt die Anzahl an Bytes, die in der übergebenen Variable oder dem übergebenen Datentyp benötigt werden.

Dabei wird immer ein vorzeichenloser Wert zurückgegeben. Der Datentyp des Rückgabewerts <return value> ist folgendermaßen festgelegt: bei 64-bit-Plattformen ist der Typ ULINT, auf allen anderen Plattformen UDINT. Um Code zu erzeugen, der auf allen Plattformen läuft, kann der Rückgabewert mit dem Datentyp __UXINT deklariert werden.

Syntax:

```
<return value> := XSIZEOF( <variable> );
```

Beispiel:

```
PROGRAM MAIN
VAR
    nReturnValue : __UXINT;             // Datentyp bei 64-bit-Plattformen: ULINT
    aData1       : ARRAY[0..4] OF INT;
END_VAR
```

```
nReturnValue := XSIZEOF(aData1);
```

Ergebnis:

```
nReturnValue = 10
```

Bei der Zuweisung an eine Variable vom Typ __UXINT ist es ratsam, den Operator XSIZEOF anstelle des Operators SIZEOF [▶ 742] zu verwenden. Denn bei XSIZEOF hängt der Datentyp des Rückgabewerts von der jeweiligen Plattform ab. Infolgedessen treten Probleme, die bei der Verwendung des Operators SIZEOF auftreten, nicht auf.

16.3.3 Aufrufoperatoren

16.3.3.1 CAL

Der IEC-Operator dient dem Aufruf eines Funktionsbausteins.

CAL ruft in AWL die Instanz eines Funktionsbausteins auf.

Syntax:

CAL <Funktionsbaustein> (<Eingangsvariable1> := <Wert>, <EingangsvariableN> := <Wert>)

Beispiel:

Aufruf der Instanz fbInst eines Funktionsbausteins mit Belegung der Eingangsvariablen nVar1, bVar2 auf 0 oder TRUE.

```
CAL fbInst(nVar1 := 0, bVar2 := TRUE)
```

16.3.4 Auswahloperatoren

16.3.4.1 LIMIT

Der IEC-Auswahloperator dient der Limitierung.

Syntax: OUT := LIMIT(Min, IN, Max)

bedeutet: OUT := MIN(MAX (IN, Min), Max)

Max ist die obere, Min ist die untere Schranke für das Ergebnis. Wenn der Wert IN die obere Grenze Max überschreitet, dann liefert LIMIT Max. Wenn IN Min unterschreitet, dann ist das Ergebnis Min.

Erlaubte Datentypen für IN und OUT: alle

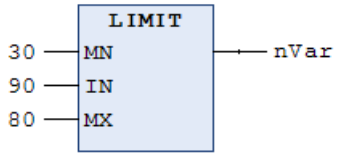
Beispiele:

Ergebnis: nVar ist 80.

ST:

```
nVar := LIMIT(30,90,80);
```

FUP:



16.3.4.2 MAX

Der IEC-Operator dient der Maximumsfunktion. Er liefert den größten Wert der übergebenen Eingangswerte.

Syntax: OUT := MAX(IN0, IN1,<weitere Eingänge>)

Erlaubte Datentypen: alle

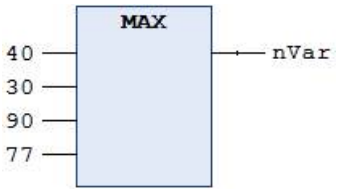
Beispiele:

Ergebnis: nVar ist 90.

ST:

```
nVar := MAX(40,30,90,77);
```

FUP:



16.3.4.3 MIN

Der IEC-Operator dient der Minimumsfunktion. Er liefert den kleinsten Wert der übergebenen Eingangswerte.

Syntax: OUT := MIN(IN0,IN1,<weitere Eingänge>)

Erlaubte Datentypen: alle

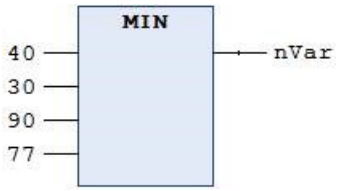
Beispiele:

Ergebnis: nVar ist 30.

ST:

```
nVar := MIN(40,30,90,77);
```

FUP:



16.3.4.4 MUX

Der IEC-Operator dient als Multiplexer.

Syntax: OUT := MUX(K, IN0,...,INn)

Das bedeutet OUT = IN_K

Erlaubte Datentypen für K: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, LINT, ULINT, UDINT

IN0, ..., INn und OUT: Beliebiger, identischer Datentyp. Achten Sie speziell bei der Verwendung von benutzerdefinierten Datentypen darauf, dass an allen drei Positionen Variablen mit identischem Typ verwendet werden. Der Compiler prüft die Typgleichheit und gibt Übersetzungsfehler aus. Insbesondere die Zuweisung von Instanzen eines Funktionsbausteins an Schnittstellen(variablen) wird nicht unterstützt.

MUX wählt aus einer Menge von Werten den K-ten aus. Der erste Wert entspricht K=0. Wenn K größer als die Anzahl der weiteren Eingänge (n) ist, so gibt TwinCAT den letzten Wert weiter (INn).

Zum Zweck der Laufzeitoptimierung berechnet TwinCAT nur den Ausdruck, den Sie IN_K vorgeschaltet haben.

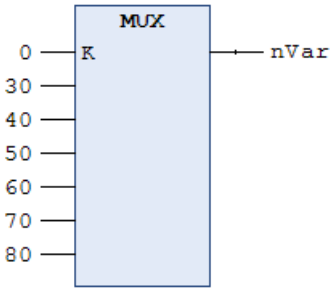
Beispiele:

Ergebnis: nVar ist 30.

ST:

```
nVar := MUX(0,30,40,50,60,70,80);
```

FUP:



16.3.4.5 SEL

Der IEC-Operator dient der bitweisen Auswahl.

Syntax:

OUT := SEL(G, IN0, IN1) bedeutet:

OUT := IN0; wenn G = FALSE

OUT := IN1; wenn G = TRUE

Erlaubte Datentypen:

IN0, ..., INn und OUT: Beliebiger, identischer Datentyp. Achten Sie speziell bei der Verwendung von benutzerdefinierten Datentypen darauf, dass an allen drei Positionen Variablen mit identischem Typ verwendet werden. Der Compiler prüft die Typgleichheit und gibt Übersetzungsfehler aus. Insbesondere die Zuweisung von Instanzen eines Funktionsbausteins an Schnittstellen(variablen) wird nicht unterstützt.

G: BOOL

TwinCAT berechnet einen Ausdruck, der IN0 vorgeschaltet ist, nicht, wenn G = TRUE. TwinCAT berechnet einen Ausdruck, der IN1 vorgeschaltet ist, nicht, wenn G = FALSE.

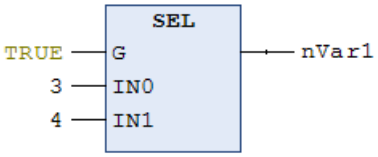
Bei grafischen Programmiersprachen werden unabhängig vom Eingang G die Ausdrücke an IN0 und IN1 berechnet, wenn ein Baustein, ein Sprung, ein Return, eine Leitungsverzweigung oder eine Flankenerkennung vorgeschaltet ist.

Beispiele:

ST:

```
nVar1 := SEL(TRUE,3,4); (*Result: 4*)
```

FUP:



16.3.5 Bitshift-Operatoren

16.3.5.1 SHL

Der IEC-Operator dient dem bitweisen Verschieben eines Operanden nach links.

Syntax: erg := SHL (in, n)

in: Operand, der nach links verschoben wird.

n: Anzahl der Bits, um die in nach links verschoben wird.

Wenn n die Datentyp-Breite überschreitet, hängt es vom Zielsystem ab, wie BYTE-, WORD-, DWORD- und LWORD-Operanden aufgefüllt werden. Die Zielsysteme bewirken Auffüllen mit Nullen oder mit n MOD <Registerbreite>.

Beachten Sie, dass Sie die Anzahl der Bits, die TwinCAT für die Rechenoperation berücksichtigt, durch den Datentyp der Eingangsvariablen in vorgeben.

Beispiele:

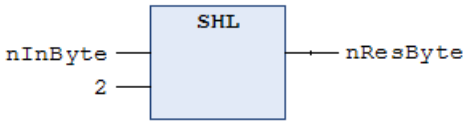
Die Ergebnisse für nResByte und nResWord sind unterschiedlich, obwohl die Werte der Eingangsvariablen nInByte und nInWord gleich, die Datentypen der Eingangsvariablen jedoch unterschiedlich sind.

ST:

```
PROGRAM Shl_st
VAR
    nInByte  :  BYTE:=16#45; (*2#01000101*)
    nInWord  :  WORD:=16#0045; (*2#0000000001000101*)
```

```
    nResByte :  BYTE;
    nResWord :  WORD;
    nVar : BYTE  := 2;
END_VAR

nResByte := SHL(nInByte,nVar); (*Result is 16#14, 2#00010100*)
nResWord := SHL(nInWord,nVar); (*Result is 16#0114, 2#0000000100010100*)
```

FUP:



16.3.5.2 SHR

Der IEC-Operator dient dem bitweisen Verschieben eines Operanden nach rechts.

Syntax: erg := SHR (in, n)

in: Operand, der nach rechts verschoben wird.

n: Anzahl der Bits, um die in nach rechts verschoben wird.

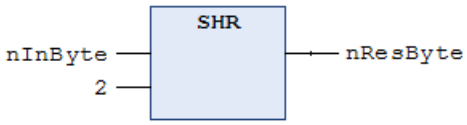
Wenn n die Datentyp-Breite überschreitet, hängt es vom Zielsystem ab, wie BYTE-, WORD-, DWORD- und LWORD-Operanden auffüllt werden. Die Zielsysteme bewirken Auffüllen mit Nullen oder mit n MOD <register width>.

Beispiele:

ST:

```
PROGRAM Shr_st
VAR
    nInByte  :  BYTE:=16#45; (*2#01000101*)
    nInWord  :  WORD:=16#0045; (*2#0000000001000101*)
    nResByte :  BYTE;
    nResWord :  WORD;
    nVar : BYTE := 2;
END_VAR

nResByte := SHR(nInByte,nVar); (*Result is 16#11, 2#00010001*)
nResWord := SHR(nInWord,nVar); (*Result is 16#0011, 2#0000000000010001*)
```

FUP:



16.3.5.3 ROL

Der IEC-Operator dient der bitweisen Rotation eines Operanden nach links.

Erlaubte Datentypen: BYTE, WORD, DWORD, LWORD

Syntax: erg := ROL (in, n)

TwinCAT verschiebt in n-mal um 1 Bit nach links und fügt gleichzeitig das Bit mit der äußersten linken Position von rechts wieder ein.

Die Anzahl der Bits, die TwinCAT für die Rechenoperation berücksichtigt, geben Sie durch den Datentyp der Eingangsvariable in vor. Wenn es sich hierbei um eine Konstante handelt, berücksichtigt TwinCAT den kleinstmöglichen Datentyp. Der Datentyp der Ausgangsvariablen bleibt ohne Auswirkung auf die Rechenoperation.

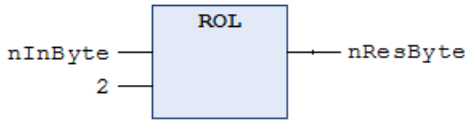
Beispiele:

Die Ergebnisse für nResByte und nResWord sind unterschiedlich, abhängig vom Datentyp der Eingangsvariablen, obwohl die Werte der Eingangsvariablen nInByte und nInWord gleich sind.

ST:

```
PROGRAM Rol_st
VAR
    nInByte  : BYTE := 16#45;
    nInWord  : WORD := 16#45;
    nResByte : BYTE;
    nResWord : WORD;
    nVar : BYTE := 2;
END_VAR

nResByte := ROL(nInByte,nVar); (*Result: 16#15*)
nResWord := ROL(nInWord,nVar); (*Result: 16#0114*)
```

FUP:



16.3.5.4 ROR

Der IEC-Operator dient der bitweisen Rotation eines Operanden nach rechts.

Erlaubte Datentypen: BYTE, WORD, DWORD, LWORD

Syntax: erg := ROR (in, n)

TwinCAT verschiebt in n-mal um 1 Bit nach rechts und fügt gleichzeitig das Bit mit der äußersten rechten Position von links wieder ein.

Die Anzahl der Bits, die TwinCAT für die Rechenoperation berücksichtigt, wird durch den Datentyp der Eingangsvariablen in vorgegeben. Handelt es sich hierbei um eine Konstante, berücksichtigt TwinCAT den kleinstmöglichen Datentyp. Der Datentyp der Ausgangsvariablen bleibt ohne Auswirkung auf die Rechenoperation.

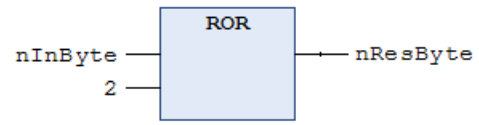
Beispiele:

Die Ergebnisse für nResByte und nResWord sind unterschiedlich, abhängig vom Datentyp der Eingangsvariablen, obwohl die Werte der Eingangsvariablen nInByte und nInWord gleich sind.

ST:

```
PROGRAM Ror_st
VAR
    nInByte  : BYTE:=16#45;
    nInWord  : WORD:=16#45;
    nResByte : BYTE;
    nResWord : WORD;
    nVar : BYTE :=2;
END_VAR

nResByte := ROR(nInByte,nVar); (*Result: 16#51*)
nResWord := ROR(nInWord,nVar); (*Result: 16#4011*)
```

FUP:



16.3.6 Bitstring-Operatoren

16.3.6.1 AND

Der IEC-Operator dient dem bitweisen AND von Bit-Operanden.

Wenn die Eingangsbits jeweils 1 sind, ist das Ausgangsbit 1, ansonsten 0.

Erlaubte Datentypen: BOOL, BYTE, WORD, DWORD, LWORD

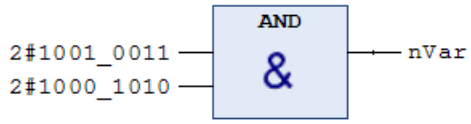
Beispiele:

Ergebnis: nVar ist 2#1000_0010.

ST:

```
nVar := 2#1001_0011 AND 2#1000_1010
```

FUP:



16.3.6.2 AND_THEN

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator AND_THEN ist nur für die Programmierung im Strukturiertem Text bei folgender Operation erlaubt: AND-Operation von Operanden des Typs BOOL und BIT mit Kurzschluss-Auswertung. Das bedeutet:

Wenn alle Operanden TRUE sind, ist das Ergebnis der Operation ebenfalls TRUE, ansonsten FALSE.

Aber: Nur wenn der erste Operand des AND_THEN Operators TRUE ist, führt TwinCAT die Ausdrücke an weiteren Operanden ebenfalls aus. Dies kann beispielsweise in Bedingungen wie IF (ptr <> 0 AND_THEN ptr^ = 99) THEN... Probleme mit Null-Pointern vermeiden.

Im Unterschied dazu wertet TwinCAT bei Verwendung des IEC-Operators AND immer alle Operanden aus.

Siehe auch:

- AND [▶ 749]

16.3.6.3 NOT

Der IEC-Operator dient dem bitweisen NOT eines Bit-Operanden.

Wenn das entsprechende Eingangsbit 0 ist, ist das Ausgangsbit 1, und umgekehrt.

Erlaubte Datentypen: BOOL, BYTE, WORD, DWORD, LWORD

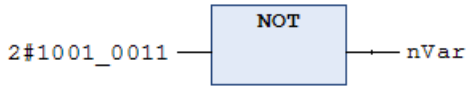
Beispiele:

Ergebnis: nVar ist 2#0110_1100.

ST:

```
nVar := NOT 2#1001_0011
```

FUP:



16.3.6.4 OR

Der IEC-Operator dient dem bitweisen OR von Bit-Operanden.

Wenn mindestens eines der Eingangs-Bits 1 ist, ist das Ausgangs-Bit 1, ansonsten 0.

Erlaubte Datentypen: BOOL, BYTE, WORD, DWORD, LWORD

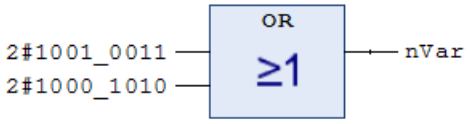
Beispiele:

Ergebnis: nVar ist 2#1001_1011.

ST:

```
nVar := 2#1001_0011 OR 2#1000_1010
```

FUP:



16.3.6.5 OR_ELSE

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator OR_ELSE ist nur für die Programmierung im Strukturiertem Text erlaubt: OR-Operation von Operanden des Typs BOOL und BIT, mit Kurzschluss-Auswertung.

Das bedeutet:

Wenn mindestens einer der Operanden TRUE ist, ist das Ergebnis der Operation ebenfalls TRUE, ansonsten FALSE.

Im Unterschied zur Verwendung des IEC-Operators OR, werden bei OR_ELSE, sobald einer der Operanden mit TRUE ausgewertet wurde, die Ausdrücke an allen weiteren Operanden nicht mehr ausgeführt.

Beispiel:

```
Function_Block FB_Sample
VAR
    nCounter : INT;
END_VAR

METHOD TestMethod : BOOL
nCounter := nCounter + 1;
TestMethod := TRUE;

PROGRAM MAIN
VAR
    fbSampleOr : FB_Sample;
    fbSampleOrElse : FB_Sample;
    bResult : BOOL;
    bVar : BOOL;
END_VAR

bResult : bVar OR fbSampleOr.TestMethod();
// Counter of fbSampleOr increases as the method is executed

bResult := bVar OR_ELSE fbSampleOrElse.TestMethod();
//Counter of fbSampleOrElse does not increases as the method is not executed
```

bVar ist TRUE, somit ist das Ergebnis bResult sowohl bei der Verwendung von OR als auch bei der Verwendung OR_ELSE ebenfalls TRUE.

Der Unterschied zwischen den beiden Operatoren liegt darin, dass der zweite Operand (der Aufruf der Methode) nur bei der Verwendung von OR ausgeführt wird. In diesem Fall wird der Zähler der FB_Instanz fbSampleOr inkrementiert.

Da der erste Operand bVar1 bereits TRUE liefert, wird bei der Verwendung von OR_ELSE der zweite Operand nicht mehr ausgeführt, sodass die Methode der FB_Instanz fbSampleOrElse nicht aufgerufen und er Zähler somit nicht hochgezählt wird.

Siehe auch:

- OR [▶ 750]

16.3.6.6 XOR

Der IEC-Operator dient der bitweisen XOR-Operation von Bit-Operanden.

Wenn nur einer der beiden Eingangs-Bits den Wert 1 liefert, wird das Ausgangs-Bit 1; wenn beide Eingänge 1 oder beide 0 sind, wird der Ausgang 0.

Erlaubte Datentypen: BOOL, BYTE, WORD, DWORD, LWORD

Beachten Sie folgendes Verhalten des XOR-Bausteins in erweiterter Form, also bei mehr als 2 Eingängen: TwinCAT vergleicht die Eingänge paarweise und die jeweiligen Ergebnisse dann wiederum gegeneinander (entspricht der Norm, jedoch nicht unbedingt der Erwartung).

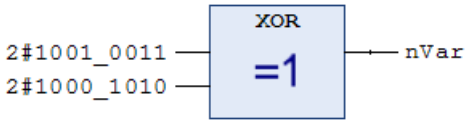
Beispiele:

Ergebnis: nVar ist 2#0001_1001.

ST:

```
nVar := 2#1001_0011 XOR 2#1000_1010
```

FUP:



16.3.7 Namensraumroperatoren

Die Namensraumoperatoren sind eine Erweiterung der IEC 61131-3 Operatoren. Sie bieten Ihnen Möglichkeiten, den Zugriff auf Variablen oder Module eindeutig zu gestalten, auch wenn Sie den gleichen Variablen- oder Modulnamen im Projekt mehrmals verwenden.

16.3.7.1 Globaler Namensraum

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Ein Instanzpfad, der mit einem Punkt . beginnt, öffnet immer einen globalen Namensraum. Wenn es also eine lokale Variable mit dem gleichen Namen <varname> gibt wie eine globale Variable, sprechen Sie mit .<varname> die globale Variable an.

16.3.7.2 Namensraum für Globale Variablenlisten

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Den Namen einer globalen Variablenliste (GVL) können Sie als Namensraum-Bezeichner für die in der Liste definierten Variablen verwenden. Dadurch ist es möglich, Variablen mit gleichen Namen in verschiedenen globalen Variablenlisten zu verwenden und dennoch eindeutig auf eine bestimmte Variable zuzugreifen. Dem Variablennamen stellen Sie den Namen der globalen Variablenliste getrennt durch einen Punkt . voran.

Syntax: <global variable list name>.<variable>

Beispiel:

```
GVL1.nVar := GVL2.nVar;
```

Die globalen Variablenlisten GVL1 und GVL2 enthalten jeweils eine Variable nVar. TwinCAT kopiert die globale Variable nVar aus Liste GVL2 in nVar aus der Liste GVL1.

Wenn Sie eine Variable, die in mehreren globalen Variablenlisten deklariert ist, ohne den vorangestellten Listennamen referenzieren, erscheint eine Fehlermeldung.

16.3.7.3 Bibliotheksnamensraum

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Syntax:

<Bibliotheksnamensraum>.<Bibliotheksbausteinbezeichner>

Ein Bibliotheksbausteinbezeichner wird (als Präfix mit einem Punkt getrennt) um den Bibliotheksnamensraum erweitert, um eindeutig und qualifiziert auf den Bibliotheksbaustein zuzugreifen. Üblicherweise stimmen Namensraum und Name einer Bibliothek überein.

Beispiel:

Eine Bibliothek ist in einem Projekt eingebunden und enthält einen Baustein FB_Sample. In dem Projekt ist aber bereits lokal ein Funktionsbaustein mit gleichem Namen instanziiert. Bezeichnen Sie den Bibliotheksbaustein mit libA.fbSample, um nicht auf den lokalen Funktionsbaustein sondern auf den Bibliotheksbaustein zuzugreifen.

```
nVar1 := fbSample(nIn := 12); // Call of the project function block FB_Sample
nVar2 := lib.fbSample(nIn := 22); // Call of the library function block FB_Sample
```

Sie können für den Namenraum einen anderen Bezeichner definieren. Dafür tragen Sie bereits (als Bibliotheksentwickler) beim Erstellen eines Bibliotheksprojekts in den Projektinformationen einen Namenraum ein. Oder Sie bestimmen beim Erstellen eines SPS-Projekts im Bibliotheksverwalter für eine Bibliothek in der Ansicht **Eigenschaften** einen speziellen Namensraum.

16.3.7.4 Enumerationsnamensraum

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Sie können den TYPE-Namen einer Enumeration für einen eindeutigen Zugriff auf eine Enumerations-Konstante verwenden. Somit können Sie Konstanten gleichen Namens in mehreren Enumerationen verwenden.

Der Enumerationsname wird dem Konstantennamen getrennt durch einen Punkt . vorangestellt.

Syntax: <Enumerationsname>.<Konstantenname>

Beispiel:

Die Konstante eBlue ist sowohl eine Komponente der Enumeration E_Colors als auch eine der Enumeration E_Feelings.

```
eColor := E_Colors.eBlue;  // Access to coomponent Blue in enumeration Colors
eFeeling := E_Feelings.eBlue; // Acces to component Blue in enumeration Feeling
```

16.3.8 Numerische Operatoren

16.3.8.1 ABS

Der IEC-Operator liefert den Absolutwert einer Zahl.

Erlaubte Datentypen für die Eingangs- und Ausgangsvariablen und Zahlenkonstanten: beliebiger numerischer Basisdatentyp

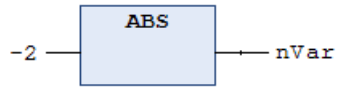
Beispiele:

Ergebnis: nVar ist 2.

ST:

```
nVar := ABS(-2);
```

FUP:



16.3.8.2 ACOS

Der IEC-Operator liefert den Arcuscosinus (Umkehrfunktion von Cosinus) für eine Zahl. Der Wert wird in Bogenmaß errechnet.

Erlaubte Datentypen für die Eingangsvariable, die den Winkel im Bogenmaß angibt: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariable: REAL und LREAL

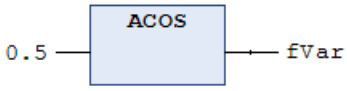
Beispiele:

Ergebnis: fVar ist 1.0472.

ST:

```
fVar := ACOS(0.5);
```

FUP:



16.3.8.3 ASIN

Der IEC-Operator liefert den Arcussinus (Umkehrfunktion des Sinus) für eine Zahl.

Erlaubte Datentypen für die Eingangsvariable: Beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariablen: REAL und LREAL

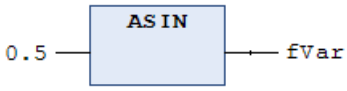
Beispiele:

Ergebnis: fVar ist 0.523599.

ST:

```
fVar := ASIN(0.5);
```

FUP:



16.3.8.4 ATAN

Der IEC-Operator liefert den Arcustangens (Umkehrfunktion von Tangens) einer Zahl. Der Wert wird in Bogenmaß errechnet.

Erlaubte Datentypen für die Eingangsvariable, die den Winkel im Bogenmaß angibt: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariablen: REAL und LREAL

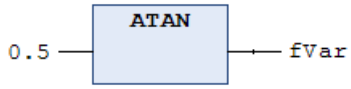
Beispiele:

Ergebnis: fVar ist 0.463648.

ST:

```
fVar := ATAN(0.5);
```

FUP:



16.3.8.5 COS

Der IEC-Operator liefert den Cosinuswert für eine Zahl.

Erlaubte Datentypen für die Eingangsvariable, die den Winkel im Bogenmaß angibt: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariablen: REAL und LREAL

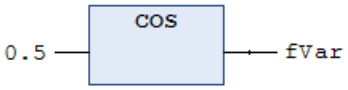
Der erlaubte Bereich für den Eingangswert liegt von -2^{63} bis $+2^{63}$.
Auf x86- und x64-Systemen gilt: Wenn der Eingangswert außerhalb des erlaubten Bereichs liegt, liefert die Funktion den Eingangswert zurück.

Beispiele:

ST:

```
fVar := COS(0.5);
```

FUP:



Ergebnis: fVar ist 0.877583.

16.3.8.6 EXP

Der IEC-Operator liefert die Exponentialfunktion.

Erlaubte Datentypen für die Eingangsvariablen: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für die Ausgangsvariablen: REAL und LREAL

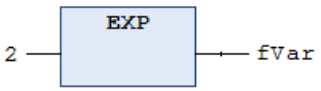
Beispiele:

Ergebnis: fVar ist 7.389056099.

ST:

```
fVar := EXP(2);
```

FUP:



16.3.8.7 EXPT

Der IEC-Operator potenziert eine Zahl mit einer anderen und liefert die Potenz von Basis hoch Exponent zurück: power = base^{exponent}. Dabei sind sowohl Basis als auch Exponent Eingabewerte (Parameter). Die Potenzfunktion ist nicht definiert, wenn die Basis 0 und gleichzeitig der Exponent negativ ist. Das Verhalten in diesem Fall ist jedoch plattformabhängig.

Syntax: EXPT(<Basis>,<Exponent>)

Erlaubte Datentypen für die Eingabewerte: Numerische Basisdatentypen (SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, LWORD)

Erlaubte Datentypen für den Rückgabewert: Gleitkommazahltypen (REAL, LREAL).

Wenn das Ergebnis in der REAL-Variablen gespeichert wird, wird eine Warnung für die Konvertierung von LREAL nach REAL erzeugt. --> Rückgabetyp ist immer LREAL.

Z.B. wird eine Warnung für den folgenden PLC-Code erzeugt:

```
real1 := EXPT(real2,INT#2);
```

Beispiele:

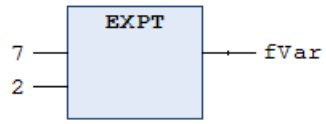
a) Potenzfunktion mit Literalen

Ergebnis: fVar ist 49.

ST:

```
fVar := EXPT(7,2);
```

FUP:



b) Potenzfunktion mit Variablen

Ergebnis: fPow ist 128.

```
PROGRAM MAIN
VAR
    fPow      : LREAL;
    nBase     : INT := 2;
    nExponent : INT := 7;
END_VAR

fPow := EXPT(nBase, nExponent);
```

16.3.8.8 LN

Der IEC-Operator liefert den natürlichen Logarithmus einer Zahl.

Erlaubte Datentypen für die Eingangsvariablen: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariablen: REAL und LREAL

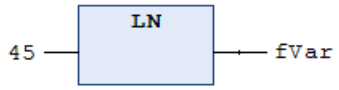
Beispiele:

Ergebnis: fVar ist 3.80666.

ST:

```
fVar := LN(45);
```

FUP:



16.3.8.9 LOG

Der IEC-Operator liefert den Logarithmus zur Basis 10 einer Zahl.

Erlaubte Datentypen für die Eingangsvariablen: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für die Ausgangsvariable: REAL oder LREAL

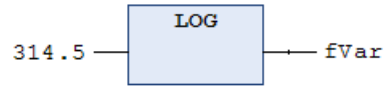
Beispiele:

Ergebnis: fVar ist 2.49762.

ST:

```
fVar := LOG(314.5);
```

FUP:



16.3.8.10 SIN

Der IEC-Operator liefert den Sinus-Wert für eine Zahl.

Erlaubte Datentypen für die Eingangsvariable, die den Winkel im Bogenmaß angibt: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariable: REAL und LREAL

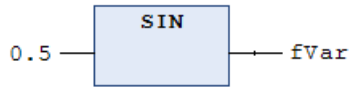
Der erlaubte Bereich für den Eingangswert liegt von -2^{63} bis $+2^{63}$.
Auf x86- und x64-Systemen gilt: Wenn der Eingangswert außerhalb des erlaubten Bereichs liegt, liefert die Funktion den Eingangswert zurück.

Beispiele:

ST:

```
fVar := SIN(0.5);
```

FUP:



Ergebnis: fVar ist 0.479426.

16.3.8.11 SQRT

Der IEC-Operator liefert die Quadratwurzel einer Zahl.

Erlaubte Datentypen für die Eingangsvariablen: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für die Ausgangsvariablen: REAL oder LREAL

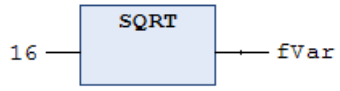
Beispiele:

Ergebnis: fVar ist 4.

ST:

```
fVar := SQRT(16);
```

FUP:



16.3.8.12 TAN

Der IEC-Operator liefert den Tangens-Wert für eine Zahl.

Erlaubte Datentypen für die Eingangsvariable, die den Winkel im Bogenmaß angibt: beliebiger numerischer Basisdatentyp

Erlaubte Datentypen für Ausgangsvariablen: REAL und LREAL

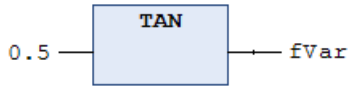
Beispiele:

Ergebnis: fVar ist 0.546302.

ST:

```
fVar := TAN(0.5);
```

FUP:



16.3.9 Typkonvertierungsoperatoren

Sie können explizit Typkonvertierungsoperatoren aufrufen. Für getypte Konvertierungen von einem elementaren Typ in einen anderen elementaren Typ und auch für Überladungen stehen die unten beschriebenen Typkonvertierungsoperatoren zur Verfügung. Konvertierungen von einem „kleineren“ Typ auf einen „größeren“ Typ, wie beispielsweise von BYTE nach INT oder von WORD nach DINT, sind aber auch implizit möglich.

Getypte Konvertierung: `<elementary data type>_TO_<another elementary data type>`

Überladene Konvertierung: `TO_<elementary data type>`

Elementare Datentypen:

```
<elementary data type> =
__UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DINT | DT | DWORD | INT | LDATE | LDT | LINT
| LREAL | LTIME | LTOD | LWORD | REAL | SINT | TIME | TOD | UDINT | UINT | ULINT | USINT | WORD
```

Die Schlüsselwörter TIME_OF_DAY und DATE_AND_TIME sind alternative Schreibweisen für die Datentypen TOD und DT. TIME_OF_DAY und DATE_AND_TIME werden nicht als Typkonvertierungsbefehl abgebildet.

Wenn bei einem Typkonvertierungsoperator der Operandenwert außerhalb des Wertebereichs des Zieldatentyps liegt, ist die Ergebnisausgabe undefiniert. Dies ist beispielsweise der Fall, wenn ein negativer Operandenwert von LREAL in den Zieldatentyp UINT konvertiert wird.

Bei der Typkonvertierung von größeren auf kleinere Typen können Informationen verloren gehen.

Stringmanipulation bei Konvertierung nach STRING oder WSTRING

Bei einer Typkonvertierung nach STRING oder WSTRING wird der getypte Wert als Zeichenfolge linksbündig abgelegt und bei Überlänge abgeschnitten. Deklarieren Sie deshalb die Rückgabevariablen für die Typkonvertierungsoperatoren <type>_TO_STRING und <type>_TO_WSTRING ausreichend lang, so dass die Zeichenfolge ohne Manipulation Platz findet.

Siehe auch:

- Datentypen [▶ 793]

16.3.9.1 Überladung

Die Operatoren konvertieren Werte in andere Datentypen, wobei explizit nur ein Zieldatentyp und kein Anfangsdatentyp (Datentyp des Operanden) angegeben wird („überladene Konvertierung“).

Überladungen sind nicht Teil der IEC 61131-3. Wenn Sie strikt nach IEC61131-3 programmieren möchten, verwenden Sie bitte die in den folgenden Abschnitten beschriebenen Operatoren des Schemas <type> _TO_ <another type>.

Syntax:

```
<variable name> := <TO operator> ( <operand> );
<operand> = <variable name> | <literal>
```

Operatoren:

```
TO___UXINT
TO___XINT
TO___XWORD
TO_BIT
TO_BYTE
TO_BOOL
TO_DATE
TO_DINT
TO_DT
TO_DWORD
TO_INT
TO_LDATE
TO_LDT
TO_LINT
TO_LREAL
TO_LTIME
TO_LTOD
TO_LWORD
TO_REAL
TO_SINT
TO_STRING
TO_TIME
TO_TOD
TO_UDINT
TO_UINT
TO_ULINT
TO_USINT
TO_WORD
TO_WSTRING
```

Die Regeln für die getypten Konvertierungen gelten auch bei der Überladung.

Wenn der Eingangswert eines Typkonvertierungsoperators außerhalb des Wertebereichs des Ausgangsdatentyps liegt, ist das Ergebnis der Operation nicht definiert und abhängig von der Plattform. Dies ist zum Beispiel der Fall, wenn der Eingangswert für die Konvertierung von LREAL nach UDINT negativ ist.

Beispiele:

Implementierungssprache ST:

```
VAR
    nNumber1 : INT;
    nNumber2 : INT;
    fNumber3 : REAL     := 123.456;
    bTruth   : BOOL;
    sText1   : STRING;
    sText2   : STRING   := 'Hello World!';
    wsText   : WSTRING;
    dEvent   : DATE     := D#2019-9-3;
    nEvent   : UINT;
    nData    : __UXINT;
END_VAR
```

```
nNumber1 := TO_INT(4.22);           // Result: 4
nNumber2 := TO_INT(fNumber1);       // Result: 123
bTruth   := TO_BOOL(1);             // Result: TRUE
sText1   := TO_STRING(342);         // Result: '342'
wsText   := TO_WSTRING(sText2);     // Result: "Hello World!"
nEvent   := TO_UINT(dEvent);        // Result: 44288
dEvent   := TO__UXINT(nNumber2);    // Result: 123
```

Siehe auch:

- Datentypen [▶ 793]

16.3.9.2 <type>_TO_BOOL

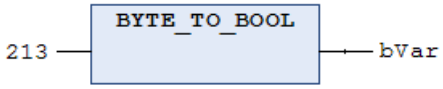
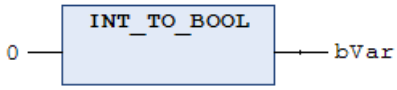
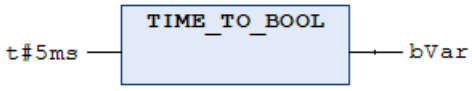
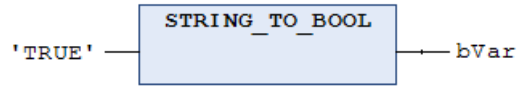
Der IEC-Operator dient der Konvertierung von einem anderen Datentyp in den Datentyp BOOL.

Syntax: <Datentyp>_TO_BOOL

Das Ergebnis ist TRUE, wenn der Operand ungleich 0 ist. Das Ergebnis ist FALSE, wenn der Operand gleich 0 ist. Beim Datentyp STRING ist das Ergebnis TRUE, wenn der Operand TRUE ist, ansonsten ist das Ergebnis FALSE.

Beispiele:

ST-Code	Ergebnis
`bVar := BYTE_TO_BOOL(2#11010101);`	TRUE
`bVar := INT_TO_BOOL(0);`	FALSE
`bVar := TIME_TO_BOOL(T#5ms);`	TRUE
`bVar := STRING_TO_BOOL('TRUE');`	TRUE

FUP-Code	Ergebnis
	TRUE
	FALSE
	TRUE
	TRUE

Siehe auch:

- Datentypen > BOOL [▶ 794]

16.3.9.3 BOOL_TO_<type>

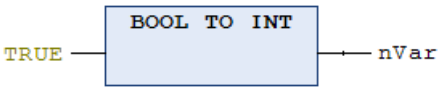
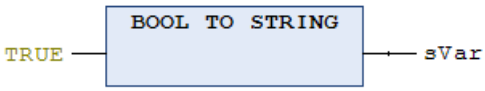
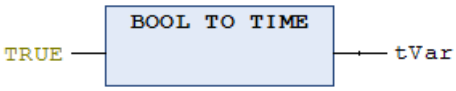
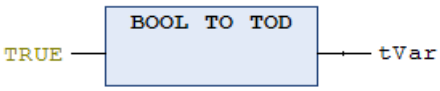
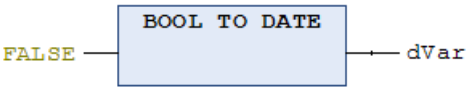
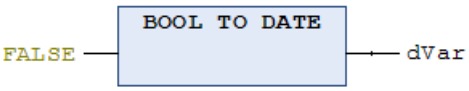
Der IEC-Operator dient der Konvertierung vom Datentyp BOOL in einen anderen Datentyp.

Syntax: BOOL_TO_<Datentyp>

Bei Zahlendatentypen ist das Ergebnis 1, wenn der Operand TRUE ist, und 0, wenn der Operand FALSE ist. Beim Datentyp STRING ist das Ergebnis TRUE oder FALSE.

Beispiele:

ST-Code	Ergebnis
`nVar := BOOL_TO_INT(TRUE);`	1
`sVar := BOOL_TO_STRING(TRUE);`	'TRUE'
`tVar := BOOL_TO_TIME(TRUE);`	T#1ms
`tVar := BOOL_TO_TOD(TRUE);`	TOD#00:00:00.001
`dVar := BOOL_TO_DATE(FALSE);`	D#1970
`dtVar := BOOL_TO_DT(TRUE);`	DT#1970-01-01-00:00:01

FUP-Code	Ergebnis
	1
	'TRUE'
	T#1ms
	TOD#00:00:00.001
	D#1970-01-01
	DT#1970-01-01-00:00:01

Siehe auch:

- Datentypen > BOOL [▶ 794]

16.3.9.4 <INT type>_TO_<INT type>

Der IEC-Operator dient der Konvertierung von einem ganzzahligen Datentyp in einen anderen ganzzahligen Datentyp.

Syntax: <INT-Datentyp>_TO_<INT-Datentyp>

Ganzzahlige Datentypen:

- BYTE
- WORD, DWORD, LWORD
- SINT, INT, DINT, LINT
- USINT, UINT, UDINT, ULINT

Bei der Typkonvertierung von größeren zu kleineren Datentypen können Informationen verloren gehen. Wenn die zu konvertierende Zahl die Bereichsgrenze überschreitet, berücksichtigt TwinCAT die ersten Bytes der Zahl nicht.

Beispiele:

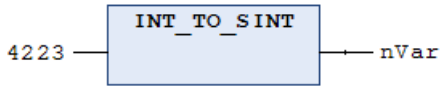
Ergebnis: nVar ist 127.

ST:

```
nVar := INT_TO_SINT(4223);
```

Wenn Sie die Integer-Zahl 4223 (16#107f in Hexadezimaldarstellung) in eine SINT-Variable speichern, dann enthält diese Variable die Zahl 127 (16#7f in Hexadezimaldarstellung).

FUP:



Siehe auch:

- Datentypen > Ganzzahlige Datentypen [▶ 794]

16.3.9.5 REAL/LREAL_TO_<type>

Der IEC-Operator dient der Konvertierung vom Datentyp REAL und LREAL in einen anderen Datentyp.

Syntax:

REAL_TO_<Datentyp>

LREAL_TO_<Datentyp>

TwinCAT rundet den Real-Wert des Operanden nach oben oder unten auf einen ganzzahligen Wert und wandelt in den entsprechenden Datentyp um. Ausgenommen davon sind die Datentypen STRING, BOOL, REAL und LREAL.

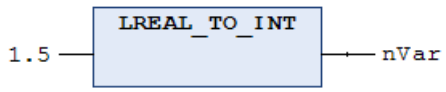
Wenn Sie ein eine Zahl vom Typ REAL oder LREAL zu SINT, USINT, INT, UINT, DINT, UDINT, LINT oder ULINT konvertieren und der Wert der REAL/LREAL-Zahl außerhalb des Wertebereichs des Integer liegt, erhalten Sie ein undefiniertes Ergebnis. Auch ein Ausnahmefehler ist dann möglich!

Beachten Sie bei der Konvertierung in den Datentyp STRING, dass die Gesamt-Kommastellen-Zahl auf 16 begrenzt ist. Enthält die (L)REAL-Zahl mehr Stellen, wird die sechzehnte Stelle gerundet und so im String dargestellt. Wenn der String für die Zahl zu kurz definiert ist, schneidet TwinCAT von rechts her ab.

Bei der Typkonvertierung von größeren zu kleineren Datentypen können Informationen verloren gehen.

Beispiele:

ST-Code	Ergebnis
`nVar1 := REAL_TO_INT(1.5);`	2
`nVar2 := REAL_TO_INT(1.4);`	1
`nVar1 := REAL_TO_INT(-1.5);`	-2
`nVar2 := REAL_TO_INT(-1.4);`	-1

FUP-Code	Ergebnis
	2

Siehe auch:

- Datentypen > REAL/LREAL [▶ 795]

16.3.9.6 STRING_TO_<type>

Der IEC-Operator dient der Konvertierung vom Datentyp STRING in einen anderen Datentyp-

Syntax: STRING_TO_<Datentyp>

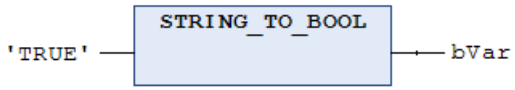
Sie müssen den Operand vom Typ STRING gemäß der Norm IEC 61131-3 angeben. Der Wert muss einer gültigen Konstante (Literal) des Zieldatentyps entsprechen. Dies betrifft die Angabe von Exponentialwerten, infiniten Werten, Präfixen, Gruppierungszeichen ("_") und Komma. Zusätzliche Zeichen hinter den Ziffern einer Zahl sind erlaubt, z B. 23xy. Zusätzliche Zeichen vor einer Zahl sind nicht erlaubt.

Der Operand muss einen gültigen Wert des Zieldatentyps darstellen.

Wenn der Datentyp des Operanden nicht zum Zieldatentyp passt oder der Wert außerhalb des Bereichs des Zieldatentyps liegt, ist die Ergebnisausgabe undefiniert. Bei der Typkonvertierung von größeren zu kleineren Datentypen können Informationen verloren gehen.

Beispiele:

ST-Code	Ergebnis
`bVar := STRING_TO_BOOL('TRUE');`	TRUE
`nVar := STRING_TO_WORD('abc34');`	0
`nVar := STRING_TO_WORD('34abc');`	34
`tVar := STRING_TO_TIME('T#127ms');`	T#127ms
`fVar := STRING_TO_REAL('1.234');`	1.234
`nVar := STRING_TO_BYTE('500');`	244

FUP-Code	Ergebnis
	TRUE

Siehe auch:

- Datentypen > STRING [▶ 796]

16.3.9.7 TO_STRING/TO_WSTRING für Enumerationsvariablen

Wenn Sie die textuellen Bezeichner einer Enumerationskomponente abfragen möchten, um diesen zum Beispiel in einer Textausgabe weiterzuverarbeiten, fügen Sie oberhalb der Deklaration der Enumeration das Attribut 'to_string' [▶ 874] hinzu. Im Implementierungsteil können Sie dann auf eine Enumerationsvariable oder auf die Komponenten der Enumeration die Konvertierungsfunktionen TO_STRING bzw. TO_WSTRING anwenden und bekommen dabei den Namen der Enumerationskomponente zurückgeliefert.

Attribut 'to_string'

Verfügbar ab TC3.1 Build 4024

Die Konvertierungsfunktionen TO_STRING/TO_WSTRING können Sie auch auf Enumerationen anwenden, die nicht mit dem Attribut 'to_string' deklariert sind. In diesem Fall bekommen Sie den numerischen Wert der Enumerationskomponente zurückgeliefert.

Beispiel:

Enumeration E_Sample

```
{attribute 'qualified_only'}
{attribute 'strict'}
{attribute 'to_string'}
TYPE E_Sample :
(
    eInit := 0,
    eStart,
```

```
    eStop
);
END_TYPE
```

Programm MAIN

```
PROGRAM MAIN
VAR
    eSample         : E_Sample;
    nCurrentValue   : INT;
    sCurrentValue   : STRING;
    wsCurrentValue  : WSTRING;

    sComponent      : STRING;
    wsComponent     : WSTRING;
END_VAR
```

```
nCurrentValue  := eSample;
sCurrentValue  := TO_STRING(eSample);
wsCurrentValue := TO_WSTRING(eSample);

sComponent     := TO_STRING(E_Sample.eStart);
wsComponent    := TO_WSTRING(E_Sample.eStop);
```

Ergebnis der Zuweisungen/Konvertierungsfunktionen:

- Wert von nCurrentValue: 0
- Wert von sCurrentValue: 'eInit'
- Wert von wsCurrentValue: "eInit"
- Wert von sComponent: 'eStart'
- Wert von wsComponent: "eStop"

Ergebnis, falls die Enumeration nicht mit dem Attribut 'to_string' deklariert wäre:

- Wert von nCurrentValue: 0
- Wert von sCurrentValue: '0'
- Wert von wsCurrentValue: "0"
- Wert von sComponent: '1'
- Wert von wsComponent: "2"

Siehe auch:

- Aufzählungen / Enumerationen [▶ 816]

16.3.9.8 TIME/TOD_TO_<type>

Der IEC-Operator dient der Konvertierung vom Datentyp TIME oderTIME_OF_DAY (TOD) in einen anderen Datentyp.

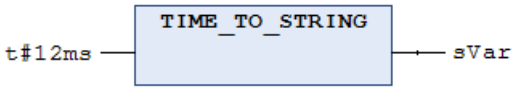
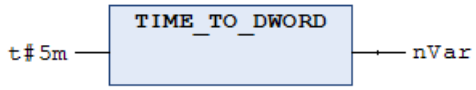
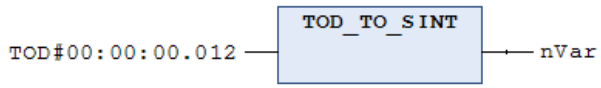
Syntax: <TIME-Datentyp>_TO_<Datentyp>

Intern speichert TwinCAT die Zeit in einem DWORD in Millisekunden ab (bei TIME_OF_DAY seit 00:00 Uhr). Diesen Wert konvertiert TwinCAT. Beim Datentyp STRING ist das Ergebnis die Zeitkonstante.

Bei der Typkonvertierung von größeren zu kleineren Datentypen können Informationen verloren gehen.

Beispiele:

ST-Code	Ergebnis
`sVar := TIME_TO_STRING(T#12ms);`	T#12ms
`nVar := TIME_TO_DWORD(T#5m);`	300000
`nVar := TOD_TO_SINT(TOD#00:00:00.012);`	12

FUP-Code	Ergebnis
	T#12ms
	300000
	12

Siehe auch:

- Datentypen > Datums- und Uhrzeitdatentypen [▶ 797]

16.3.9.9 DATE/DT_TO_<type>

Der IEC-Operator dient der Konvertierung vom Datentyp DATE oder DATE_AND_TIME (DT) in einen anderen Datentyp.

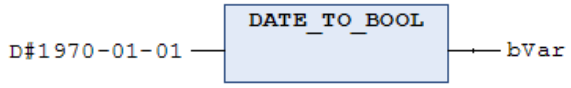
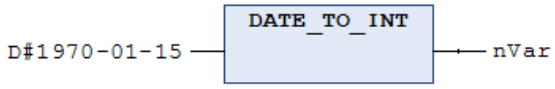
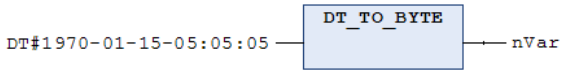
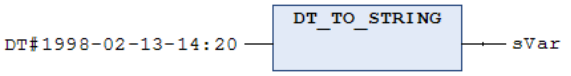
Syntax: <DATE-Datentyp>_TO_<Datentyp>

Intern speichert TwinCAT das Datum in einem DWORD in Sekunden seit dem 1.Januar 1970. Diesen Wert konvertiert TwinCAT. Beim Datentyp STRING ist das Ergebnis die Datumskonstante.

Bei der Typkonvertierung von größeren zu kleineren Datentypen können Informationen verloren gehen.

Beispiele:

ST-Code	Ergebnis
`bVar := DATE_TO_BOOL(D#1970-01-01);`	FALSE
`nVar := DATE_TO_INT(D#1970-01-15);`	29952
`nVar := DT_TO_BYTE(DT#1970-01-15-05:05:05);`	129
`sVar := DT_TO_STRING(DT#1998-02-13-14:20);`	'DT#1998-02-13-14:20:00'

FUP-Code	Ergebnis
	FALSE
	29952
	129
	'DT#1998-02-13-14:20:00'

Siehe auch:

- Datentypen > Datums- und Uhrzeitdatentypen [▶ 797]

16.3.9.10 TRUNC

Der IEC-Operator dient der Konvertierung vom Datentyp REAL in den Datentyp DINT. TwinCAT nimmt nur den ganzzahligen Anteil der Zahl.

In TwinCAT 2.x PLC Control konvertiert der TRUNC-Operator von REAL nach INT. Wenn Sie ein TwinCAT 2.x PLC-Projekt importieren, ersetzt TwinCAT TRUNC automatisch durch TRUNC_INT.

Wenn TwinCAT den Eingangswert nicht durch einen DINT oder INT darstellen kann, ist das Ergebnis dieser Funktion undefiniert.

Wenn bei einem Typkonvertierungsoperator der Operandenwert außerhalb des Wertebereichs des Zieldatentyps liegt, ist die Ergebnisausgabe undefiniert. Dies ist beispielsweise der Fall, wenn ein negativer Operandenwert von LREAL in den Zieldatentyp UINT konvertiert wird.

Bei der Typkonvertierung von größeren auf kleinere Typen können Informationen verloren gehen.

Beispiele:

Ergebnis: nVar1 ist 1.

ST:

```
nVar1 := TRUNC(1.9);  (* Result: 1 *)
nVar2 := TRUNC(-1.4); (* Result: -1 *)
```

Siehe auch:

- Datentypen [▶ 793]

16.3.9.11 TRUNC_INT

Der IEC-Operator dient der Konvertierung vom Datentyp REAL in Datentyp INT. TwinCAT nimmt nur den Betrag des ganzzahligen Anteils der Zahl.

TRUNC_INT entspricht dem Operator TRUNC in TwinCAT 2.x PLC und wird beim Import von TwinCAT 2.x PLC-Projekten automatisch an dessen Stelle verwendet. Beachten Sie die veränderte Funktion von TRUNC.

Das Ergebnis dieser Funktion ist undefiniert, wenn TwinCAT den Eingangswert nicht durch einen DINT oder INT darstellen kann.

Wenn bei einem Typkonvertierungsoperator der Operandenwert außerhalb des Wertebereichs des Zieldatentyps liegt, ist die Ergebnisausgabe undefiniert. Dies ist beispielsweise der Fall, wenn ein negativer Operandenwert von LREAL in den Zieldatentyp UINT konvertiert wird.

Bei der Typkonvertierung von größeren auf kleinere Typen können Informationen verloren gehen.

Beispiele:

Ergebnis: nVar1 ist 1.

ST:

```
nVar1 := TRUNC_INT(1.9);  (* Result: 1 *)
nVar2 := TRUNC_INT(-1.4); (* Result: -1 *)
```

Siehe auch:

- Datentypen [▶ 793]

16.3.10 Vergleichsoperatoren

Die Vergleichsoperatoren sind boolesche Operatoren, die jeweils zwei Eingänge (erster und zweiter Operand) miteinander vergleichen.

16.3.10.1 EQ

Der IEC-Operator dient der Funktion „Gleichheit“.

Erlaubte Datentypen der Operanden: beliebiger Basisdatentyp.

Wenn die Operanden gleich sind, liefert der Operator das Ergebnis TRUE, ansonsten FALSE.

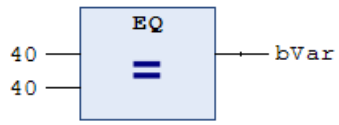
Beispiele:

Ergebnis: bVar ist TRUE.

ST:

```
bVar := 40 = 40;
```

FUP:



16.3.10.2 GE

Der IEC-Operator dient der Funktion „größer oder gleich“.

Erlaubte Datentypen der Operanden: beliebiger Basisdatentyp.

Wenn der erste Operand größer als der zweite Operand oder gleich groß wie der zweite Operand ist, liefert der Operator das Ergebnis TRUE, ansonsten FALSE.

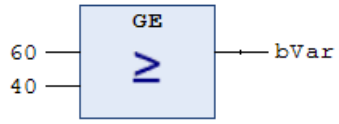
Beispiele:

Ergebnis: bVar ist TRUE.

ST:

```
bVar := 60 >= 40;
```

FUP:



16.3.10.3 GT

Der IEC-Operator dient der Funktion „Größer als“.

Erlaubte Datentypen der Operanden: beliebiger Basisdatentyp.

Wenn der erste Operand größer als der zweite Operand ist, liefert der Operator das Ergebnis TRUE, ansonsten FALSE.

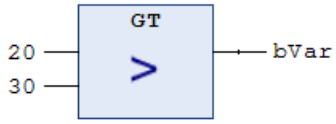
Beispiele:

Ergebnis: bVar ist FALSE.

ST:

```
bVar := 20 > 30;
```

FUP:



16.3.10.4 LE

Der IEC-Operator dient der Funktion „Kleiner oder gleich“.

Erlaubte Datentypen der Operanden: beliebiger Basisdatentyp.

Wenn der erste Operand kleiner als der zweite Operand oder gleich groß wie der zweite Operand ist, liefert der Operator das Ergebnis TRUE, ansonsten FALSE.

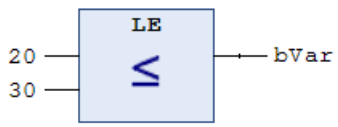
Beispiele:

Ergebnis: bVar ist TRUE.

ST:

```
bVar := 20 <= 30;
```

FUP:



16.3.10.5 LT

Der IEC-Operator dient der Funktion „Kleiner als“.

Erlaubte Datentypen der Operanden: beliebiger Basisdatentyp.

Wenn der erste Operand kleiner als der zweite ist, liefert der Operator das Ergebnis TRUE, ansonsten FALSE.

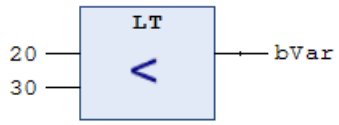
Beispiele:

Ergebnis: bVar ist TRUE.

ST:

```
bVar := 20 < 30;
```

FUP:



16.3.10.6 NE

Der IEC-Operator dient der Funktion „Ungleichheit“.

Erlaubte Datentypen der Operanden: beliebiger Basisdatentyp.

Wenn die Operanden ungleich sind, liefert der Operator das Ergebnis TRUE, ansonsten FALSE.

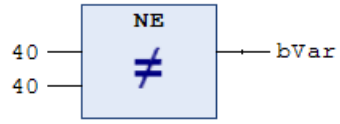
Beispiele:

Ergebnis: bVar ist FALSE.

ST:

```
bVar := 40 <> 40;
```

FUP:



16.3.11 Weitere Operatoren

16.3.11.1 __NEW

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der __NEW-Operator reserviert dynamisch Speicher um Funktionsbausteine, um benutzerdefinierte Datentypen oder Arrays von Standarddatentypen zu instanziieren. Der Operator gibt einen passend getypten Pointer zurück.

Syntax:

```
<pointer name> := __NEW( <type> ( , <size> )? );
__DELETE( <pointer name> );
```

```
<type> : <function block> | <data unit type> | <standard data type>
// (...)? : Optional
```

Beispiel:

```
pScalarType := __NEW(ScalarType, length);
```

Der Operator __NEW erzeugt eine Instanz des Typs <type> und gibt einen Pointer auf diese Instanz zurück. Anschließend wird die Initialisierung der Instanz aufgerufen. Wenn es sich bei <type> um einen skalaren Standarddatentyp handelt, wird zusätzlich der optionale Operand <size> ausgewertet. Der Operator erzeugt dann ein Array des Typs <standard data type> der Länge <size> (Elementanzahl). Wenn der Versuch Speicher zu allozieren scheitert, gibt __NEW den Wert 0 zurück.

Verwenden Sie den __NEW-Operator innerhalb einer Zuweisung („:=“), ansonsten wird eine Fehlermeldung ausgegeben.

Keine Typänderung per Online-Change möglich

Ein Funktionsbaustein oder ein benutzerdefinierter Datentyp, dessen Instanz mit __NEW dynamisch erzeugt wird, belegt einen fixen Speicherbereich. Er kann sein Datenlayout durch Online-Change nicht ändern. Somit können keine neuen Variablen hinzugefügt und keine gelöscht werden, und es können keine Typen geändert werden. Dadurch ist sichergestellt, dass der Pointer auf dieses Objekt auch nach dem Online-Change gültig ist.

Aus diesem Grund kann der Operator __NEW nur auf Funktionsbausteine/DUTs aus Bibliotheken und auf Funktionsbausteine/DUTs mit dem Attribut 'enable_dynamic_creation' [▶ 836] angewendet werden. Wenn die Schnittstelle eines solchen Funktionsbausteins/DUTs geändert wird, gibt TwinCAT eine Fehlermeldung aus.

Dynamischer Speicher wird aus dem Routerspeicherpool alloziert.

Statusinformationen des TwinCAT-Routers

Mithilfe des Funktionsbausteins FB_GetRouterStatusInfo aus der Tc2_Utilities-Bibliothek können Sie Statusinformationen des TwinCAT-Routers, wie z. B. den verfügbaren Router-Speicher, aus der SPS auslesen.

Ein mit __NEW reservierter dynamischer Speicher muss mittels __DELETE explizit freigegeben werden. Dies muss entweder bereits im gleichen Taskzyklus oder spätestens vor dem Herunterfahren der SPS geschehen. Andernfalls kann es zu einem sogenannten Speicherleck (memory leak) kommen. Infolgedessen können andere Programmteile keinen Speicher mehr reservieren, wodurch ein instabiles System entstehen kann. Es wird die Verwendung der SPS Bibliothek Tc3_DynamicMemory empfohlen, um die Handhabung mit dynamischem Speicher zu vereinfachen.

Allozieren einer STRING-Variablen:

Wenn Sie einen STRING der Länge cLength allozieren möchten, verwenden Sie BYTE anstelle von STRING als Datentyp.

```
VAR CONSTANT
    cLength : UINT := 150;
END_VAR
VAR
    bNew    : BOOL;
    pString : POINTER TO STRING(cLength);
    bDelete : BOOL;
END_VAR
```

```
IF bNew AND (pSTRING = 0) THEN
    pString := __NEW(BYTE, cLength);
    bNew    := FALSE;
END_IF

IF bDelete THEN
    __DELETE(pSTRING);
    bDelete := FALSE;
END_IF
```

Beispiel mit Struktur:

Struktur ST_Sample:

```
{attribute 'enable_dynamic_creation'}
TYPE ST_Sample :
STRUCT
    a,b,c,d,e,f : INT;
END_STRUCT
END_TYPE
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    pDut    : POINTER TO ST_Sample;
    bNew    : BOOL := TRUE;
    bDelete : BOOL;
END_VAR
```

```
IF bNew AND (pDut = 0) THEN
    pDut := __NEW(ST_Sample);
    bNew := FALSE;
```

```
END_IF

IF bDelete THEN
    __DELETE(pDut);
    bDelete := FALSE;
END_IF
```

Beispiel mit Funktionsbaustein:

Funktionsbaustein FB_Dynamic:

```
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FB_Dynamic
VAR
    nCounts : INT;
END_VAR
```

Methode Increase:

```
METHOD Increase : INT
VAR_INPUT
    nCountStep : INT;
END_VAR
```

```
nCounts := nCounts + nCountStep;
Increase := nCounts;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    pFB       : POINTER TO FB_Dynamic;
    nResult   : INT;
    nIncrease : INT := 5;
    bNew      : BOOL;
    bDelete   : BOOL;
END_VAR
```

```
IF bNew AND (pFB = 0) THEN
    pFB := __NEW(FB_Dynamic);
    bNew := FALSE;
END_IF

IF (pFB <> 0) THEN
    nResult := pFB^.Increase(nCountStep := nIncrease);
END_IF

IF bDelete THEN
    __DELETE(pFB);
    bDelete := FALSE;
END_IF
```

Beispiel mit Array:

Programm MAIN:

```
PROGRAM MAIN
VAR
    bNew        : BOOL := TRUE;
    bDelete     : BOOL;
    pArrayBytes : POINTER TO BYTE;
    nTest       : INT;
END_VAR
```

```
IF bNew AND (pArrayBytes = 0) THEN
    pArrayBytes := __NEW(BYTE, 25);
    bNew := FALSE;
END_IF

IF (pArrayBytes <> 0)THEN
    pArrayBytes[24] := 125;  // writing a value to the last array element
    nTest := pArrayBytes[24]
END_IF

IF bDelete THEN
    __DELETE(pArrayBytes);
    bDelete := FALSE;
END_IF
```

16.3.11.2 __DELETE

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator gibt den Speicher von Instanzen wieder frei, die der Operator __NEW dynamisch erzeugt hat. Der Operator __DELETE hat keinen Rückgabewert und der Operand wird nach dieser Operation auf 0 gesetzt.

Syntax: __DELETE (<Pointer>)

Wenn Pointer auf einen Funktionsbaustein zeigt, ruft TwinCAT die zugehörige Methode FB_exit auf, bevor der Pointer auf 0 gesetzt wird.

Umgang mit dynamischem Speicher

Geben Sie immer den genauen Typen der Instanz an, dessen Speicher freigegeben werden soll. Geben Sie bei Vererbung den abgeleiteten Funktionsbaustein an (Variable vom Typ POINTER TO FB_Sub) und nicht den Basis-Funktionsbaustein (Variable vom Typ POINTER TO FB_Base). Wenn der Basis-Funktionsbaustein keine FB_exit-Funktion implementiert, dann wird bei der späteren Verwendung von __DELETE (pfbBase) kein FB_exit aufgerufen!

Beispiel:

Funktionsbaustein FB_Dynamic:

```
FUNCTION_BLOCK FB_Dynamic
VAR_INPUT
    nIn1, nIn2 : INT;
END_VAR
VAR_OUTPUT
    nOut : INT;
END_VAR
VAR
    nTest1 : INT := 1234;
    _inc   : INT := 0;
    _dut   : POINTER TO DUT;
    bNeu   : BOOL;
END_VAR
```

```
nOut := nIn1 + nIn2;
```

Methode FB_exit:

```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL;
END_VAR
```

```
__DELETE(_dut);
```

Methode FB_init:

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode  : BOOL;
END_VAR
```

```
_dut := __NEW(DUT);
```

Methode INC:

```
METHOD INC : INT
VAR_INPUT
END_VAR
```

```
_inc := _inc + 1;
INC  := _inc;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    pFB     : POINTER TO FB_Dynamic;
    bInit   : BOOL := TRUE;
    bDelete : BOOL;
    nLoc    : INT;
END_VAR
```

```
IF (bInit) THEN
    pFB := __NEW(FB_Dynamic);
    bInit := FALSE;
END_IF

IF (pFB <> 0) THEN
    pFB^(nIn1 := 1, nIn2 := nLoc, nOut => nLoc);
    pFB^.INC();
END_IF

IF (bDelete) THEN
    __DELETE(pFB);
END_IF
```

16.3.11.3 __ISVALIDREF

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator dient der Überprüfung, ob eine Referenz auf einen Wert verweist. Die Überprüfung ist somit vergleichbar mit einer Prüfung auf ungleich 0 bei einer Zeigervariablen.

Eine Beschreibung der Anwendung und ein Beispiel zu dem Operator finden Sie in der Beschreibung des Datentyps REFERENCE [▶ 807].

Prüfen von Schnittstellenvariablen

Den Operator __ISVALIDREF können Sie nur für Operanden vom Typ REFERENCE TO verwenden. Die Prüfung von Schnittstellenvariablen ist mit diesem Operator nicht möglich. Um zu überprüfen, ob einer Schnittstellenvariable bereits eine Funktionsbausteininstanz zugewiesen wurde, können Sie die Schnittstellenvariable auf ungleich 0 prüfen (IF iSample <> 0 THEN …).

16.3.11.4 __QUERYINTERFACE

Der Operator ist eine Erweiterung der Norm IEC 61131-3.

Der Operator führt zur Laufzeit eine Typkonvertierung einer Schnittstellen-Referenz zu einer anderen aus. Der Operator liefert ein Ergebnis vom Typ BOOL zurück. TRUE bedeutet, dass TwinCAT die Konvertierung erfolgreich durchgeführt hat.

Syntax: `__QUERYINTERFACE(<ITF_Source>,<ITF_Dest>);`

1. Operand: Schnittstellenvariable oder FB-Instanz

2. Operand: Schnittstellenvariable mit gewünschten Zieltypen

Voraussetzung für die explizite Konvertierung ist, dass sowohl das ITF_Source als auch ITF_Dest eine Ableitung vom Interface __System.IQueryInterface sind. Dieses Interface steht implizit zur Verfügung und benötigt keine Bibliothek.

Beispiel:

Schnittstellen:

```
INTERFACE I_Base EXTENDS __System.IQueryInterface
METHOD BaseMethod : BOOL
```

```
INTERFACE I_Sub1 EXTENDS I_Base
METHOD SubMethod1 : BOOL
```

```
INTERFACE I_Sub2 EXTENDS I_Base
METHOD SubMethod2 : BOOL
```

```
INTERFACE I_Sample EXTENDS __System.IQueryInterface
METHOD SampleMethod : BOOL
```

Funktionsbausteine:

```
FUNCTION_BLOCK FB_1 IMPLEMENTS I_Sub1
METHOD BaseMethod : BOOL
    BaseMethod := TRUE;
METHOD SubMethod1 : BOOL
    SubMethod1 := TRUE;
```

```
FUNCTION_BLOCK FB_2 IMPLEMENTS I_Sub2
METHOD BaseMethod : BOOL
    BaseMethod := FALSE;
METHOD SubMethod2 : BOOL
    SubMethod2 := TRUE;
```

```
FUNCTION_BLOCK FB_3 IMPLEMENTS I_Base, I_Sample
METHOD BaseMethod : BOOL
    BaseMethod   := FALSE;
METHOD SampleMethod : BOOL
    SampleMethod := FALSE;
```

Programm:

```
PROGRAM MAIN
VAR
    fb1           : FB_1;
    fb2           : FB_2;
    fb3           : FB_3;
    iBase1        : I_Base   := fb1;
    iBase2        : I_Base   := fb2;
    iBase3        : I_Base   := fb3;
    iSub1         : I_Sub1   := 0;
    iSub2         : I_Sub2   := 0;
    iSample       : I_Sample := 0;
    bResult1      : BOOL;
    bResult2      : BOOL;
    bResult3      : BOOL;
    bResult4      : BOOL;
    bResult5      : BOOL;
END_VAR
```

```
// Result: bResult1 = TRUE as the conversion is successful => iSub1 references fb1
// Explanation: iBase1 references the object fb1 of type FB_1 that implements the interface I_Sub1
bResult1 := __QUERYINTERFACE(iBase1, iSub1);

// Result: bResult2 = FALSE as the conversion is not successful => iSub2 = 0
// Explanation: iBase1 references the object fb1 of type FB_1 that does not implement the interface
I_Sub2
bResult2 := __QUERYINTERFACE(iBase1, iSub2);

// Result: bResult3 = FALSE as the conversion is not successful => iSub1 = 0
// Explanation: iBase2 references the object fb2 of type FB_2 that does not implement the interface
I_Sub1
bResult3 := __QUERYINTERFACE(iBase2, iSub1);

// Result: bResult4 = TRUE as the conversion is successful => iSub2 references fb2
// Explanation: iBase2 references the object fb2 of type FB_2 that implements the interface I_Sub2
bResult4 := __QUERYINTERFACE(iBase2, iSub2);

// Result: bResult5 = TRUE as the conversion is successful => iSample references fb3
// Explanation: iBase3 references the object fb3 of type FB_3 that implements the interface I_Sample
bResult5 := __QUERYINTERFACE(iBase3, iSample);
```

16.3.11.5 __QUERYPOINTER

Der Operator ist eine Erweiterung der IEC61131-3.

Der Operator ermöglicht zur Laufzeit die Typkonvertierung einer Interface-Reference eines Funktionsbausteins auf einen Pointer. Der Operator liefert ein Ergebnis vom Typ BOOL zurück. TRUE bedeutet, dass TwinCAT die Konvertierung erfolgreich durchgeführt hat.

Aus Kompatibilitätsgründen muss die Definition des zu konvertierenden Pointers eine Erweiterung des Basis-Interface __SYSTEM.IQueryInterface sein.

Syntax: __QUERYPOINTER (<ITF_Source>, <Pointer_Dest>)

Der Operator bekommt als ersten Operanden eine Interface-Referenz oder eine FB-Instanz mit den gewünschten Zieltypen und als zweiten Operanden einen Pointer. Nach Abarbeiten von __QUERYPOINTER enthält Pointer_Dest den Pointer auf diejenige Referenz oder Instanz eines Funktionsbausteins, auf die die Interface-Referenz ITF_Source aktuell verweist. Pointer_Dest ist nicht getypt und kann auf einen beliebigen Typ gecastet werden. Sie müssen den Typ sicherstellen. Beispielsweise könnte das Interface eine Methode anbieten, die einen Typ-Code zurück liefert.

Beispiel:

Schnittstellen:

```
INTERFACE I_Base EXTENDS __System.IQueryInterface
METHOD Base : BOOL
```

```
INTERFACE I_Derived EXTENDS I_Base
METHOD Derived : BOOL
```

Funktionsbaustein:

```
FUNCTION_BLOCK FB_Variante IMPLEMENTS I_Derived
METHOD Base : BOOL
METHOD Derived : BOOL
```

Programm:

```
PROGRAM MAIN
VAR
    iDerived   : I_Derived;
    fbVariante : FB_Variante;
    bResult    : BOOL;
    bTest      : BOOL;
    pFB        : POINTER TO FB_Variante;
END_VAR
```

```
iDerived := fbVariante;
bResult  := __QUERYPOINTER(iDerived, pFB);

IF bResult THEN
    bTest := pFB^.Derived();
END_IF
```

16.3.11.6 __TRY, __CATCH, __FINALLY, __ENDTRY

Die Operatoren sind eine Erweiterung der Norm IEC 61131-3 und dienen einem gezielten Exception-Handling im IEC-Code.

Verfügbar ab TC3.1 Build 4024 für 32 Bit Laufzeitsysteme

Verfügbar ab TC3.1 Build 4026 für 64 Bit Laufzeitsysteme

Syntax:

```
__TRY
    <try_statements>

__CATCH(exc)
    <catch_statements>

__FINALLY
    <finally_statements>

__ENDTRY

<further_statements>
```

Wenn eine Anweisung, die unter dem Operator __TRY steht, eine Exception produziert, hält das SPS-Programm nicht an. Sie führt stattdessen die Anweisungen unter __CATCH aus und startet damit das Exception-Handling. Danach führt sie die Anweisungen unter __FINALLY aus. Das Exception-Handling endet mit __ENDTRY. Dann führt das SPS-Programm die anschließenden Anweisungen aus (Anweisungen nach __ENDTRY).

Die Anweisungen des __TRY-Blocks, die unterhalb jener Anweisung stehen, die die Exception auslöst, werden nicht mehr ausgeführt. Das bedeutet, dass, sobald die Exception geworfen wird, die weitere Ausführung des __TRY-Blocks abgebrochen wird und die Anweisungen unter __CATCH ausgeführt werden.

Die Anweisungen unter __FINALLY werden immer ausgeführt, d. h. auch dann, wenn die Anweisungen unter __TRY keine Exception werfen.

Eine IEC-Variable für eine Exception hat den Datentyp __SYSTEM.ExceptionCode [▶ 776].

Beispiel

Der __TRY-Block des folgenden Beispiels enthält einen Zeigerzugriff und eine Division. Im ersten Zyklus wird innerhalb dieses Blocks eine „Access Violation“-Exception geworfen, da der Zeiger *pSample* zu diesem Zeitpunkt noch ein Null-Pointer ist. Die Verwendung eines Null-Pointers führt zu einer Exception. Im zweiten Zyklus wird innerhalb des __TRY-Blocks eine weitere Exception geworfen – dieses Mal eine Exception aufgrund einer Division durch 0.

Beide Exception-Ursachen werden innerhalb der __CATCH-Anweisungen behoben: die erste Exception durch eine Wertekorrektur des Zeigers *pSample* und die zweite Exception durch eine Wertekorrektur des Divisors *nDivisor*.

Dass bzw. wie das Exception-Handling funktioniert, kann zur Laufzeit wie folgt nachvollzogen werden:

- Da die fehlerhaften Zugriffe, die normalerweise zu einer Laufzeit-Exception und zu einem entsprechenden Stopp der Programmabarbeitung führen, innerhalb des Exception-Handlings abgefangen werden, bleibt die TC-Laufzeit im Run-Modus und die Programmabarbeitung läuft weiter.
- Da zwei Exceptions gefangen werden, wird der Zähler *nCounter_CATCH* zweimal inkrementiert und hat somit den Wert 2.
- Da in jedem Zyklus jeweils die nachfolgend beschriebenen Anweisungen ausgeführt werden, besitzen die dort inkrementierten Zähler alle denselben Wert. Der Wert entspricht der Anzahl der bisherigen Zyklen.
  - die Anweisungen vor dem TRY-CATCH-Block = Inkrementieren des Zählers *nCounter1*
  - die Anweisungen zu Beginn des __TRY-Blocks = Inkrementieren des Zählers *nCounter_TRY1*
  - die Anweisungen unter __FINALLY = Inkrementieren des Zählers *nCounter_FINALLY*
  - die Anweisungen nach dem TRY-CATCH-Block = Inkrementieren des Zählers *nCounter2*
- Da die Ausführung des __TRY-Blocks aufgrund der zwei geworfenen Exceptions zweimal abgebrochen wird, wird in zwei Zyklen das Ende der __TRY-Anweisungen nicht erreicht. Somit ist der Variablenwert von *nCounter_TRY2* um 2 kleiner als der von *nCounter_TRY1*.
- Da die Ursachen für die beiden Exceptions innerhalb der __CATCH-Anweisungen behoben werden (*pSample* eine gültige Adresse sowie *nDivisor* einen Wert ungleich 0 zuweisen), treten die beiden Exceptions jeweils nur einmal auf.
- Die beiden Exceptions werden in dem Beispiel in ein globales Array zur Erstellung einer Exception-Historie gespeichert. Im Anschluss an die beiden Exceptions ist die entsprechende Indexvariable *nExcIndex* somit 2 und das Array *aExceptionHistory* besitzt die folgenden Werte:
  - *aExceptionHistory*[0] = RTSEXCPT_ACCESS_VIOLATION
  - *aExceptionHistory*[1] = RTSEXCPT_DIVIDEBYZERO
  - *aExceptionHistory*[2] bis *aExceptionHistory*[10] = RTSEXCPT_NOEXCEPTION
- Im Anschluss an die beiden Exceptions ist die Variable *exc* zurück auf dem Default-Wert RTSEXCPT_NOEXCEPTION und die Variable *lastExc* zeigt mit RTSEXCPT_DIVIDEBYZERO den zuletzt aufgetretenen Exception-Code an.

Globale Variablenliste „GVL_Exc“:

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
    cMaxExc             : UINT := 10;
END_VAR
VAR_GLOBAL
    nExcIndex           : UINT;
    aExceptionHistory   : ARRAY[0..cMaxExc] OF __SYSTEM.ExceptionCode;
END_VAR
```

Funktion „F_SaveExceptionCode“:

```
FUNCTION F_SaveExceptionCode
VAR_INPUT
    excInput            : __SYSTEM.ExceptionCode;
END_VAR
```

```
// Log the thrown exception into the global exception history array
IF GVL_Exc.nExcIndex <= GVL_Exc.cMaxExc THEN
    GVL_Exc.aExceptionHistory[GVL_Exc.nExcIndex] := excInput;
    GVL_Exc.nExcIndex := GVL_Exc.nExcIndex + 1;
END_IF
```

Programm „MAIN“:

```
PROGRAM MAIN
VAR
    nCounter1          : INT;
    nCounter2          : INT;
    nCounter_TRY1      : INT;
    nCounter_TRY2      : INT;
    nCounter_CATCH     : INT;
    nCounter_FINALLY   : INT;

    exc                : __SYSTEM.ExceptionCode;
    lastExc            : __SYSTEM.ExceptionCode;

    pSample            : POINTER TO BOOL;
    bVar               : BOOL;
    nSample            : INT := 100;
    nDivisor           : INT;
END_VAR
```

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

// TRY-CATCH block
__TRY
    nCounter_TRY1 := nCounter_TRY1 + 1;
    pSample^      := TRUE;                // 1. cycle: null pointer access leads to "access
violation" exception
    nSample       := nSample/nDivisor;    // 2. cycle: division by zero leads to "divide by zero"
exception
    nCounter_TRY2 := nCounter_TRY2 + 1;

__CATCH(exc)
    nCounter_CATCH := nCounter_CATCH + 1;

    // Exception logging
    lastExc := exc;
    F_SaveExceptionCode(excInput := exc);

    // Correct the faulty variable values
    IF (exc = __SYSTEM.ExceptionCode.RTSEXCPT_ACCESS_VIOLATION) AND (pSample = 0) THEN
        pSample  := ADR(bVar);
    ELSIF ((exc = __SYSTEM.ExceptionCode.RTSEXCPT_DIVIDEBYZERO) OR (exc =
__SYSTEM.ExceptionCode.RTSEXCPT_FPU_DIVIDEBYZERO)) AND (nDivisor = 0) THEN
        nDivisor := 1;
    END_IF

__FINALLY
    nCounter_FINALLY := nCounter_FINALLY + 1;

__ENDTRY

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

16.3.11.6.1 Datentyp __SYSTEM.ExceptionCode

Siehe auch: __TRY, __CATCH, __FINALLY, __ENDTRY [▶ 774]

```
TYPE ExceptionCode :
(
    RTSEXCPT_UNKNOWN                                := 16#FFFFFFFF,
    RTSEXCPT_NOEXCEPTION                            := 16#00000000,
    RTSEXCPT_WATCHDOG                               := 16#00000010,
    RTSEXCPT_HARDWAREWATCHDOG                       := 16#00000011,
    RTSEXCPT_IO_CONFIG_ERROR                        := 16#00000012,
    RTSEXCPT_PROGRAMCHECKSUM                        := 16#00000013,
    RTSEXCPT_FIELDBUS_ERROR                         := 16#00000014,
    RTSEXCPT_IOUPDATE_ERROR                         := 16#00000015,
    RTSEXCPT_CYCLE_TIME_EXCEED                      := 16#00000016,
    RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED             := 16#00000017,
    RTSEXCPT_UNRESOLVED_EXTREFS                     := 16#00000018,
    RTSEXCPT_DOWNLOAD_REJECTED                      := 16#00000019,
    RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR  := 16#0000001A,
    RTSEXCPT_LOADBOOTPROJECT_FAILED                 := 16#0000001B,
    RTSEXCPT_OUT_OF_MEMORY                          := 16#0000001C,
    RTSEXCPT_RETAIN_MEMORY_ERROR                    := 16#0000001D,
```

```
    RTSEXCPT_BOOTPROJECT_CRASH                      := 16#0000001E,
    RTSEXCPT_BOOTPROJECTTARGETMISMATCH              := 16#00000021,
    RTSEXCPT_SCHEDULEERROR                          := 16#00000022,
    RTSEXCPT_FILE_CHECKSUM_ERR                      := 16#00000023,
    RTSEXCPT_RETAIN_IDENTITY_MISMATCH               := 16#00000024,
    RTSEXCPT_IEC_TASK_CONFIG_ERROR                  := 16#00000025,
    RTSEXCPT_APP_TARGET_MISMATCH                    := 16#00000026,
    RTSEXCPT_ILLEGAL_INSTRUCTION                    := 16#00000050,
    RTSEXCPT_ACCESS_VIOLATION                       := 16#00000051,
    RTSEXCPT_PRIV_INSTRUCTION                       := 16#00000052,
    RTSEXCPT_IN_PAGE_ERROR                          := 16#00000053,
    RTSEXCPT_STACK_OVERFLOW                         := 16#00000054,
    RTSEXCPT_INVALID_DISPOSITION                    := 16#00000055,
    RTSEXCPT_INVALID_HANDLE                         := 16#00000056,
    RTSEXCPT_GUARD_PAGE                             := 16#00000057,
    RTSEXCPT_DOUBLE_FAULT                           := 16#00000058,
    RTSEXCPT_INVALID_OPCODE                         := 16#00000059,
    RTSEXCPT_MISALIGNMENT                           := 16#00000100,
    RTSEXCPT_ARRAYBOUNDS                            := 16#00000101,
    RTSEXCPT_DIVIDEBYZERO                           := 16#00000102,
    RTSEXCPT_OVERFLOW                               := 16#00000103,
    RTSEXCPT_NONCONTINUABLE                         := 16#00000104,
    RTSEXCPT_PROCESSORLOAD_WATCHDOG                 := 16#00000105,
    RTSEXCPT_FPU_ERROR                              := 16#00000150,
    RTSEXCPT_FPU_DENORMAL_OPERAND                   := 16#00000151,
    RTSEXCPT_FPU_DIVIDEBYZERO                       := 16#00000152,
    RTSEXCPT_FPU_INEXACT_RESULT                     := 16#00000153,
    RTSEXCPT_FPU_INVALID_OPERATION                  := 16#00000154,
    RTSEXCPT_FPU_OVERFLOW                           := 16#00000155,
    RTSEXCPT_FPU_STACK_CHECK                        := 16#00000156,
    RTSEXCPT_FPU_UNDERFLOW                          := 16#00000157,
    RTSEXCPT_VENDOR_EXCEPTION_BASE                  := 16#00002000,
    RTSEXCPT_USER_EXCEPTION_BASE                    := 16#00010000
) UDINT ;
END_TYPE
```

16.3.11.7 __VARINFO

Der Operator ist eine Erweiterung der Norm IEC 61131-3. Der Operator liefert Informationen zu einer Variablen zurück. Sie können die Informationen als Datenstruktur in einer Variablen des Datentyps __SYSTEM.VAR_INFO speichern.

Syntax in der Deklaration:

```
<name of the info variable> : __SYSTEM.VAR_INFO; // Data structure for info variable
```

Syntax beim Aufruf:

```
<name of the info variable> := __VARINFO( <variable name> ); // Call of the operator
```

Beispiel:

Zur Laufzeit enthält die Variable MyVarInfo die Informationen über die Variable nVar.

```
VAR
    MyVarInfo : __SYSTEM.VAR_INFO;
    nVar      : INT;
END_VAR
```

```
MyVarInfo := __VARINFO(nVar);
```

Datentyp SYSTEM.VAR_INFO

Eine Variable mit Datentyp __SYSTEM.VAR_INFO enthält:

Name	Datentyp	Initialisierung	Beschreibung
ByteAddress	DWORD	0	Adresse der Variablen Beispiel: 16#072E35EC Hinweis: Beim Bitzugriff einer Variablen `<variable name>.<bit index>` wird die Adresse der Variablen angegeben, die das Bit enthält.
ByteOffset	DWORD	0	Offset der Variablenadresse in Bytes. Beispiel: 13936 Bytes. Hinweis: Wenn die Variable global ist, dann ist der Offset relativ zum Beginn der Area. Wenn die Variable eine lokale Variable in einer Funktion oder Methode ist, dann ist der Offset relativ zum aktuellen Stack-Frame. Wenn die Variable eine lokale Variable in einem Funktionsbaustein ist, dann ist der Offset relativ zur Funktionsbaustein-Instanz.
Area	DINT	0	Speicherbereichsnummer Area im Laufzeitsystem. Beispiel: -1. Bedeutet, dass die Variable nicht global im Speicher liegt, sondern relativ zu einer Instanz oder auf dem Stack. Hinweis: Die Speicherbereiche sind geräteabhängig.
BitNr	INT	0	Anzahl der Bits in Bytes Beispiel: 16#00FF Bytes Hinweis: Wenn die Variable kein ganzzahliger Datentyp ist, gilt: `BitNr = -1 = 16#FFFF`
BitSize	INT	0	Speicherplatzgröße der Variablen in Bits Beispiel: 16 Bits
BitAddress	UDINT	0	Bitadresse der Variablen Voraussetzung: Die Variable liegt im Eingangsspeicherbereich I, Ausgangsspeicherbereich Q oder Merkerspeicherbereich M. Ansonsten ist der Wert undefiniert.
TypeClass	TYPE_CLASS	TYPE_BOOL	Datentypklasse der Variablen Beispiel: TYPE_INT, TYPE_ARRAY Hinweis: Bei benutzerdefinierten Datentypen oder Funktionsbaustein-Instanzen wird als Datantypklasse TYPE_USERDEF ausgegeben.
TypeName	STRING(79)	‘’	Datentypname der Variablen als STRING(79) Hinweis: Bei benutzerdefinierten Datentypen ist der Funktionsbausteinname oder der DUT-Name angegeben. Beispiel: 'INT', 'ARRAY'
NumElements	UDINT	0	Anzahl der Arrayelemente Voraussetzung: Die Variable hat den Datentyp ARRAY. Beispiel: 8
BaseTypeClass	TYPE_CLASS	TYPE_BOOL	Elementarer Basisdatentyp der Arrayelemente. Voraussetzung: Die Variable hat den Datentyp ARRAY. Beispiel: TYPE_INT bei arrA : ARRAY [1..2,1..2,1..2] OF INT;
ElemBitSize	UDINT	0	Speicherplatzgröße des Arrayelements in Bits Voraussetzung: Die Variable hat den Datentyp ARRAY. Beispiel: 16 Bits bei arrA : ARRAY [1..2,1..2,1..2] OF INT;

MemoryArea	MEMORY_AREA	MEM_MEMORY	Information zum Speicherbereich • MEM_GLOBAL: Globaler Speicherbereich Beispielsweise in Area 0 • MEM_LOCAL: Lokaler Speicherbereich in Area -1 • MEM_MEMORY: Merkerspeicherbereich %M Beispielsweise in 16#10 in Area 1 • MEM_INPUT: Eingangsspeicherbereich %I Beispielsweise in 16#04 in Area 2 • MEM_OUTPUT: Ausgangsspeicherbereich %Q Beispielsweise in 16#08 in Area 3 • MEM_RETAIN: Retain-Speicherbereich Beispielsweise in 16#20 in Area 0 Beispiel: MEM_GLOBAL
Symbol	STRING(39)	‘’	Variablenname als STRING(39) Beispiel : 'iCounter', 'arrA'
Comment	STRING(79)	‘’	Kommentar der Variablendeklaration Beispiel: 'Counts the calls' oder 'Stores the A data'

16.3.11.8 __POUNAME

Verfügbar ab TC3.1 Build 4026

Der Operator ist eine Erweiterung der Norm IEC 61131-1.

Der Operator liefert zur Laufzeit den Namen des Programmierbausteins (POU), die den Operator __POUNAME enthält. Dazu muss der Operator im Deklarationsteil oder im Implementierungsteil einer Variablen vom Typ `STRING` zugewiesen werden.

Das Ergebnis von __POUNAME ist abhängig vom Ort der Verwendung:

- Innerhalb eines Programms: Programmname
- Innerhalb einer Funktion: Funktionsname
- Innerhalb eines Funktionsbausteins: Name des Funktionsbausteins
- Innerhalb einer Methode: Name der Methode, qualifiziert mit dem Namen des Funktionsbausteins
- Innerhalb einer Get-/Set-Accessors in einer Eigenschaft: Name der Eigenschaft, qualifiziert mit dem Namen des Funktionsbausteins, + Get/Set
- Innerhalb eine GVL: Name der GVL
- Innerhalb einer Struktur: Name der Struktur
- Innerhalb einer Datenstruktur UNION: Name der UNION

Beispiel:

```
PROGRAM MAIN
VAR
    sPouNameDecl : STRING := __POUNAME();  //Liefert 'MAIN'
    sPouNameImpl : STRING;
END_VAR
```

```
sPouNameImpl:= __POUNAME();               //Liefert 'MAIN'
```

16.3.11.9 __POSITION

Verfügbar ab TC3.1 Build 4026

Der Operator ist eine Erweiterung der Norm IEC 61131-1.

Der Operator liefert zur Laufzeit die Position einer Variablen im Deklarationsteil oder im Implementierungsteil eines Programmierbausteins. Dazu muss der Operator __POSITION im Deklarationsteil beziehungsweise im Implementierungsteil einer Variablen vom Typ `STRING` zugewiesen werden.

Ergebnis von __POSITION:

- Deklarationsteil: `Line <line number> (Decl)`
- Implementierungsteil: `Line <line number>, Column <Column number> (Impl)`

Beispiel:

```
PROGRAM MAIN
VAR
    sPositionDecl  : STRING := __POSITION();   //Liefert die Zeilennummer dieser Deklaration
    sPositionImpl  : STRING;
END_VAR
```

```
sPositionImpl := __POSITION();                 //Liefert Zeilen- und Spaltennummer dieser Zuweisung
```

16.4 Operanden

Konstanten und Literale

Konstanten sind Bezeichner für unveränderliche Werte. Sie können Konstanten lokal innerhalb eines Programmierbausteins oder global innerhalb einer globalen Variablenliste deklarieren. Der Deklarationsabschnitt wird dazu mit dem Schlüsselwort CONSTANT erweitert.

Konstanten sind auch Zeichenfolgen, die den Wert eines Basistyps wie beispielsweise Ganzzahlen oder Gleitkommazahlen darstellen, beispielsweise 16#FFFF_FFFF, T#5s oder -1.234 E-5. Zur Unterscheidung werden solche Konstanten auch als Literale, literale Konstanten oder unbenannte Konstanten bezeichnet. Es gibt logische (TRUE, FALSE) oder numerische Literale (3.1415, T#5s), aber auch Zeichenliterale ('Hello world!', "black").

Syntax Deklaration:

```
<scope> CONSTANT
    <identifier>:<data type> := <initial value>;
END_VAR
```

```
<scope>         : VAR | VAR_INPUT | VAR_STAT | VAR_GLOBAL
<data type>     : <elementary data type | user defined data type | function block >
<initial value> : literal value | identifier | expression
```

Erlaubte Initialwerte:

- Literal, beispielsweise TRUE, FALSE, 16#FFFF_FFFF
- Benannte Konstante, die an anderer Stelle deklariert wurde.
- Einfacher Ausdruck aus Literalen, auch kombiniert mit einfachen Operatoren wie beispielsweise + - *.

Eingänge oder Funktionsaufrufe können nicht als Initialwert angegeben werden.

Beispiel:

```
VAR_GLOBAL CONSTANT
    cMax     : INT := 100;
    cSpecial : INT := cMax - 10;
END_VAR
```

Konstanten werden nur bei der Deklaration beschrieben. Die Zuweisung eines Initialwerts ist obligat. Innerhalb einer Implementierung werden Konstanten ausschließlich gelesen und stehen deshalb in einer Anweisung immer rechts vom Zuweisungsoperator.

Die Konstanten werden beim Compilieren des Codes mit dem Initialwert ersetzt. Der Initialwert muss außerdem zur Compilezeit berechnet werden können.

Konstanten von strukturierten oder benutzerdefinierten Typen werden erst zur Laufzeit berechnet. Strukturierte Konstanten in Programmen oder GVLs werden einmal zum Programmstart berechnet. Strukturierte Konstanten in Funktionen oder Methoden werden jedes Mal berechnet, wenn die Funktion oder Methode aufgerufen wird. Die Initialisierung von strukturierten Konstanten kann somit von Eingaben abhängen oder Funktionsaufrufe ausführen.

Siehe auch:

- BOOL-Konstanten [▶ 781]
- Zahlenkonstanten [▶ 782]
- REAL/LREAL-Konstanten [▶ 782]
- STRING-Konstanten [▶ 783]
- TIME/LTIME-Konstanten [▶ 784]
- Datums- und Uhrzeitkonstanten [▶ 785]
- Getypte Konstanten / Typed Literals [▶ 787]

Variablen

Sie können Variablen entweder lokal im Deklarationsteil eines Bausteins oder in einer globalen Variablenliste deklarieren.

An welcher Stelle Sie eine Variable verwenden können, hängt von ihrem Datentyp ab.

Siehe auch:

- Zugriff auf Variablen von Arrays, Strukturen und Bausteinen [▶ 788]
- Bitzugriff auf Variablen [▶ 788]

Weitere

- Adressen [▶ 790]
- Funktionen [▶ 792]

Siehe auch:

- Variablen deklarieren [▶ 68]
- Eingabeunterstützung nutzen [▶ 141]
- Konstante Variablen - CONSTANT [▶ 724]
- ST-Ausdrücke [▶ 657]

Sehen Sie dazu auch

- Eingabeunterstützung nutzen [▶ 141]
- Eingabeunterstützung nutzen [▶ 141]
- Variablen deklarieren [▶ 68]

16.4.1 BOOL-Konstanten

BOOL-Konstanten sind die Wahrheitswerte TRUE (1) und FALSE (0).

Siehe auch:

- BOOL [▶ 794]

16.4.2 Zahlenkonstanten

Zahlenwerte können als Dualzahlen, Oktalzahlen, Dezimalzahlen und Hexadezimalzahlen auftreten. Wenn ein Integerwert keine Dezimalzahl ist, dann müssen Sie seine Basis gefolgt von einem Doppelkreuz (#) vor die Integerkonstante schreiben. Die Ziffernwerte für die Zahlen 10 bis 15 bei Hexadezimalzahlen geben Sie wie üblich durch die Buchstaben A-F an.

Sie können Unterstriche innerhalb eines Zahlenwertes verwenden.

Beispiele:

14	Dezimalzahl
2#1001_0011	Dualzahl
8#67	Oktalzahl
16#A	Hexadezimalzahl
DINT#16#A1	Getypter Datentyp DINT# und Basis 16# kombiniert.

Der Typ dieser Zahlenwerte kann dabei BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL oder LREAL sein.

Implizite Konvertierungen von "größeren" zu "kleineren" Typen sind nicht erlaubt. Sie können eine DINT-Variable nicht ohne weiteres als INT-Variable benutzen. Hierfür müssen Sie eine Typkonvertierungs-Funktion verwenden.

Da Zahlenkonstanten grundsätzlich als ganzzahlige Werte behandelt werden, müssen Sie in Divisionen die Konstante im Format einer Gleitkommazahl angeben, um den Rest nicht zu verlieren. Beispiel: Division 1/10 ergibt 0, Division 1.0/10 ergibt 0.1.

Siehe auch:

- Operatoren [▶ 732]
- Getypte Konstanten / Typed Literals [▶ 787]

16.4.3 REAL/LREAL-Konstanten

Sie können Gleitkommazahlen als REAL- und LREAL-Konstanten in Punktschreibweise oder in Exponentialschreibweise mit Mantisse und Exponent angeben. Der Punkt dient als Dezimaltrennzeichen gemäß der International System of Units (English).

Syntax Exponentialdarstellung:

```
<significand> e | E <exponent>
```

```
exponent :  -44..38    // REAL
exponent : -324..308   // LREAL
```

Beispiele:

7.4	Dezimalzahl; 7,4 mit Komma liefert Compilerfehler.
1/3.0	Dezimalbruch für 0.333333343 Hinweis: Bei Division von Integertypen bleibt das Ergebnis ein Integertyp. Dabei wird abgeschnitten. Beispielsweise liefert 1/3 als Ergebnis 0.
1.64e+009	Exponentialdarstellung

REAL-Literal

-3.402823e+38	Kleinste Zahl
-1E-44	Größte negative Zahl
1.0E-44	Kleinste positive Zahl
3.402823e+38	Größte Zahl

LREAL-Literal

-1.7976931348623157E+308	Kleinste Zahl
-4.94065645841247E-324	Größte negative Zahl
4.94065645841247E-324	Kleinste positive Zahl
1.7976931348623157E+308	Größte Zahl

Zuweisung eines besonders großen Literals

Um ein ganzzahliges Literal, welches größer als die Obergrenze von ULINT ist, zuzuweisen, muss entweder ein Komma gesetzt werden oder ein expliziter Typecast angegeben werden. Ohne eine solche Spezifizierung als Gleitkommazahl können Informationen verloren gehen.
Beispiel:
fMyReal : REAL := 3400000000000000000000.0;
fMyReal : REAL := REAL#340000000000000000000;

Siehe auch:

- REAL/LREAL [▶ 795]

16.4.4 STRING-Konstanten

Eine STRING-Konstante ist eine von einfachen Anführungszeichen umschlossene Zeichenfolge. Die Zeichen werden gemäß des Zeichensatzes Windows-1252 codiert. Als Untermenge von Windows-1252 wird somit der Zeichensatz der ISO/IEC 8859-1 unterstützt. Eine STRING-Konstante kann Leerzeichen und Umlaute enthalten, da die Zeichen ein Teil des Zeichensatzes sind. Sie wird auch als Zeichenliteral oder einfach nur als String bezeichnet.

Beispiel:

```
'Hello World!'
```

Wenn in einer STRING-Konstanten ein Dollarzeichen ($) ist, werden die nachfolgenden zwei Zeichen als Hexadezimalcode gemäß der Codierung Windows-1252 interpretiert. Der Code entspricht auch dem ASCII-Code. Beachten Sie außerdem die Sonderfälle.

Hexadezimalcode

String mit $-Code	Interpretation
$<8-Bit-Code>	8-Bit-Code: Zweistellige hexadezimale Zahl, die gemäß ISO/IEC 8859-1 interpretiert wird.
'$41'	A
'$A9'	©
'$40'	@
'$0D'	Steuerzeichen Zeilenumbruch, entspricht '$R'.
'$0A'	Steuerzeichen neue Zeile, entspricht '$L' und '$SN'.

Sonderfälle

String mit $-Code	Interpretation
'$L', '$l'	Steuerzeichen Zeilenvorschub, entspricht '$0A'.
'$N', '$n'	Steuerzeichen Neue Zeile, entspricht '$0A'.
'$P', '$p'	Steuerzeichen Seitenvorschub
'$R', '$r'	Steuerzeichen Zeilenumbruch, entspricht '$0D'.
'$T', '$t'	Steuerzeichen Tabulator
'$$'	Dollarzeichen: $
'$''	Einfaches Anführungszeichen: '

Beispiel: Konstantendeklaration

```
VAR CONSTANT
    sConstA : STRING := 'Hello Allgäu';
    sConstB : STRING := 'Hello Allgäu $21'; // Hello Allgäu!
END_VAR
```

16.4.5 TIME/LTIME-Konstanten

Sie können TIME-Konstanten benutzen, um die Standard-Timer-Module zu bedienen. Die Konstante hat eine Größe von 32 Bit und somit eine Auflösung in Millisekunden.

Außerdem steht Ihnen die Zeitkonstante LTIME als Zeitbasis für hochauflösende Timer zur Verfügung. Die LTIME-Konstante hat eine Größe von 64 Bit und somit eine Auflösung in Nanosekunden.

TIME-Konstante

Syntax:

```
<time keyword> # <length of time>
```

```
<time keyword>   : TIME | time | T | t
<length of time> : ( <number of days>d )? ( <number of hours>h )? ( <number of minutes>m )?
( <number of seconds>s )? (<number of milliseconds>ms)?    // ( ...)? Optional
```

Die Reihenfolge der Zeiteinheiten darf nicht verändert werden. Es ist jedoch nicht erforderlich, alle Einheiten anzugeben.

Zeiteinheiten:

- D | d : Tage
- H | h : Stunden
- M | m : Minuten
- S | s : Sekunden
- MS | ms : Millisekunden

Beispiele: Korrekte Zeitkonstanten in einer ST-Zuweisung

```
VAR
    tLength0 : TIME := T#14ms;
    tLength1 : TIME := T#100s12ms;      // Overflow in the highest unit is allowed.
    tLength2 : TIME := T#12h34m15s;
    tCompare : TIME;
    bOK      : BOOL;
    tLongest := T#49D17H2M47S295MS;    // 4294967295
END_VAR
```

```
IF tLength < T#15MS THEN
    IF tCompare < tLength1 THEN
        bOK := TRUE;
    END_IF;
END_IF
```

Beispiele: Unkorrekte Verwendung von Zeitkonstanten

`tIncorrect1 := t#5m68s;`	Überlauf bei einer niedrigeren Stelle
`tIncorrect2 := 15ms;`	Zeitkennung T# fehlt
`tIncorrect3 := t#4ms13d;`	Unkorrekte Reihenfolge der Zeiteinheiten

LTIME-Konstante

Syntax:

```
<long time keyword> # <length of high resolution time>
```

```
<long time keyword>              : LTIME | ltime
<length of high resolution time> : <length of time> ( <number of microseconds>us )? ( <number of
nanoseconds>ns )?    // ( ...)? Optional
```

Sie können für LTIME-Konstanten die gleichen Zeiteinheiten wie für TIME-Konstanten verwenden. Zusätzlich können Sie Mikrosekunden und Nanosekunden angeben, da die Zeitangabe in höherer Zeitauflösung gerechnet wird. Intern werden LTIME-Literale wie der Datentyp LWORD behandelt und der Wert deswegen in Nanosekunden aufgelöst.

Zusätzliche Zeiteinheiten:

- US | us : Mikrosekunden
- NS | ns : Nanosekunden

Beispiele: Korrekte Zeitkonstanten in einer ST-Zuweisung

```
VAR
    tLength0 : TIME := LTIME#1000d15h23m12s34ms2us44ns;
    tLength1 : TIME := LTIME#3445343m3424732874823ns;
END_VAR
```

16.4.6 Datums- und Uhrzeitkonstanten

32-Bit-Datumsangaben ‚DATE'

Verwenden Sie das Schlüsselwort DATE (D), um Datumsangaben zu machen.

Syntax:

```
<date keyword>#<year>-<month>-<day>
```

```
<date keyword> : DATE | date | D | d
<year>  : 1970-2106
<month> : 1-12
<day>   : 1-31
```

DATE-Literale werden intern wie der Datentyp DWORD behandelt, was einer Obergrenze von DATE#2106-2-7 entspricht.

Beispiel:

```
PROGRAM MAIN
VAR
    dStart    : DATE := DATE#2018-8-8;
    dEnd      : DATE := D#2018-8-31;
    dCompare  : DATE := date#1996-05-06;
    bInTime   : BOOL;

    dEarliest : DATE := d#1970-1-1;      // = 0
    dLatest   : DATE := DATE#2106-2-7;   // = 4294967295
END_VAR
```

```
IF dStart < dCompare THEN
    IF dCompare < dEnd THEN
        bInTime := TRUE;
    END_IF;
END_IF
```

64-Bit-Datumsangaben ‚LDATE'

Verwenden Sie das Schlüsselwort LDATE (LD), um Datumsangaben zu machen.

Syntax:

```
<date keyword>#<year>-<month>-<day>
```

```
<date keyword> : LDATE | ldate | LD | ld
<year>  : 1970-2262
<month> : 1-12
<day>   : 1-31
```

LDATE-Literale werden intern wie der Datentyp LWORD behandelt, was einer Obergrenze von DATE#2554-7-21 entspricht.

Beispiel:

```
PROGRAM MAIN
VAR
    dStart    : LDATE := LDATE#2018-8-8;
    dEnd      : LDATE := ldate#2018-8-31;
    dCompare  : LDATE := LD#1996-05-06;
    bInTime   : BOOL;

    dEarliest : LDATE := d#1970-1-1;      // = 0
    dLatest   : LDATE := DATE#2106-2-7;   // = 4294967295
END_VAR
```

```
IF dStart < dCompare THEN
    IF dCompare < dEnd THEN
        bInTime := TRUE;
    END_IF;
END_IF
```

32-Bit-Datums- und Uhrzeitangaben ‚DATE_AND_TIME'

Verwenden Sie das Schlüsselwort DATE_AND_TIME (DT), um Datums- und Uhrzeitangaben zu machen.

Syntax:

<date and time keyword>#<date and time value>

<date and time keyword> : DATE_AND_TIME | date_and_time | DT | dt
<date and time value> : <year>-<month>-<day>-<hour>:<minute>:<second>
<year> : 1970-2106
<month> : 1-12
<day> : 1-31
<hour> : 0-24
<minute> : 0-59
<second> : 0-59

DATE_AND_TIME-Literale werden intern als Datentyp DWORD behandelt. Die Zeit wird in Sekunden verarbeitet und kann folglich Werte von 1. Januar 1970 00:00 Uhr bis 07.Februar 2106 6:28:15 Uhr annehmen.

Beispiel:

```
PROGRAM MAIN
VAR
    dtDate0    : DATE_AND_TIME := DATE_AND_TIME#1996-05-06-15:36:30;
    dtDate1    : DATE_AND_TIME := DT#1972-03-29-00:00:00;
    dtDate2    : DT            := DT#2018-08-08-13:33:20.5;

    dtEarliest : DATE_AND_TIME := DATE_AND_TIME#1979-1-1-00:00:00;    // 0
    dtLatest   : DATE_AND_TIME := DATE_AND_TIME#2106-2-7-6:28:15;     // 4294967295
END_VAR
```

64-Bit-Datums- und Uhrzeitangaben ‚LDATE_AND_TIME'

Verwenden Sie das Schlüsselwort LDATE_AND_TIME (LDT), um Datums- und Uhrzeitangaben zu machen.

Syntax:

<date and time keyword>#<long date and time value>

<date and time keyword> : LDATE_AND_TIME | ldate_and_time | LDT | ldt
<date and time value> : <year>-<month>-<day>-<hour>:<minute>:<second>
<year> : 1970-2554
<month> : 1-12
<day> : 1-31
<hour> : 0-24
<minute> : 0-59
<second> : 0-59

LDATE_AND_TIME-Literale werden intern als Datentyp LWORD behandelt. Die Zeit wird in Sekunden verarbeitet und kann folglich Werte von 1. Januar 1970 00:00 Uhr bis 21. Juli 2554 23:59:59.999999999 Uhr annehmen.

Beispiel:

```
PROGRAM MAIN
VAR
    dtDate0    : LDATE_AND_TIME := LDATE_AND_TIME#1996-05-06-15:36:30;
    dtDate1    : LDATE_AND_TIME := LDT#1972-03-29-00:00:00;
    dtDate2    : LDT            := LDT#2018-08-08-13:33:20.5;

    dtEarliest : LDATE_AND_TIME := LDT#1979-1-1-00:00:00;    // 0
    dtLatest   : LDATE_AND_TIME := LDT#2266-4-10-23:59:59;   // 16#7FFF63888C620000
END_VAR
```

32-Bit-Datumsangaben ‚TIME_OF_DAY'

Verwenden Sie das Schlüsselwort TIME_OF_DAY (TOD), um Uhrzeitangaben zu machen.

Syntax:

```
<time keyword>#<time value>
```

```
<time keyword> : TIME_OF_DAY | time_of_day | TOD | tod
<time value>   : <hour>:<minute>:<second>
<hour>   : 0-23
<minute> : 0-59
<second> : 0.000-59.999
```

Sie können bei Sekunden auch Sekundenbruchteile angeben. TIME_OF_DAY-Literale werden intern als DWORD behandelt und somit der Wert in Millisekunden aufgelöst.

Beispiel:

```
PROGRAM MAIN
VAR
    tdClockTime0 : TIME_OF_DAY := TIME_OF_DAY#15:36:30.123;
    tdClockTime1 : TOD         := TOD#12:34:56.789;

    tdEarliest   : TIME_OF_DAY := TIME_OF_DAY#0:0:0.000;
    tdLatest     : TIME_OF_DAY := TIME_OF_DAY #23:59:59.999;
END_VAR
```

64-Bit-Datumsangaben ‚LTIME_OF_DAY'

Verwenden Sie das Schlüsselwort LTIME_OF_DAY (LTOD), um Uhrzeitangaben zu machen.

Syntax:

```
<time keyword>#<time value>
```

```
<time keyword> : LTIME_OF_DAY | ltime_of_day | LTOD | ltod
<time value>   : <hour>:<minute>:<second>
<hour>   : 0-23
<minute> : 0-59
<second> : 0.000-59.999999999
```

Sie können bei Sekunden auch Sekundenbruchteile angeben. LTIME_OF_DAY-Literale werden intern als LWORD behandelt, somit wird der Wert in Nanosekunden aufgelöst.

Beispiel:

```
PROGRAM MAIN
VAR
    tdClockTime0 : LTIME_OF_DAY := LTIME_OF_DAY#15:36:30.123;
    tdClockTime1 : LTOD         := LTOD#12:34:56.7890123456;

    tdEarliest   : LTIME_OF_DAY := LTIME_OF_DAY#0:0:0.000;
    tdLatest     : LTIME_OF_DAY := LTIME_OF_DAY#23:59:59. 999999999;
END_VAR
```

Siehe auch:

- Datums- und Uhrzeitdatentypen [▶ 797]

16.4.7 Getypte Konstanten / Typed Literals

Mit Ausnahme von REAL/LREAL-Konstanten (hier wird immer LREAL verwendet) verwendet TwinCAT beim Rechnen mit IEC-Konstanten den kleinstmöglichen Datentyp. Wenn Sie einen anderen Datentyp verwenden wollen, können Sie dies mithilfe von Typed Literals (Getypte Konstanten) erreichen, ohne dass Sie die Konstante explizit deklarieren müssen. Versehen Sie hierbei die Konstante mit einem Präfix, das den Typ festlegt.

Syntax: <Type>#<Literal>

<Type> gibt den gewünschten Datentyp an, mögliche Eingaben: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL. Sie müssen den Typ in Großbuchstaben schreiben.

<Literal> gibt die Konstante an. Die Eingabe muss zum unter <Type> angegebenen Datentypen passen.

Beispiel:

```
var1:=DINT#34;
```

Wenn TwinCAT die Konstante nicht ohne Datenverlust in den Zieltyp überführen kann, wird eine Fehlermeldung ausgegeben.

Sie können getypte Konstanten überall dort verwendet, wo Sie normale Konstanten verwenden können.

16.4.8 Zugriff auf Variablen von Arrays, Strukturen und Bausteinen

Schreibweise für den Zugriff auf:

- Zweidimensionale Array-Komponente: `<Array-Name> [ <1st dimension>, <2nd dimension> ]`
- Strukturvariable: `<structure name> . <component name>`
- Funktionsbaustein- oder Programmvariable: `<function block name> | <project name > . <variable name>`

Siehe auch:

- ARRAY [▶ 809]
- Struktur [▶ 814]
- Objekt Funktionsbaustein [▶ 86]
- Objekt Programm [▶ 88]

16.4.9 Bitzugriff auf Variablen

Mit einem Indexzugriff können Sie einzelne Bits in ganzzahligen Variablen adressieren. Mit Hilfe einer Strukturvariablen oder einer Funktionsbaustein-Instanz können Sie einzelne Bits symbolisch adressieren.

Indexzugriff auf Bits in ganzzahligen Variablen

Sie können in ganzzahligen Variablen einzelne Bits adressieren. Dazu hängen Sie an die Variable mit einem Punkt abgetrennt den Index des zu adressierenden Bits an. Der Bitindex kann durch eine beliebige Konstante angegeben werden. Die Indizierung ist 0-basiert.

Syntax:

```
<integer variable name> . <index>
```

```
<integer data typ> = BYTE | WORD | DWORD | LWORD | SINT | USINT | INT | UINT | DINT | UDINT | LINT | ULINT
```

Wenn der Typ der Variablen nicht zulässig ist, gibt TwinCAT folgende Fehlermeldung aus: Unzulässiger Datentyp <Typ> für direkte Indizierung. Wenn der Index größer als die Bitbreite der Variablen ist, gibt TwinCAT folgenden Fehler aus: Index <n> außerhalb des gültigen Bereichs für Variable <Name>.

Beispiel Indexzugriff:

Im Programm wird das dritte Bit der Variablen `nVarA` auf den Wert der Variablen `nVarB` gesetzt.

```
PROGRAM MAIN
VAR
    nVarA : WORD := 16#FFFF;
bVarB : BOOL := 0;
END_VAR
```

```
// Index access in an integer variable
nVarA.2 := bVarB;
```

Ergebnis: nVarA = 2#1111_1111_1111_1011 = 16#FFFB

Beispiel Konstante als Index:

Die Konstante `cEnable` fungiert als Index, um auf das dritte Bit der Variablen `nVar` zuzugreifen.

```
// GVL declaration
VAR_GLOBAL CONSTANT
    cEnable : USINT := 2;
END_VAR
```

```
PROGRAM MAIN
VAR
    nVar    : INT   := 0;
END_VAR
```

```
// Constant as index
nVar.cEnable := TRUE; // Third bit in nVar is set TRUE
```

Ergebnis: nVar = 4

Erreichbarkeit der Variablen, die die Bitnummer definiert

Beachten Sie, dass die Variable, die die Bitnummer definiert (im Beispiel oben `nEnable`), direkt über den Variablennamen und ohne vorhergehenden Namensraum erreichbar sein muss.
Der Bitzugriff ist also beispielsweise zulässig, wenn die Variable im lokalen Scope (gleiche Ebene wie `nVar`) oder im globalen Scope auf einer GVL ohne das Attribut 'qualified_only' [▶ 859] deklariert wurde.
Wird die Variable auf einer GVL deklariert, die mit dem Attribut 'qualified_only' versehen ist, ist ein Zugriff auf `nEnable` nur durch Angabe des globalen Variablenlistennamens möglich (GVL.nEnable). Ein solcher Zugriff auf eine globale Variable kann nicht für den Bitzugriff verwendet werden (nicht möglich: `nVar.GVL.nEnable := TRUE;`).

Symbolischer Bitzugriff in Strukturvariablen

Mit dem Datentyp `BIT` können Sie einzelne Bits mit einem Namen bezeichnen und zu einer Struktur zusammenfassen. Das Bit wird dann mit dem Komponentennamen adressiert.

Beispiel

Typdeklaration der Struktur:

```
TYPE ST_ControllerData :
STRUCT
    nStatus_OperationEnabled : BIT;
    nStatus_SwitchOnActive   : BIT;
    nStatus_EnableOperation  : BIT;
    nStatus_Error            : BIT;
    nStatus_VoltageEnabled   : BIT;
    nStatus_QuickStop        : BIT;
    nStatus_SwitchOnLocked   : BIT;
    nStatus_Warning          : BIT;
END_STRUCT
END_TYPE
```

Deklaration und Schreibzugriff auf ein Bit:

```
PROGRAM MAIN
VAR
    stControllerDrive1 : ST_ControllerData;
END_VAR
```

```
// Symbolic bit access to nStatus_ EnableOperation
stControllerDrive1.nStatus_EnableOperation := TRUE;
```

Symbolischer Bitzugriff in Funktionsbaustein-Instanzen

In Funktionsbausteinen können Sie Variablen für einzelne Bits deklarieren.

Beispiel

Typdeklaration der Struktur:

```
FUNCTION_BLOCK FB_Controller
VAR_INPUT
    nSwitchOnActive   : BIT;
    nEnableOperation  : BIT;
    nVoltageEnabled   : BIT;
    nQuickStop        : BIT;
    nSwitchOnLocked   : BIT;
END_VAR
VAR_OUTPUT
```

```
    nOperationEnabled : BIT;
    nError            : BIT;
    nWarning          : BIT;
END_VAR
VAR
END_VAR
```

```
PROGRAM MAIN
VAR
    fbController : FB_Controller;
END_VAR
```

```
// Symbolic bit access to nSwitchOnActive
fbController(nSwitchOnActive:= TRUE);
```

Siehe auch:

- BIT [▶ 795]
- Ganzzahlige Datentypen [▶ 794]
- Bitzugriff in Strukturen [▶ 815]

Sehen Sie dazu auch

Struktur [▶ 815]

16.4.10 Adressen

⚠ VORSICHT

Verschiebung der Inhalte von Adressen durch Online-Change

Wenn Sie Pointer auf Adressen verwenden, können sich bei einem Online-Change Inhalte von Adressen verschieben.

Automatische Adressierung

Es wird empfohlen, keine direkte Adressierung für allokierte Variablen zu verwenden, sondern stattdessen den Platzhalter * zu nutzen.
Mit dem Platzhalter * (%I*, %Q* bzw. %M*) wird eine flexible und optimierte Adressierung von TwinCAT automatisch durchgeführt.

Syntax:

```
<identifier> AT <address> : <data type>;
```

Bei der Angabe einer Adresse werden die Position im Speicher und die Größe mittels spezieller Zeichenfolgen ausgedrückt. Eine Adresse ist gekennzeichnet mit dem Prozentzeichen %, dann folgt der Speicherbereichspräfix, der optionale Größenpräfix und die Speicherposition.

```
%<memory area prefix> ( <size prefix> )? <memory position>
```

```
<memory area prefix> : I | Q | M
<size prefix>        : X | B | W | D
<memory position>    : * | <number> ( .<number> )*
```

Speicherbereichspräfix

I	Eingangsspeicherbereich für Eingänge ("Inputs") Für physikalische Eingänge über Eingangstreiber ("Sensoren")
Q	Ausgangsspeicherbereich für Ausgänge ("Outputs") Physikalische Ausgänge über Ausgangstreiber ("Aktoren")
M	Merkerspeicherbereich

Größenpräfix

X	Single Bit
B	Byte (8 Bits)
W	Word (16 Bits)
D	Double word (32 Bits)

Beispiele:

```
IbSensor1 AT%I*    : BOOL;
IbSensor2 AT%IX7.5 : BOOL;
```

Wenn Sie nicht explizit eine Einzelbitadresse angeben, werden boolesche Variablen byteweise alloziert. Beispiel: Eine Wertänderung von bVar AT %QB0 betrifft den Bereich von QX0.0 bis QX0.7.

Beispiele

Variablendeklarationen:

IbSensor AT%I* : BOOL;	Bei der Adressangabe ist statt der Speicherposition der Platzhalter * angegeben. Dadurch wird eine flexible und optimierte Adressierung von TwinCAT automatisch durchgeführt.
InInput AT%IW0 : WORD;	Variablendeklaration mit Adressangabe eines Eingangsworts
ObActuator AT%QB0 : BOOL;	Boolesche Variablendeklaration Hinweis: Für boolesche Variable wird intern ein Byte alloziert, wenn keine Einzelbitadresse angegeben ist. Eine Wertänderung von ObActuator betrifft folglich den Bereich von QX0.0 bis QX0.7.
IbSensor AT%IX7.5 : BOOL;	Boolesche Variablendeklaration mit expliziter Angabe einer Einzelbitadresse. Beim Zugriff wird nur das Eingangsbit 7.5 gelesen.

Weitere Adressen:

%QX7.5 %Q7.5	Einzelbitadresse des Ausgangsbits 7.5
%IW215	Wortadresse des Eingangsworts 215
%QB7	Byteadresse des Ausgangsbytes 7
%MD48	Adresse eines Doppelworts an der Speicherstelle 48 im Merkerbereich
%IW2.5.7.1	Interpretation abhängig von der aktuellen Steuerungskonfiguration (siehe unten)

Mögliche Speicherbereichsüberlappungen bei direkten Adressen

Um in einem SPS-Projekt eine gültige Adresse zuzuweisen, muss Ihnen die gewünschte Position im Prozessabbild bekannt sein. Dafür müssen Sie zunächst den Speicherbereich sowie die erforderliche Größe festlegen. Bei der Wahl der Speicherposition muss die Zuordnung der verschiedenen Größen im Speicher, wie in der folgenden Tabelle dargestellt, beachtet werden, damit Sie Speicherbereichsüberlappungen ausschließen können.

DWord	Word	Byte	Bit
D0	W0	B0	X0.0
		B1	X1.0
	W1	B2	X2.0
		B3	X3.0
D1	W2	B4	X4.0
		B5	X5.0
	W3	B6	X6.0
		B7	X7.0
D2	W4	B8	X8.0
…			

Beispiele: Speicherbereichsüberlappung

1. W0 enthält B0 und B1. Wenn Sie eine Word Variable auf W0 und eine boolesche Variable auf B1 legen, würden sich Ihre Speicherbereiche überlappen.

2. W3 enthält B6 und B7. Wenn Sie eine Word Variable auf W3 und eine boolesche Variable auf B6 legen, würden sich Ihre Speicherbereiche überlappen.

Siehe auch:

- SPS-Projekt programmieren > Variablen deklarieren > AT-Deklaration [▶ 71]

16.4.11 Funktionen

In ST können Sie einen Funktionsaufruf als Operand verwenden.

Beispiel:

```
nResult := F_Add(7,5) + 3;
```

Siehe auch:

- Objekt Funktion [▶ 83]

TIME()-Funktion

Diese Funktion liefert einen Wert vom Datentyp TIME.
Die TIME()-Funktion stellt keinen absoluten Bezugspunkt dar, sondern kann durch die Differenzbildung von mindestens zwei TIME()-Rückgabewerten für relative Zeitmessungen verwendet werden.

Beispiel in ST:

Das folgende Beispiel beinhaltet einen TON-Baustein (fbTimer), der mit einer Zeit von 5 Sekunden beschaltet (tTimerValue) und zu Programmbeginn gestartet wird. Die steigende Flanke der Timer-Aktivierung wird von einem R_TRIG-Baustein erkannt (fbTrigger), woraufhin der Rückgabewert der TIME()-Funktion zum ersten Mal zwischengespeichert wird (tTimeReturn1). Sobald die Zeitspanne des Timers abgelaufen ist, wird ein zweiter Rückgabewert der TIME()-Funktion zwischengespeichert (tTimeReturn2). Durch die Differenzbildung der beiden gespeicherten TIME()-Rückgabewerte (tDifference) wird die relative Zeit zwischen den jeweiligen TIME()-Aufrufen berechnet. In diesem Fall wird demnach die Zeit zwischen Timer-Anfang und Timer-Ende kalkuliert, welche 5 Sekunden beträgt.

```
PROGRAM MAIN
VAR
    bStart          : BOOL := TRUE;
    fbTimer         : TON;
    tTimerValue     : TIME := T#5S;
    fbTrigger       : R_TRIG;
    tTimeReturn1    : TIME;
    tTimeReturn2    : TIME;
    tDifference     : TIME;
END_VAR

//=================================================

fbTimer(IN := bStart, PT := tTimerValue);
fbTrigger(CLK := fbTimer.IN);

IF fbTrigger.Q THEN
    tTimeReturn1  := TIME();
END_IF

IF fbTimer.Q THEN
    bStart        := FALSE;
    tTimeReturn2  := TIME();
    tDifference   := tTimeReturn2 - tTimeReturn1;    // The difference will be T#5s
END_IF
```

16.5 Datentypen

In der Programmierung wird eine Variable durch ihren Namen identifiziert und hat eine Adresse im Speicher des Zielsystems. Variablennamen sind demnach Bezeichner, unter denen der zugewiesene Speicherplatz adressiert wird. Die Größe der Variable wird durch ihren Datentyp bestimmt. Dieser gibt an, wieviel Speicherplatz für die Variable reserviert ist und wie die Werte im Speicher zu interpretieren sind. Der Datentyp bestimmt auch, welche Operationen erlaubt sind.

In TwinCAT besteht außerdem die Möglichkeit, Funktionsbausteine zu instanziieren. Funktionsbausteininstanzen belegen dann ähnlich wie Variablen den Speicher. Der Speicherbedarf wird vom Funktionsbaustein bestimmt.

Folgende Gruppen von Datentypen stehen Ihnen zur Verfügung:

Standarddatentypen

Ein Standarddatentyp ist ein elementarer Datentyp oder ein Stringdatentyp.

```
<standard data type> : __UXINT | __XINT | __XWORD | BIT | BOOL | BYTE | DATE | DATE_AND_TIME | DINT
| DT | DWORD | INT | LDATE | LDATE_AND_TIME | LDT | LINT | LREAL | LTIME | LTOD | LWORD | REAL |
SINT |STRING | TIME | TOD | TIME_OF_DAY | UDINT | UINT | ULINT | USINT | WORD | WSTRING
```

Siehe auch:

- BOOL [▶ 794]
- Ganzzahlige Datentypen [▶ 794]
- REAL/LREAL [▶ 795]
- STRING [▶ 796]
- WSTRING [▶ 797]
- TIME/LTIME [▶ 797]
- Datums- und Uhrzeitdatentypen [▶ 797]
- Spezialdatentypen UXINT, XINT, XWORD und PVOID [▶ 804]

Erweiterungen zur Norm IEC 61131-3

Siehe auch:

- BIT [▶ 795]
- Zeiger / POINTER [▶ 804]
- REFERENCE [▶ 807]
- UNION [▶ 820]
- ANY und ANY_<type> [▶ 799]
- Datentyp __SYSTEM.ExceptionCode [▶ 805]

Benutzerdefinierte Datentypen

Sie können eigene Datentypen deklarieren, die auf den standardmäßig vordefinierten oder auf bereits bestehende Datentypen basieren.

Solche Datentypen werden als benutzerdefiniert oder anwenderspezifisch bezeichnet. Die Datentypen sind entweder als eigenes DUT-Objekt organisiert oder werden innerhalb des Deklarationsteils eines Programmierobjekts deklariert. Außerdem werden sie aufgrund ihres Zwecks und ihrer Syntax unterschieden.

Benutzerdefinierter Datentyp	Deklaration	Siehe auch
Alias	DUT-Objekt	Alias [▶ 819]
Arrays	Programmierobjekt	ARRAY [▶ 809]
Enumeration	DUT-Objekt, Programmierobjekt	Aufzählungen / Enumerationen [▶ 816]
Pointer	Programmierobjekt	Zeiger / POINTER [▶ 804]
Referenz	Programmierobjekt	REFERENCE [▶ 807]
Struktur	DUT-Objekt	Struktur [▶ 814]
Unterbereichstyp	Programmierobjekt	Unterbereichstypen [▶ 795]
Union	DUT-Objekt	UNION [▶ 820]

Beachten Sie die Empfehlungen zur Namensvergabe für Bezeichner.

Siehe auch:

- Bezeichner [▶ 882]

16.5.1 BOOL

Datentyp	Werte	Speicherplatz
BOOL	TRUE (1), FALSE (0)	8 Bit

Siehe auch:

- BOOL-Konstanten [▶ 781]

16.5.2 Ganzzahlige Datentypen

In TwinCAT stehen Ihnen folgende ganzzahlige Datentypen zur Verfügung.

Datentyp	Untergrenze	Obergrenze	Speicherplatz
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
LWORD	0	2^{64}-1	64 Bit
SINT	-128	127	8 Bit
USINT	0	255	8 Bit
INT	-32768	32767	16 Bit
UINT	0	65535	16 Bit
DINT	-2147483648	2147483647	32 Bit
UDINT	0	4294967295	32 Bit
LINT	-2^{63}	2^{63}-1	64 Bit
ULINT	0	2^{64}-1	64 Bit

Bei Typkonvertierung von größeren zu kleineren Typen können Information verloren gehen.

Siehe auch:

- Zahlenkonstanten [▶ 782]

16.5.3 Unterbereichstypen

Ein Unterbereichstyp ist ein Datentyp, dessen Wertebereich eine Untermenge eines Basistypen umfasst. Als Basistyp sind nur Integer-Typen möglich.

Syntax: <Name> : <Inttype> (<ug>..<og>)

<Name>	Gültiger IEC-Bezeichner
<Inttype>	Datentyp des Unterbereichs (SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, LINT, ULINT, LWORD).
<ug>	Untergrenze des Bereichs: Konstante, die mit dem Basisdatentyp kompatibel sein muss. Die Untergrenze selbst gehört zu diesem Bereich.
<og>	Obergrenze des Bereichs: Konstante, die mit dem Basisdatentyp kompatibel sein muss. Die Obergrenze selbst gehört zu diesem Bereich.

Beispiel:

```
VAR
    nVarA: INT (-4095..4095);
    nVarB : UINT (0..10000);
END_VAR
```

Wenn Sie einem Unterbereichstypen in der Deklaration oder in der Implementierung ein Wert zuweisen, der nicht in diesen Bereich fällt (zum Beispiel nVarA := 5000), gibt TwinCAT eine Fehlermeldung aus.

Beachten Sie die Möglichkeit, die Bereichsgrenzen eines Unterbereichstypen während der Laufzeit mit den impliziten Überwachungsfunktionen CheckRangeSigned und CheckRangeUnsigned zu überwachen.

Siehe auch:

- Range/LRange Checks (POUs CheckRangeSigned, CheckRangeUnsigned, CheckLRangeSigned, CheckLRangeUnsigned) [▶ 176]
- POU CheckRangeUnsigned

16.5.4 BIT

Sie können den Datentyp BIT nur für einzelne Variablen innerhalb von Strukturen oder Funktionsbausteinen verwenden. Die möglichen Werte sind TRUE (1) und FALSE (0).

Ein BIT-Element benötigt 1 Bit Speicherplatz und Sie können damit einzelne Bits einer Struktur oder eines Funktionsbausteins über ihren Namen ansprechen. BIT-Elemente, die nacheinander deklariert sind, werden in Bytes zusammengefasst. Dadurch können Sie im Vergleich zu BOOL-Typen, die jeweils mindestens 8 Bits belegen, den Speicherverbrauch optimieren. Allerdings benötigt der Bitzugriff wesentlich mehr Zeit. Deshalb sollten Sie den Datentyp BIT nur verwenden, wenn Sie die Daten in genau vorgegebenem Format definieren wollen.

Siehe auch:

- Struktur [▶ 814]
- Objekt Funktionsbaustein [▶ 86]

16.5.5 REAL/LREAL

Die Datentypen REAL und LREAL sind Gleitpunkttypen nach IEEE 754. Sie sind nötig bei der Verwendung von Dezimalzahlen und Gleitpunktzahlen in Punktdarstellung oder Exponentialdarstellung.

Datentyp	Untergrenze	Obergrenze	Betragsmäßig kleinste Zahl	Speicherplatz
REAL	-3.402823e+38	3.402823e+38	1.0e-44	32 Bit
LREAL	-1.79769313486231 58e+308	1.797693134862315 8e+308	4.94065645841247e -324	64 Bit

Beispiel

```
PROGRAM MAIN
VAR
    fMaxReal     : REAL  := 3.402823E+38;              // Largest REAL number
    fPosMinReal  : REAL  := 1.0E-44;                   // Smallest positive REAL number
    fNegMaxReal  : REAL  := -1.0E-44;                  // Largest negative REAL number
    fMinReal     : REAL  := -3.402823E+38;             // Smallest REAL number

    fMaxLreal    : LREAL := 1.7976931348623157E+308;   // Largest LREAL number
    fPosMinLreal : LREAL := 4.94065645841247E-324;     // Smallest positive LREAL number
    fNegMaxLreal : LREAL := -4.94065645841247E-324;    // Largest negative LREAL number
    fMinLreal    : LREAL := -1.7976931348623157E+308;  // Smallest LREAL number
END_VAR
```

Wenn der Wert der REAL/LREAL-Zahl außerhalb des Wertebereichs des Integers liegt, wird bei einer Datentypkonvertierung von REAL oder LREAL nach SINT, USINT, INT, UINT, DINT, UDINT, LINT oder ULINT ein undefiniertes Ergebnis geliefert.

Zuweisung eines besonders großen Literals

Um ein ganzzahliges Literal zuzuweisen, das größer ist als die Obergrenze von ULINT, muss entweder ein Komma gesetzt oder ein expliziter Typecast angegeben werden. Ohne eine solche Spezifizierung als Gleitkommazahl können Informationen verloren gehen.
Beispiel:
fMyReal : REAL := 3400000000000000000000.0;
fMyReal : REAL := REAL#340000000000000000000;

Siehe auch:

- REAL/LREAL-Konstanten [▶ 782]

16.5.6 STRING

Eine Variable vom Datentyp STRING kann eine beliebige Zeichenkette aufnehmen. Die Größenangabe zur Speicherplatz-Reservierung bei der Deklaration bezieht sich auf Zeichen und steht in runden oder eckigen Klammern. Ist keine Größe angegeben, nimmt TwinCAT standardmäßig 80 Zeichen an.

TwinCAT begrenzt die String-Länge grundsätzlich nicht, allerdings verarbeitet die String-Funktion nur Längen von 1-255! Wenn eine Variable mit einem String initialisiert wird, der zu lang ist für den Datentyp der Variablen, schneidet TwinCAT den String von hinten her entsprechend ab.

Der für eine STRING-Variable benötigte Speicherplatz ist immer 1 Byte pro Zeichen + 1 zusätzliches Byte, beispielsweise 81 Bytes im Falle einer STRING(80)-Deklaration.

Beispiel: Stringdeklaration mit 35 Zeichen

```
sVar : STRING(35) := 'This is a String';
```

Siehe auch:

- STRING-Konstanten [▶ 783]
- WSTRING [▶ 797]

16.5.7 WSTRING

Der Datentyp WSTRING ist im Gegensatz zum Datentyp STRING (ASCII) in Unicode-Format interpretiert. Durch diese Codierung hängt bei WSTRING die Anzahl der darstellbaren Zeichen davon ab, welche Zeichen dargestellt werden sollen. Eine Länge von 10 heißt bei WSTRING, dass die Länge des WSTRINGs maximal 10 WORDs belegen kann. In Unicode werden aber für einige Zeichen mehrere WORDs für die Codierung eines Zeichens benötigt, sodass die Anzahl der Zeichen nicht der Länge des WSTRINGs (hier 10) entsprechen muss.

Der Datentyp benötigt 1 WORD pro Character und 1 WORD extra an Speicherplatz. Ein STRING benötigt jeweils nur 1 Byte. Der Datentyp WSTRING ist mit einer 0 terminiert.

Beispiel:

```
wsVar : WSTRING := "This is a WString";
```

Siehe auch:

- STRING [▶ 796]
- STRING-Konstanten [▶ 783]

16.5.8 TIME/LTIME

Der Zeitdatentyp TIME wird intern wie ein UDINT (32 Bit) behandelt. Das führt zu einer Auflösung in Millisekunden.

Der Zeitdatenyp LTIME wird intern wie ein ULINT (64 Bit) behandelt. Sie können diesen Datentyp als Zeibasis für hochauflösender Timer mit einer Auflösung in Nanosekunden verwenden.

Datentyp	Untergrenze	Obergrenze	Speicherplatz	Auflösung
TIME	0	4294967295 (49d17h2m47s295ms)	32 Bit	Millisekunden
LTIME	0	213503d23h34m33s709ms551us615ns	64 Bit	Nanosekunden

Die Zeitdeklaration kann die Zeiteinheiten enthalten, die für TIME- bzw. LTIME-Konstanten gelten.

Beispiel:

```
VAR
    tTime  : TIME  := T#1d2h30m40s500ms
    tLTime : LTIME := LTIME# 100d2h30m40s500ms600us700ns
END_VAR
```

Siehe auch:

- TIME/LTIME-Konstanten [▶ 784]
- Datums- und Uhrzeitdatentypen [▶ 797]
- Datums- und Uhrzeitkonstanten [▶ 785]
- TIME/TOD_TO_<type> [▶ 763]

16.5.9 Datums- und Uhrzeitdatentypen

Die Datentypen DATE, DATE_AND_TIME (DT) und TIME_OF_DAY (TOD) werden intern wie ein UDINT (32-Bit-Wert) behandelt.

Datentyp	Untergrenze	Obergrenze	Speicherplatz	Auflösung
DATE	0 = D#1970-01-01 (01.01.1970)	4294967295 = D#2106-02-07 (07.02.2106)	32 Bit	Sekunden, obwohl nur der Tag angezeigt wird.
DATE_AND_TIME DT	0 = DT#1970-1-1-0:0:0 (01.01.1970, 00:00 Uhr)	4294967295 = DT#2106-02-07-06:28:15 (07.02.2106, 6:28:15)	32 Bit	Sekunden
TIME_OF_DAY TOD	0 = TOD#0:0:0 (00:00:00:000 Uhr)	86399999 = TOD#23:59:59.999 (23:59:59.999 Uhr)	32 Bit	Millisekunden

Die Datentypen LDATE, LDATE_AND_TIME (LDT) und LTIME_OF_DAY (LTOD) werden intern wie ein ULINT (64 Bit) behandelt.

Datentyp	Untergrenze	Obergrenze	Speicherplatz	Auflösung
LDATE	0 = LD#1970-1-1 (01.01.1970)	2^{64}-1 = LD#2554-7-21 (21.07. 2554)	64 Bit	Nanosekunden, obwohl nur der Tag angezeigt wird.
LDATE_AND_TIME LDT	0 = LDT#1970-1-1-0:0:0 (01.01.1970, 00:00 Uhr)	2^{64}-1 = LDT#2554-7-21-23:34:33.709551615 (21.07.2554, 23:34:33.709551615 Uhr)	64 Bit	Nanosekunden
LTIME_OF_DAY LTOD	0 = LTOD#0:0:0 (00:00:00:000 Uhr)	86399999999999 = LTOD#23:59:59.999999999 (23:59:59.999999999 Uhr)	64 Bit	Nanosekunden

Voraussetzungen

Für die Datentypen LDATE, LDATE_AND_TIME (LDT) und LTIME_OF_DAY (LTOD) wird die TwinCAT Version 3.1.4026.0 oder höher vorausgesetzt.

Beispiele:

```
VAR

    // Date
    dLowerLimit   : DATE           := DATE#1970-1-1;
    dUpperLimit   : DATE           := DATE#2106-2-7;
    dAppointment  : DATE           := D#2020-2-7;

    // Date and time
    dtLowerLimit  : DATE_AND_TIME  := DATE_AND_TIME#1970-1-1-0:0:0;
    dtUpperLimit  : DATE_AND_TIME  := DATE_AND_TIME#2106-02-07-06:28:15;
    dtAppointment : DT             := DT#2020-2-7-12:55:1.234;

    // Time of day
    tdLowerLimit  : TIME_OF_DAY    := TIME_OF_DAY#0:0:0;
    tdUpperLimit  : TIME_OF_DAY    := TIME_OF_DAY#23:59:59.999;
    tdAppointment : TOD            := TOD#12:3:4.567;

    // Long date
    dLowerLimit   : LDATE          := LDATE#1970-1-1;
    dUpperLimit   : LDATE          := LDATE#2106-2-7;
    dAppointment  : LDATE          := LD#2020-2-7;

    // Long date and time
    dtLowerLimit  : LDATE_AND_TIME := LDATE_AND_TIME#1970-1-1-0:0:0;
    dtUpperLimit  : LDATE_AND_TIME := LDATE_AND_TIME#2262-4-10-23:59:59.99999999;
    dtAppointment : LDT            := LDT#2020-2-7-12:55:1.234567891;

    // Long time of day
    tdLowerLimit  : LTIME_OF_DAY   := LTIME_OF_DAY#0:0:0;
    tdUpperLimit  : LTIME_OF_DAY   := LTIME_OF_DAY#23:59:59.999999999;
```

```
    tdAppointment : LTOD           := LTOD#12:3:4.567890123;

END_VAR
```

Siehe auch:

- Datums- und Uhrzeitkonstanten [▶ 785]
- TIME/LTIME [▶ 797]
- TIME/LTIME-Konstanten [▶ 784]
- DATE/DT_TO_<type> [▶ 764]
- FileTime Datentyp T_FILETIME64 aus der Tc2_Utilities SPS Bibliothek
- DC Time Datentyp T_DCTIME64 aus der Tc2_EtherCAT SPS Bibliothek

16.5.10 ANY und ANY_<type>

Bei der Implementierung einer Funktion, einer Methode oder (ab Build 4026) eines Funktionsbausteins können Sie Eingänge (VAR_INPUT) als Variablen mit generischem IEC-Datentyp, `ANY` oder `ANY_<type>`, deklarieren. Folglich können Sie Aufrufe implementieren, deren Aufrufparameter sich im Datentyp unterscheiden.

Zur Laufzeit können Sie innerhalb des Programmierbausteins für die Eingangsvariable den übergebenen Wert und dessen Typ über eine vordefinierte Struktur abfragen.

Der Compiler ersetzt den Typ der Eingangsvariable intern mit der unten beschriebenen Datenstruktur, wobei der Wert nicht direkt übergeben wird. Stattdessen wird ein Zeiger auf den eigentlichen Wert übergeben, weswegen nur eine Variable übergeben werden kann. Erst beim Aufruf wird also der Datentyp konkretisiert. Aufrufe solcher Programmierbausteine können daher mit Argumenten, die jeweils unterschiedliche Datentypen haben, erfolgen.

Einem Eingang vom Datentyp `ANY` oder `ANY_<type>` kann bei dem Aufruf der Funktion, des Funktionsbausteins oder der Methode keine Konstante und keine Eigenschaft zugewiesen werden. Andersherum kann einer Eigenschaft keine Variable vom Datentyp `ANY` oder `ANY_<type>` zugewiesen werden.

Die unten dargestellten generischen IEC-Datentypen werden unterstützt. Die Tabelle stellt dar, welche generischen Datentypen welche elementaren Datentypen erlauben.

Generische Datentypen			Elementare Datentypen
ANY	ANY_BIT		• BYTE • WORD • DWORD • LWORD
	ANY_DATE		• DATE_AND_TIME, DT • DATE • TIME_OF_DAY, TOD • LDATE • LDATE_AND_TIME, LDT • LTIME_OF_DAY, LTOD
	ANY_NUM	ANY_REAL	• REAL • LREAL
		ANY_INT	• USINT • UINT • UDINT • ULINT • SINT • INT • DINT • LINT
	ANY_STRING		• STRING • WSTRING

Interne Datenstruktur bei 'ANY' und 'ANY_<type>'

Beim Übersetzen des Codes werden die Eingangsvariablen mit ANY-Datentyp intern mit der folgenden Struktur ersetzt. Der tatsächliche Aufrufparameter wird den Strukturelementen zur Laufzeit zugewiesen.

```
TYPE AnyType :
STRUCT
      // the type of the actual parameter
  typeclass  : __SYSTEM.TYPE_CLASS ;
      // the pointer to the actual parameter
  pvalue     : POINTER TO BYTE;
      // the size of the data, to which the pointer points
  diSize     : DINT;
END_STRUCT
END_TYPE
```

Über diese Struktur können Sie innerhalb des Programmierbausteins auf die Eingangsvariable zugreifen und beispielsweise den übergebenen Wert abfragen.

Deklaration

Die Syntaxbeschreibungen beziehen sich auf einen Programmierbaustein mit genau einem Parameter (eine Eingangsvariable).

Syntax

```
FUNCTION | FUNCTION_BLOCK | METHOD <POU name>
( : <return data type> )?
VAR_INPUT
    <input variable name> : <generic data type>;
END_VAR
<generic data type> = ANY | ANY_BIT | ANY_DATE |
ANY_NUM | ANY_REAL | ANY_INT | ANY_STRING
```

Aufruf

Die Syntaxbeschreibungen beziehen sich auf einen Programmierbaustein mit genau einem Parameter, dem ein Argument übergeben wird. Der Datentyp des Arguments konkretisiert dabei den generischen Datentyp der Eingangsvariable. Beispielsweise können Argumente des Typs `BYTE`, `WORD`, `DWORD`, `LWORD` an eine `ANY_BIT`-Eingangsvariable übergeben werden.

Syntax Funktionsaufruf

```
<variable name> := <function name> ( <argument name> );
<argument name> : variable with valid data type
```

Syntax Funktionsbausteinaufruf

```
<function block name> ( <input variable name> := <argument name> );
```

Syntax Methodenaufruf

```
<function block name> . <method name> ( <input variable name> := <argument name> );
```

Beispiel 1: Übergabe von elementaren Datentypen an Eingänge mit generischem Datentyp

```
FUNCTION F_ComputeAny : BOOL
VAR_INPUT
    anyInput1         : ANY; // valid data type see table
END_VAR
```

```
FUNCTION_BLOCK FB_ComputeAny
VAR_INPUT
    anyInput1     : ANY;
END_VAR
```

```
FUNCTION_BLOCK FB_ComputeMethod
METHOD ComputeAny : BOOL
VAR_INPUT
    anyInput1     : ANY_INT; // valid data types are SINT, INT, DINT,INT, USINT, UINT, UDINT, ULINT
END_VAR
```

```
PROGRAM PLC_PRG
VAR
    fbComputeAnyByte  : FB_ComputeAny;
    fbComputeAnyInt   : FB_ComputeAny;

    fbComputeM1       : FB_ComputeMethod;
    fbComputeM2       : FB_ComputeMethod;

    nByte             : BYTE := 16#AB;
    nInt              : INT  := -1234;
    bResultByte       : BOOL;
    bResultInt        : BOOL;
END_VAR
```

```
bResultByte := F_ComputeAny(nByte);
bResultInt  := F_ComputeAny(nInt);

fbComputeAnyByte(anyInput1 := nByte);
fbComputeAnyInt(anyInput1 := nInt);

fbComputeM1.methComputeAny(anyInput1 := nByte);
fbComputeM2.methComputeAny(anyInput1 := nInt);
```

Beispiel 2: Übergabe von elementaren Datentypen an Eingänge mit generischem Datentyp

Die Übergabeparameter der Funktionsaufrufe haben unterschiedliche Datentypen.

```
FUNCTION F_AnyBitFunc    : BOOL
VAR_INPUT
    value : ANY_BIT;
END_VAR

FUNCTION F_AnyDateFunc   : BOOL
VAR_INPUT
    value : ANY_DATE;
END_VAR

FUNCTION F_AnyFunc       : BOOL
VAR_INPUT
    value : ANY;
END_VAR
```

```
FUNCTION F_AnyIntFunc    : BOOL
VAR_INPUT
    value : ANY_INT;
END_VAR

FUNCTION F_AnyNumFunc    : BOOL
VAR_INPUT
    value : ANY_NUM;
END_VAR

FUNCTION F_AnyRealFunc   : BOOL
VAR_INPUT
    value : ANY_REAL;
END_VAR

FUNCTION F_AnyStringFunc : BOOL
VAR_INPUT
    value : ANY_STRING;
END_VAR
```

```
PROGRAM MAIN
VAR
    bBOOL            : BOOL           := TRUE;
    nBYTE            : BYTE           := 16#AB;
    nWORD            : WORD           := 16#1234;
    nDWORD           : DWORD          := 16#6789ABCD;
    nLWORD           : LWORD          := 16#0123456789ABCDEF;
    sSTRING          : STRING         := 'xyz';
    wsWSTRING        : WSTRING        := "abc";
    dtDATEANDTIME    : DATE_AND_TIME  := DT#2017-02-20-11:07:00;
    dDATE            : DATE           := D#2017-02-20;
    tdTIMEOFDAY      : TIME_OF_DAY    := TOD#11:07:00;
    fREAL            : REAL           := 42.24;
    fLREAL           : LREAL          := 24.42;
    nUSINT           : USINT          := 12;
    nUINT            : UINT           := 1234;
    nUDINT           : UDINT          := 12345;
    nULINT           : ULINT          := 123456;
    nSINT            : SINT           := -12;
    nINT             : INT            := -1234;
    nDINT            : DINT           := -12345;
    nLINT            : LINT           := -123456;
END_VAR
```

```
F_AnyFunc(bBOOL);
F_AnyFunc(nBYTE);
F_AnyFunc(nWORD);
F_AnyFunc(nDWORD);
F_AnyFunc(nLWORD);
F_AnyFunc(sSTRING);
F_AnyFunc(wsWSTRING);
F_AnyFunc(dtDATEANDTIME);
F_AnyFunc(tdTIMEOFDAY);
F_AnyFunc(fREAL);
F_AnyFunc(fLREAL);
F_AnyFunc(nUSINT);
F_AnyFunc(nUINT);
F_AnyFunc(nUDINT);
F_AnyFunc(nULINT);
F_AnyFunc(nSINT);
F_AnyFunc(nINT);
F_AnyFunc(nDINT);
F_AnyFunc(nLINT);

F_AnyBitFunc(nBYTE);
F_AnyBitFunc(nWORD);
F_AnyBitFunc(nDWORD);
F_AnyBitFunc(nLWORD);

F_AnyStringFunc(sSTRING);
F_AnyStringFunc(wsWSTRING);

F_AnyDateFunc(dtDATEANDTIME);
F_AnyDateFunc(dDATE);
F_AnyDateFunc(tdTIMEOFDAY);

F_AnyNumFunc(fREAL);
F_AnyNumFunc(fLREAL);
F_AnyNumFunc(nUSINT);
F_AnyNumFunc(nUINT);
F_AnyNumFunc(nUDINT);
```

```
F_AnyNumFunc(nULINT);
F_AnyNumFunc(nSINT);
F_AnyNumFunc(nINT);
F_AnyNumFunc(nDINT);
F_AnyNumFunc(nLINT);

F_AnyRealFunc(fREAL);
F_AnyRealFunc(fLREAL);

F_AnyIntFunc(nUSINT);
F_AnyIntFunc(nUINT);
F_AnyIntFunc(nUDINT);
F_AnyIntFunc(nULINT);
F_AnyIntFunc(nSINT);
F_AnyIntFunc(nINT);
F_AnyIntFunc(nDINT);
F_AnyIntFunc(nLINT);
```

Beispiel 3: Vergleich zweier übergebener Variablen

Die Funktion vergleicht, ob die zwei übergebenen Variablen den gleichen Typ und den gleichen Wert haben.

```
FUNCTION F_GenericCompare : BOOL
VAR_INPUT
    any1 : ANY;
    any2 : ANY;
END_VAR
VAR
    nCount: DINT;
END_VAR
```

```
IF any1.typeclass <> any2.typeclass THEN
   RETURN;
END_IF

IF any1.diSize <> any2.diSize THEN
    RETURN;
END_IF

// Byte comparison
FOR nCount := 0 TO any1.diSize-1 DO
    IF any1.pvalue[nCount] <> any2.pvalue[nCount] THEN
        RETURN;
    END_IF
END_FOR

F_GenericCompare := TRUE;
```

Beispiel 4: Feststellung des übergebenen Datentyps

Die Funktion überprüft, ob die übergebene Variable vom Typ `REAL` oder `LREAL` ist. Wenn dies der Fall ist, wird der Wert der Variablen gerundet.

```
// function to round transfer parameters of the type REAl and LREAL (other types are detected as
invalid)
FUNCTION F_RoundFloatingValue : INT
VAR_INPUT
    anyIn        : ANY;              // input variable of the type ANY
END_VAR
VAR
    pAnyReal     : POINTER TO REAL;  // pointer to a variable of the type REAL
    pAnyLReal    : POINTER TO LREAL; // pointer to a variable of the type LREAL
END_VAR
VAR_OUTPUT
    bInvalidType : BOOL;             // output variable with value TRUE if the transferred
parameter has an invalid type
END_VAR
```

```
// round floating value for a transfer parameter of the type REAL
IF (anyIn.TypeClass = __SYSTEM.TYPE_CLASS.TYPE_REAL) THEN
    pAnyReal             := anyIn.pValue;
    F_RoundFloatingValue  := REAL_TO_INT(pAnyReal^);

// round floating value for a transfer parameter of the type LREAL
ELSIF (anyIn.TypeClass = __SYSTEM.TYPE_CLASS.TYPE_LREAL) THEN
    pAnyLReal            := anyIn.pValue;
    F_RoundFloatingValue  := LREAL_TO_INT(pAnyLReal^);

// inform about invalid type if the transfer parameter is not of the type REAL or LREAL
```

```
ELSE
    bInvalidType          := TRUE;
END_IF
```

16.5.11 Spezialdatentypen UXINT, XINT, XWORD und PVOID

Variablen mit diesen Datentypen werden zielsystemabhängig in einen plattformkonformen Datentyp konvertiert.

TwinCAT unterstützt Systeme mit Adressregistern von 32 Bit Breite und 64 Bit Breite. Um den IEC-Code soweit wie möglich vom Zielsystem unabhängig zu machen, können Sie die unten genannten „Pseudo“-Datentypen `UXINT`, `XINT`, `XWORD` und `PVOID` verwenden. Der Compiler prüft, welcher Zielsystemtyp aktuell verwendet wird, und konvertiert diese Datentypen entsprechend in die passenden Standard-Datentypen. Außerdem stehen für Variablen dieses Datentyps Typkonvertierungsoperatoren zur Verfügung.

Folgende „Pseudo“-Datentypen sind verfügbar:

	Typkonvertierung auf 64-Bit-Plattformen	Typkonvertierung auf 32-Bit-Plattformen
XINT bzw. __XINT	LINT	DINT
UXINT bzw. __UXINT	ULINT	UDINT
XWORD bzw. __XWORD	LWORD	DWORD
PVOID	UXINT	

16.5.12 Zeiger / POINTER

Ein Pointer speichert zur Laufzeit die Speicheradresse von Objekten wie beispielsweise Variablen oder Funktionsbausteininstanzen.

Syntax

```
<pointer name>: POINTER TO <data type> | <data unit type> | <function block
name>;
```

Beispiel

```
FUNCTION_BLOCK FB_Sample
VAR
    pSample : POINTER TO INT;
    nVar1 : INT := 5;
    nVar2 : INT;
END_VAR
```

```
pSample := ADR(nVar1);  // pointer pSample is assigned to address of nVar1
nVar2 := pSample^;      //
value 5 of nVar1 is assigned to variable nVar2 by dereferencing of pointer pSample
```

Einen Pointer zu dereferenzieren bedeutet, den Wert zu erhalten, auf den der Pointer zeigt. Ein Pointer wird dereferenziert, indem der Inhaltsoperator ^ an den Pointer-Bezeichner angehängt wird, beispielsweise pSample^ im oben gezeigten Beispiel. Um einem Pointer die Adresse eines Objekts zuzuweisen, verwenden Sie den Adressoperator ADR: ADR(nVar1).

Im Onlinebetrieb können Sie mit dem Befehl Gehe zur Referenz [▶ 921] von einem Pointer zur Deklarationsstelle des referenzierten Objekts springen (verfügbar ab TC3.1 Build 4026).

Wenn ein Pointer auf einen lokierte Eingangsvariable verwendet wird, gilt der Zugriff als schreibender Zugriff. Dies führt bei der Codeerzeugung zu der Compilerwarnung "<Pointer Name> ist kein gültiges Zuordnungsziel".

Beispiel: `pTest := ADR(nInput);`

Wenn Sie ein Konstrukt dieser Art benötigen, müssen Sie den Eingangswert (nInput) zuerst auf eine Variable mit Schreibzugriff kopieren.

Indexzugriff auf Pointer

TwinCAT erlaubt den Indexzugriff [ ] auf Variablen vom Typ POINTER, ebenso wie auf die Datentypen STRING oder WSTRING.

Auf die Daten, auf die der Pointer zeigt, kann auch zugegriffen werden, indem der Klammeroperator [ ] an den Pointer-Bezeichner angehängt wird, beispielsweise `pData[i]`. Der Basisdatentyp des Pointers bestimmt den Datentyp und die Größe der indizierten Komponente. Der Indexzugriff auf den Pointer erfolgt dabei arithmetisch, indem zur Adresse des Pointers der indexabhängige Offset `i * SIZEOF(<base type>)` addiert wird. Gleichzeitig wird der Pointer implizit dereferenziert. Berechnung:
`pData[i] :=(pData + i * SIZEOF(INT))^;`

Indexzugriff auf STRING

Wenn Sie den Indexzugriff bei einer Variablen vom Typ STRING verwenden, erhalten Sie das Zeichen am Offset des Indexausdrucks. Das Ergebnis ist vom Typ BYTE. Beispielsweise gibt `sData[i]` das i-te Zeichen der Zeichenkette `sData` als SINT (ASCII) zurück.

Indexzugriff WSTRING

Wenn Sie den Indexzugriff bei einer Variablen vom Typ WSTRING verwenden, erhalten Sie das Zeichen am Offset des Indexausdrucks. Das Ergebnis ist vom Typ WORD. Beispielsweise gibt `wsData[i]` das i-te Zeichen der Zeichenkette als INT (Unicode) zurück.

Pointer subtrahieren

Das Ergebnis der Differenz zweier Pointer ist ein Wert vom Typ DWORD, auch auf 64-Bit-Plattformen, wenn die Pointer 64-Bit-Pointer sind.

Beachten Sie die Möglichkeit, Referenzen zu verwenden.Die Verwendung von Referenzen hat den Vorteil, dass Typsicherheit garantiert ist. Das ist bei Pointern nicht der Fall.

Der Speicherzugriff von Pointern während der Laufzeit kann durch die implizite Überwachungsfunktion CheckPointer geprüft werden.

Siehe auch:

- REFERENCE [▶ 807]
- Pointer Checks (POU CheckPointer) [▶ 177]

16.5.13 Datentyp __SYSTEM.ExceptionCode

Siehe auch: __TRY, __CATCH, __FINALLY, __ENDTRY [▶ 774]

```
TYPE ExceptionCode :
(
    RTSEXCPT_UNKNOWN                                := 16#FFFFFFFF,
    RTSEXCPT_NOEXCEPTION                            := 16#00000000,
    RTSEXCPT_WATCHDOG                               := 16#00000010,
    RTSEXCPT_HARDWAREWATCHDOG                       := 16#00000011,
    RTSEXCPT_IO_CONFIG_ERROR                        := 16#00000012,
    RTSEXCPT_PROGRAMCHECKSUM                        := 16#00000013,
    RTSEXCPT_FIELDBUS_ERROR                         := 16#00000014,
    RTSEXCPT_IOUPDATE_ERROR                         := 16#00000015,
    RTSEXCPT_CYCLE_TIME_EXCEED                      := 16#00000016,
    RTSEXCPT_ONLCHANGE_PROGRAM_EXCEEDED             := 16#00000017,
    RTSEXCPT_UNRESOLVED_EXTREFS                     := 16#00000018,
    RTSEXCPT_DOWNLOAD_REJECTED                      := 16#00000019,
    RTSEXCPT_BOOTPROJECT_REJECTED_DUE_RETAIN_ERROR  := 16#0000001A,
    RTSEXCPT_LOADBOOTPROJECT_FAILED                 := 16#0000001B,
    RTSEXCPT_OUT_OF_MEMORY                          := 16#0000001C,
    RTSEXCPT_RETAIN_MEMORY_ERROR                    := 16#0000001D,
    RTSEXCPT_BOOTPROJECT_CRASH                      := 16#0000001E,
```

```
    RTSEXCPT_BOOTPROJECTTARGETMISMATCH              := 16#00000021,
    RTSEXCPT_SCHEDULEERROR                          := 16#00000022,
    RTSEXCPT_FILE_CHECKSUM_ERR                      := 16#00000023,
    RTSEXCPT_RETAIN_IDENTITY_MISMATCH               := 16#00000024,
    RTSEXCPT_IEC_TASK_CONFIG_ERROR                  := 16#00000025,
    RTSEXCPT_APP_TARGET_MISMATCH                    := 16#00000026,
    RTSEXCPT_ILLEGAL_INSTRUCTION                    := 16#00000050,
    RTSEXCPT_ACCESS_VIOLATION                       := 16#00000051,
    RTSEXCPT_PRIV_INSTRUCTION                       := 16#00000052,
    RTSEXCPT_IN_PAGE_ERROR                          := 16#00000053,
    RTSEXCPT_STACK_OVERFLOW                         := 16#00000054,
    RTSEXCPT_INVALID_DISPOSITION                    := 16#00000055,
    RTSEXCPT_INVALID_HANDLE                         := 16#00000056,
    RTSEXCPT_GUARD_PAGE                             := 16#00000057,
    RTSEXCPT_DOUBLE_FAULT                           := 16#00000058,
    RTSEXCPT_INVALID_OPCODE                         := 16#00000059,
    RTSEXCPT_MISALIGNMENT                           := 16#00000100,
    RTSEXCPT_ARRAYBOUNDS                            := 16#00000101,
    RTSEXCPT_DIVIDEBYZERO                           := 16#00000102,
    RTSEXCPT_OVERFLOW                               := 16#00000103,
    RTSEXCPT_NONCONTINUABLE                         := 16#00000104,
    RTSEXCPT_PROCESSORLOAD_WATCHDOG                 := 16#00000105,
    RTSEXCPT_FPU_ERROR                              := 16#00000150,
    RTSEXCPT_FPU_DENORMAL_OPERAND                   := 16#00000151,
    RTSEXCPT_FPU_DIVIDEBYZERO                       := 16#00000152,
    RTSEXCPT_FPU_INEXACT_RESULT                     := 16#00000153,
    RTSEXCPT_FPU_INVALID_OPERATION                  := 16#00000154,
    RTSEXCPT_FPU_OVERFLOW                           := 16#00000155,
    RTSEXCPT_FPU_STACK_CHECK                        := 16#00000156,
    RTSEXCPT_FPU_UNDERFLOW                          := 16#00000157,
    RTSEXCPT_VENDOR_EXCEPTION_BASE                  := 16#00002000,
    RTSEXCPT_USER_EXCEPTION_BASE                    := 16#00010000
) UDINT ;
END_TYPE
```

16.5.14 Schnittstellenzeiger / INTERFACE

Ein Schnittstellenzeiger (auch Interfacepointer oder Schnittstellenvariable genannt) speichert die Adresse einer Funktionstabelle einer Funktionsbausteininstanz, welche die Schnittstelle implementiert. Über diese Indirektion können mit einem Schnittstellenzeiger alle Methoden und Eigenschaften aufgerufen werden, die von der Schnittstelle definiert werden. Daher müssen diese Methoden und Eigenschaften von dem Funktionsbaustein implementiert werden, dessen Instanz dem Schnittstellenzeiger zugewiesen wird.

Einem Schnittstellenzeiger muss vor der Verwendung immer eine Instanz eines Funktionsbausteines zugewiesen werden, der die entsprechende Schnittstelle implementiert.

Vor der Verwendung wird empfohlen, den Schnittstellenzeiger auf Gültigkeit (<> 0) zu prüfen.

Syntax:

```
<Kennzeichner> : <Interfacetyp>;
```

```
FUNCTION_BLOCK <Funktionsbausteintyp> IMPLEMENTS <Interfacetyp>
```

Beispiel:

```
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample
```

```
VAR
    ipSample : I_Sample;
    fbSample : FB_Sample;
    nResult  : INT;
END_VAR
```

```
ipSample := fbSample;

// calling a method of this interface
IF ipSample <> 0 THEN
    nResult := ipSample.Add(3, 6); // result is 9
END_IF
```

Die Verwendung von Schnittstellen und Schnittstellenzeigern wird ausführlich im Abschnitt Objektorientiert programmieren > Objekt Schnittstelle [▶ 107] erläutert.

Typkonvertierungen sind mit dem Operator __QUERYINTERFACE() [▶ 772] möglich.

16.5.15 REFERENCE

Eine Referenz verweist implizit auf ein anderes Objekt. Beim Zugriff wird die Referenz implizit dereferenziert, und benötigt deswegen keinen speziellen Inhaltsoperator ^ wie ein Pointer.

Syntax

```
<identifier> : REFERENCE TO <data type> ;
<data type>  : base type of the reference
```

Beispieldeklaration:

```
PROGRAM_MAIN
VAR
    refInt  : REFERENCE TO INT;
    nA      : INT;
    nB      : INT;
END_VAR
```

Sie können nun refInt als „Alias“ für Variablen des Typs INT verwenden.

Zuweisung:

Die Adresse der Referenz müssen Sie über eine separate Zuweisungsoperation mit Hilfe des Zuweisungsoperators REF= [▶ 807] setzen. Eine Ausnahme hiervon besteht, wenn ein Input ein REFERENCE TO ist und der Input innerhalb des Aufrufs übergeben wird. Dann wird anstelle des Zuweisungsoperators REF= der normale Zuweisungsoperator := verwendet.

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    refInput1  : REFERENCE TO INT;
    refInput2  : REFERENCE TO INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbSample   : FB_Sample;
    n1         : INT;
    n2         : INT;
END_VAR
```

```
fbSample.refInput1 REF= n1;
fbSample(refInput2   := n2);
```

Ob eine Referenz auf einen gültigen Wert (zum Beispiel ungleich 0) zeigt, können Sie mithilfe eines speziellen Operators überprüfen (siehe Referenzen auf Gültigkeit prüfen [▶ 808]).

Anwendungsbeispiel:

```
refInt REF= nA;         // refInt points now to nA
refInt   := 12;         // nA has got the value 12
nB       := refInt * 2; // nB has got the value 24
refInt REF= nB;         // refInt points now to nB
refInt   := nA / 2;     // nB has got the value 6
refInt REF= 0;          // explicit initialisation of the reference
```

TwinCAT initialisiert Referenzen (mit 0).

Wenn eine Referenz auf eine lokierte Eingangsvariable verweist, gilt der Zugriff (beispielsweise rInput REF=Input;) als schreibender Zugriff. Dies ist nicht möglich und führt bei der Codeerzeugung zu einem Compilerfehler.

Zuweisungsoperator REF=

Der Operator erzeugt eine Referenz [▶ 807] (Pointer) auf einen Wert.

Syntax:

```
<variable name> REF= <variable name>
```

Beispiel:

```
PROGRAM MAIN
VAR
    refA  : REFERENCE TO ST_Sample;
    stA   : ST_Sample;
    refB  : REFERENCE TO ST_Sample;
    stB1  : ST_Sample;
    stB2  : ST_Sample;
END_VAR
```

```
refA REF= stA;   // represents => refA  := ADR(stA);
refB REF= stB1;  // represents => refB  := ADR(stB1);
refA := refB;    // represents => refA^ := refB^; (value assignment of refB as refA and refB are
implicitly dereferenced)
refB := stB2;    // represents => refB^ := stB2; (value assignment of stB2 as refB is implicitly
dereferenced)
END_VAR
```

Ungültige Deklarationen

```
PROGRAM MAIN
VAR
     aTest      : ARRAY[0..9] OF REFERENCE TO INT;
     pTest      : POINTER TO REFERENCE TO INT;
     refTestRef : REFERENCE TO REFERENCE TO INT;
     refTestBit : REFERENCE TO BIT;
END_VAR
```

Ein Referenztyp darf nicht als Basistyp eines Arrays, Pointers oder einer Referenz verwendet werden. Außerdem darf eine Referenz nicht auf eine Bit-Variable verweisen. Solche Konstrukte erzeugen Compilerfehler.

Vergleich von Referenz und Pointer

Eine Referenz hat gegenüber einem Pointer folgende Vorteile:

- Einfachere Nutzung:

  Eine Referenz kann direkt, ohne Dereferenzierung auf die Inhalte des referenzierten Objekts zugreifen.
- Schönere und einfachere Syntax bei Übergabe von Werten:

  Aufruf eines Funktionsbausteins, der statt einem Pointer eine Referenz ohne Adressoperator übergibt.

  Beispiel: `FB_Test1(refInput := nValue);`
  statt: `FB_Test2(pInput := ADR(nValue));`
- Typsicherheit:

  Der Compiler prüft bei der Zuweisung zweier Referenzen, ob deren Basistypen übereinstimmen. Bei Pointern wird dies nicht geprüft.

Referenzen auf Gültigkeit prüfen

Sie können den Operator `__ISVALIDREF` verwenden, um zu prüfen, ob eine Referenz auf einen gültigen Wert weist, das heißt auf einen Wert ungleich 0.

Syntax:

```
<boolesche Variable> := __ISVALIDREF(<mit REFERENCE TO <datatype> deklarierter Kennzeichner>);
```

Die boolesche Variable wird TRUE, wenn die Referenz auf einen gültigen Wert zeigt, andernfalls FALSE.

Beispiel

```
PROGRAM_MAIN
VAR
    nVar      : INT;
    refInt1   : REFERENCE TO INT;
    refInt2   : REFERENCE TO INT;
    bTestRef1 : BOOL := FALSE;
    bTestRef2 : BOOL := FALSE;
END_VAR
```

```
nVar       := nVar + 1;
refInt1  REF= nVar;
refInt2  REF= 0;
bTestRef1  := __ISVALIDREF(refInt1); (* becomes TRUE, because refInt1 points to nVar, which is non-
zero *)
bTestRef2  := __ISVALIDREF(refInt2); (* becomes FALSE, because refInt2 is set to 0 *)
```

Die implizite Überwachungsfunktion Checkpointer wirkt auch auf Variablen vom Typ `REFERENCE` in gleicher Weise wie auf Zeigervariablen.

Siehe auch:

- Pointer Checks (POU CheckPointer) [▶ 177]

Sehen Sie dazu auch

Zeiger / POINTER [▶ 804]

16.5.16 ARRAY

Ein Array ist eine Sammlung von Datenelementen des gleichen Datentyps. TwinCAT unterstützt ein- und mehrdimensionale Arrays von fester oder variabler Länge.

16.5.16.1 Array mit fester Länge

Sie können Arrays im Deklarationsteil einer POU oder in globalen Variablenlisten deklarieren.

Syntax zur Deklaration eines eindimensionalen Arrays

```
<variable name> : ARRAY[ <dimension> ] OF <data type> ( := <initialization> )? ;
```

```
<dimension> : <lower index bound>..<upper index bound>
<data type> : elementary data types | user defined data types | function block types
// (...)? : Optional
```

Syntax zur Deklaration eines mehrdimensionalen Arrays

```
<variable name> : ARRAY[ <1st dimension> ( , <next dimension> )
+ ] OF <data type> ( := <initialization> )? ;
```

```
<1st dimension> : <1st lower index bound>..<1st upper index bound>
<next dimension> : <next lower index bound>..<next upper index bound>
<data type> : elementary data types | user defined data types | function block types
// (...)+ : One or more further dimensions
// (...)? : Optional
```

Die Indexgrenzen sind ganzzahlige Zahlen, maximal des Datentyps DINT.

Syntax zum Datenzugriff

```
<variable name>[ <index of 1st dimension> ( , <index of next dimension> )* ]
```

```
// (...)* : 0, one or more further dimensions
```

Beachten Sie die Möglichkeit, die Verletzung der Feldgrenzen während der Laufzeit durch die implizite Überwachungsfunktion CheckBounds zu überwachen.

Siehe auch:

- Bound Checks (POU CheckBounds) [▶ 173]

Einstellen des Monitoring-Bereichs für große Arrays

Wenn ein Array über mehr als 1000 Elemente verfügt, werden in der Online-Ansicht standardmäßig 1000 Elemente angezeigt. Diesen Monitoring-Bereich können Sie ändern, indem Sie in der Online-Ansicht im Deklarationsbereich auf die Arraydeklaration doppelklicken. Daraufhin öffnet sich ein Dialog, in welchem Sie den Start- und Endindex des gewünschten Monitoring-Bereichs einstellen können.

Beachten Sie, dass die Anzeigeperformance abnimmt, je mehr Elemente gleichzeitig angezeigt werden (maximal 20000 Elemente).

Beispiel: 1-dimensionales Array

Deklaration 1:

Eindimensionales Array von 10 Integerelementen

Untere Indexgrenze: 0
Obere Indexgrenze: 9

```
VAR
    aCounter : ARRAY[0..9] OF INT;
END_VAR
```

Initialisierung 1:

```
aCounter : ARRAY[0..9] OF INT := [0, 10, 20, 30, 40, 50, 60, 70, 80, 90];
```

Datenzugriff 1:

Der lokalen Variablen wird der Wert 20 zugewiesen.

```
nLocalVariable := aCounter[2];
```

Deklaration 2:

```
VAR CONSTANT
    cMin    : INT := 0;
    cMax    : INT := 5;
END_VAR
VAR
    aSample : ARRAY [cMin..cMax] OF BOOL;
END_VAR
```

Datenzugriff 2:

Auf das Array wird mit Hilfe einer Indexvariablen zugegriffen. Vor dem Zugriff wird überprüft, ob der Wert der Indexvariable innerhalb der gültigen Arraygrenzen liegt.

```
IF nIndex >= cMin AND nIndex <= cMax THEN
    bValue := aSample[nIndex];
END_IF
```

Beispiel: 2-dimensionales Array

Deklaration:

1te Dimension: 1 bis 2
2te Dimension: 3 bis 4

```
VAR
    aCardGame : ARRAY[1..2, 3..4] OF INT;
END_VAR
```

Initialisierung:

```
aCardGame : ARRAY[1..2, 3..4] OF INT := [2(10),2(20)]; // Short notation for [10, 10, 20, 20]
```

Datenzugriff:

```
nLocal1 := aCardGame[1, 3]; // Assignment of 10
nLocal2 := aCardGame[2, 4]; // Assignment of 20
```

Beispiel: 3-dimensionales Array

Deklaration:

1te Dimension: 1 bis 2
2te Dimension: 3 bis 4
3te Dimension: 5 bis 6
2 * 2 * 2 = 8 Arrayelemente

```
VAR
    aCardGame : ARRAY[1..2, 3..4, 5..6] OF INT;
END_VAR
```

Initialisierung 1:

```
aCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [10, 20, 30, 40, 50, 60, 70, 80];
```

Datenzugriff 1:

```
nLocal1 := aCardGame[1, 3, 5]; // Assignment of 10
nLocal2 := aCardGame[2, 3, 5]; // Assignment of 20
nLocal3 := aCardGame[1, 4, 5]; // Assignment of 30
nLocal4 := aCardGame[2, 4, 5]; // Assignment of 40
nLocal5 := aCardGame[1, 3, 6]; // Assignment of 50
nLocal6 := aCardGame[2, 3, 6]; // Assignment of 60
nLocal7 := aCardGame[1, 4, 6]; // Assignment of 70
nLocal8 := aCardGame[2, 4, 6]; // Assignment of 80
```

Initialisierung 2:

```
aCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [2(10), 2(20), 2(30), 2(40)]; // Short notation for [1
0, 10, 20, 20, 30, 30, 40, 40]
```

Datenzugriff 2:

```
nLocal1 := aCardGame[1, 3, 5]; // Assignment of 10
nLocal2 := aCardGame[2, 3, 5]; // Assignment of 10
nLocal3 := aCardGame[1, 4, 5]; // Assignment of 20
nLocal4 := aCardGame[2, 4, 5]; // Assignment of 20
nLocal5 := aCardGame[1, 3, 6]; // Assignment of 30
nLocal6 := aCardGame[2, 3, 6]; // Assignment of 30
nLocal7 := aCardGame[1, 4, 6]; // Assignment of 40
nLocal8 := aCardGame[2, 4, 6]; // Assignment of 40
```

Beispiel: 3-dimensionales Array einer benutzerdefinierten Struktur

Deklaration:

Das Array aData besteht aus insgesamt 3 * 3 * 10 = 90 Arrayelementen des Datentyps ST_Data.

```
TYPE ST_Data
STRUCT
    n1 : INT;
    n2 : INT;
    n3 : DWORD;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    aData : ARRAY[1..3, 1..3, 1..10] OF ST_Data;
END_VAR
```

Teilweises Initialisieren:

Im Beispiel werden nur die ersten 3 Elemente explizit initialisiert. Elemente, denen explizit kein Initialisierungswert zugewiesen wird, werden intern mit dem Standardwert des Basisdatentyps initialisiert. Somit werden die Strukturkomponenten ab dem Element aData[2, 1, 1] mit 0 initialisiert.

```
aData : ARRAY[1..3, 1..3, 1..10] OF ST_Data := [(n1 := 1, n2 := 10, n3 := 16#00FF),
                                                (n1 := 2, n2 := 20, n3 := 16#FF00),
                                                (n1 := 3, n2 := 30, n3 := 16#FFFF)];
```

Datenzugriff:

```
nLocal1 := aData[1,1,1].n1; // Assignment of 1
nLocal2 := aData[3,1,1].n3; // Assignment of 16#FFFF
```

Beispiel: Array eines Funktionsbausteins

Deklaration:

Das Array aObjects besteht aus 4 Elementen. Jedes Element instanziiert einen Funktionsbaustein FB_Object.

```
FUNCTION BLOCK FB_Object
VAR
    nCounter : INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    aObjects : ARRAY[1..4] OF FB_Object;
END_VAR
```

Funktionsaufruf:

```
aObjects[2]();
```

Beispiel: Array eines Funktionsbausteins mit FB_init-Methode

Deklaration:

Implementierung von FB_Sample mit Methode FB_init

```
FUNCTION_BLOCK FB_Sample
VAR
    _nId : INT;
    _fIn : LREAL;
END_VAR
```

Der Funktionsbaustein FB_Sample hat eine Methode FB_init, die zwei Parameter benötigt.

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;
    bInCopyCode  : BOOL;
    nId          : INT;
    fIn          : LREAL;
END_VAR
```

```
_nId := nId;
_fIn := fIN;
```

Deklaration des Arrays mit Initialisierung:

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample (nId := 11, fIn := 33.44);
    aSample  : ARRAY[0..1, 0..1] OF FB_Sample [(nId := 12, fIn := 11.22),
                                               (nId := 13, fIn := 22.33),
                                               (nId := 14, fIn := 33.55),
                                               (nId := 15, fIn := 11.22)];
END_VAR
```

16.5.16.2 Array von Arrays

Die Deklaration eines “Array von Arrays” ist eine alternative Schreibweise für multidimensionale Arrays. Eine Sammlung von Elementen wird dabei verschachtelt, statt die Elemente zu dimensionieren. Die Schachteltiefe kann beliebig sein.

Syntax zur Deklaration

```
<variable name> : ARRAY[<first>] ( OF ARRAY[<next>] )+ OF <data type> ( := <initialization> )? ;
```

```
<first> : <first lower index bound>..<first upper index bound>
<next> : <lower index bound>..<upper index bound> // one or more arrays
<data type> : elementary data types | user defined data types | function block types
// (...)+ : One or more further arrays
// (...)? : Optional
```

Syntax zum Datenzugriff

```
<variable name>[<index of first array>] ( [<index of next array>] )+ ;
```

```
// (...)* : 0, one or more further arrays
```

Beispiel

Die Variablen aiPoints und ai2Boxes sammeln die gleichen Datenelemente, aber die Schreibweisen bei der Deklaration und beim Datenzugriff unterscheiden sich.

```
PROGRAM MAIN
VAR
    aPoints : ARRAY[1..2,1..3] OF INT := [1,2,3,4,5,6];
    a2Boxes : ARRAY[1..2] OF ARRAY[1..3] OF INT := [ [1, 2, 3], [ 4, 5, 6]];
    a3Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF INT := [ [ [1, 2, 3, 4], [5, 6, 7, 8], [9
, 10, 11, 12] ], [ [13, 14, 15, 16], [ 17, 18, 19, 20], [21, 22, 23, 24] ] ];
    a4Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF ARRAY[1..5] OF INT;
END_VAR
```

```
aPoints[1, 2] := 1200;
a2Boxes[1][2] := 1200;
```

aPoints	ARRAY [1..2, 1..3] OF INT	
aPoints[1, 1]	INT	1
aPoints[1, 2]	INT	1200
aPoints[1, 3]	INT	3
aPoints[2, 1]	INT	4
aPoints[2, 2]	INT	5
aPoints[2, 3]	INT	6
a2Boxes	ARRAY [1..2] OF ARRAY [1..3] OF INT	
a2Boxes[1]	ARRAY [1..3] OF INT	
a2Boxes[1][1]	INT	1
a2Boxes[1][2]	INT	1200
a2Boxes[1][3]	INT	3
a2Boxes[2]	ARRAY [1..3] OF INT	
a2Boxes[2][1]	INT	4
a2Boxes[2][2]	INT	5
a2Boxes[2][3]	INT	6

16.5.16.3 Array mit variabler Länge

In Funktionsbausteinen, Funktionen oder Methoden können Sie im Deklarationsabschnitt VAR_IN_OUT Arrays mit variabler Länge deklarieren. Um zur Laufzeit die Indexgrenzen des tatsächlich verwendeten Arrays zu ermitteln, stehen Ihnen die Operatoren LOWER_BOUND und UPPER_BOUND zur Verfügung. LOWER_BOUND liefert die Untergrenze, UPPER_BOUND liefert die Obergrenze.

An ein Array mit variabler Länge dürfen nur statisch deklarierte Arrays übergeben werden. Dynamische Arrays, die mit Hilfe des Operators __NEW erzeugt wurden, dürfen nicht übergeben werden.

Syntax zur Deklaration eines eindimensionalen Arrays mit variabler Länge

```
<variable name> : ARRAY[*] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
// (...)? : Optional
```

Syntax zur Deklaration eines mehrdimensionalen Arrays mit variabler Länge

```
<variable name> : ARRAY[* ( , * )+ ] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
// (...)+ : One or more further dimensions
// (...)? : Optional
```

Syntax der Operatoren zur Grenzindexberechnung

```
LOWER_BOUND( <variable name> , <dimension number> )
UPPER_BOUND( <variable name> , <dimension number> )
```

Beispiel

Die Funktion F_SUM addiert die Integerwerte der Arrayelemente und gibt als Ergebnis die ermittelte Summe zurück. Die Summe wird über alle zur Laufzeit vorhandenen Arrayelemente berechnet. Da die tatsächliche Anzahl an Arrayelementen erst zur Laufzeit bekannt sein wird, wird die lokale Variable als eindimensionales Array variabler Länge deklariert. An diese Additionsfunktion können Arrays mit verschiedener, fester Länge übergeben werden.

```
FUNCTION F_Sum : DINT;
VAR_IN_OUT
    aData     : ARRAY [*] OF INT;
END_VAR
VAR
    i, nSum   : DINT;
END_VAR
```

```
nSum := 0;

FOR i := LOWER_BOUND(aData,1) TO UPPER_BOUND(aData,1) DO
    nSum := nSum + aData[i];
END_FOR;

F_Sum := nSum;
```

16.5.17 Struktur

Eine Struktur ist ein benutzerdefinierter Datentyp und fasst mehrere Variablen mit beliebigen Datentypen zu einer logischen Einheit zusammen. Die innerhalb einer Struktur deklarierten Variablen werden als Komponenten bezeichnet.

Die Deklaration einer Struktur nehmen Sie in einem DUT-Objekt vor, das Sie über den Befehl **Hinzufügen > DUT** im Kontextmenü des SPS-Projektbaums im Projekt anlegen.

Syntax:

```
TYPE <structure name> :
STRUCT
    (<variable declaration optional with initialization>)+
END_STRUCT
END_TYPE
```

Der <structure name> ist ein Bezeichner, der im gesamten Projekt gültig ist, so dass Sie ihn wie einen Standarddatentyp verwenden können. Außerdem können Sie beliebig viele Variablen (mindestens eine) deklarieren, die optional durch eine Initialisierung ergänzt werden.

Weiterhin können Sie Strukturen verschachteln. Das bedeutet, dass Sie eine Strukturkomponente mit einem bestehenden Strukturtypen deklarieren. Dabei ist die einzige Beschränkung, dass Sie der Variablen (Strukturkomponente) keine Adressen zuweisen dürfen (die AT-Deklaration ist hier nicht zulässig).

Beispiel: Strukturdeklaration

```
TYPE ST_POLYGONLINE :
STRUCT
    aStart  : ARRAY[1..2] OF INT := [-99, -99];
    aPoint1 : ARRAY[1..2] OF INT;
    aPoint2 : ARRAY[1..2] OF INT;
    aPoint3 : ARRAY[1..2] OF INT;
    aPoint4 : ARRAY[1..2] OF INT;
    aEnd    : ARRAY[1..2] OF INT := [99, 99];
END_STRUCT
END_TYPE
```

8-Byte-Alignment

Mit TwinCAT 3 wurde ein 8-Byte-Alignment eingeführt. Achten Sie auf ein passendes Alignment, wenn Daten als gesamter Speicherblock mit anderen Steuerungen oder Softwarekomponenten ausgetauscht werden (siehe Alignment [▶ 826]).

Erweitern einer Typdeklaration

Ausgehend von einer bestehenden Struktur wird eine weitere Struktur deklariert. Die erweiterte Struktur besitzt zusätzlich zu den eigenen Komponenten die gleichen Strukturkomponenten wie die Basisstruktur.

Syntax

```
TYPE <structure name> EXTENDS <basis structure> :
STRUCT
    (<variable declaration optional with initialization>)+
END_STRUCT
END_TYPE
```

Beispiel: Strukturdeklaration

```
TYPE ST_PENTAGON EXTENDS ST_POLYGONLINE :
STRUCT
    aPoint5 : ARRAY[1..2] OF INT;
END_STRUCT
END_TYPE
```

Siehe auch:

- Objekt DUT [▶ 77]

Deklaration und Initialisierung von Strukturen

Beispiel

```
PROGRAM Line
VAR
    stPolygon : ST_POLYGONLINE := (aStart:=[1,1], aPoint1:=[5,2], aPoint2:=[7,3], aPoint3:=[8,5],
aiPoint4:=[5,7], aEnd:=[1,1]);
    stPentagon : ST_PENTAGON := (aStart:=[0,0], aPoint1:=[1,1], aPoint2:=[2,2], aPoint3:=[3,3],
aPoint4:=[4,4], aPoint5:=[5,5], aEnd:=[0,0]);
END_VAR
```

Sie dürfen keine Initialisierungen mit Variablen verwenden. Ein Beispiel für die Initialisierung eines Arrays einer Struktur finden Sie auf der Hilfeseite zum Datentyp ARRAY.

Siehe auch:

- Objekt DUT [▶ 77]

Zugriff auf eine Strukturkomponente

Sie greifen auf eine Strukturkomponente gemäß folgender Syntax zu:

```
<variable name>.<component name>
```

Beispiel

```
PROGRAM Polygon
VAR
    stPolygon : ST_POLYGONLINE := := (aStart:=[1,1], aPoint1:=[5,2], aPoint2:=[7,3], aPoint3:=[8,5],
aiPoint4:=[5,7], aEnd:=[1,1]);
    nPoint: INT;
END_VAR
```

```
// Assigns 5 to nPoint
nPoint := stPolygon.aPoint1[1];
```

Ergebnis: nPoint = 5

Symbolischer Bitzugriff in Strukturvariablen

Sie können eine Struktur mit Variablen des Datentyps BIT deklarieren, um einzelne Bits zu einer logischen Einheit zusammenzufassen. Dann können Sie einzelne Bits symbolisch über einen Namen (statt über den Bitindex) adressieren.

Syntax Deklaration:

```
TYPE <structure name> :
STRUCT
    ( <bit name> : BIT; )+
END_STRUCT
END_TYPE
```

Syntax Bitzzugriff:

```
<Strukturname> . <Bitname>
```

Beispiel:

Strukturdeklaration

```
TYPE ST_CONTROL :
STRUCT
    bitOperationEnabled  : BIT;
    bitSwitchOnActive    : BIT;
    bitEnableOperation   : BIT;
    bitError             : BIT;
    bitVoltageEnabled    : BIT;
    bitQuickStop         : BIT;
    bitSwitchOnLocked    : BIT;
```

```
    bitWarning          : BIT;
END_STRUCT
END_TYPE
```

Bitzugriff

```
FUNCTION_BLOCK FB_Controller
VAR_INPUT
    bStart : BOOL;
END_VAR
VAR_OUTPUT
END_VAR
VAR
    stControl : ST_CONTROL;
END_VAR
```

```
IF bStart = TRUE THEN
    // Symbolic bit access
    stControl.bitEnableOperation := TRUE;
END_IF
```

```
PROGRAM MAIN
VAR
    fbController : FB_Controller;
END_VAR
```

```
fbController();
fbController.bStart := TRUE;
```

Referenzen und Pointer auf BIT-Variablen sind ungültige Deklarationen, ebenso wie Arraykomponenten mit Basistyp BIT.

Siehe auch:

- BIT [▶ 795]
- ARRAY [▶ 809]

Sehen Sie dazu auch

- EQ [▶ 766]
- ARRAY [▶ 809]

16.5.18 Aufzählungen / Enumerationen

Eine Enumeration oder Aufzählung ist ein benutzerdefinierter Datentyp, der sich aus einer kommaseparierten Reihe von Komponenten, auch Enumerationswerte genannt, zusammensetzt, um benutzerdefinierte Variablen zu deklarieren. Außerdem können Sie die Enumerationskomponenten wie konstante Variablen verwenden, deren Bezeichner `<enumeration name>.<component name>` global im Projekt bekannt sind.

Die Deklaration einer Enumeration nehmen Sie in einem DUT-Objekt vor, das Sie über den Befehl **Hinzufügen > DUT** im Kontextmenü des SPS-Projektbaums im Projekt anlegen.

Jede Enumeration, die Sie einem Projekt neu hinzufügen, erhält automatisch ein Attribut 'strict' [▶ 860] und ein Attribut 'qualified only' [▶ 859] in der Zeile oberhalb der TYPE-Deklaration. Die Attribute können auch explizit hinzugefügt oder entfernt werden.

Siehe auch:

- Objekt DUT [▶ 77]
- Attribut 'to_string' [▶ 874]
- TO_STRING/TO_WSTRING für Enumerationsvariablen [▶ 762]

Deklaration

Syntax

```
{attribute 'strict'}
TYPE <enumeration name>:
(
    <component declaration>,
    <component declaration>
) <basic data type> := <default variable initialization>
;
END_TYPE
```

{attribute 'strict'}	Optional Das Pragma bewirkt, dass eine strenge Typprüfung, so wie unten beschrieben, durchgeführt wird Das Pragma ist optional, wird aber empfohlen.
<enumeration name>	Name der Enumeration, der im Code als Datentyp verwendet werden kann Beispiel: E_ColorBasic
<component declaration>	Beliebig viele, aber mindestens zwei Komponenten Die Werte der Komponenten werden automatisch initialisiert: Beginnend bei 0 werden die Werte fortlaufend um 1 inkrementiert. Sie können den einzelnen Komponenten aber auch explizit feste Initialwerte zuweisen. Beispiel: eYellow := 1
<basic data type>	Optional Sie können explizit einen der folgenden Basisdatentypen deklarieren: UINT | SINT | USINT | DINT | UDINT | LINT | ULINT | BYTE | WORD | DWORD | LWORD Falls kein expliziter Basisdatentyp deklariert ist, wird automatisch der Basisdatentyp INT verwendet.
<default variable initialization>	Optional Eine der Komponenten kann explizit als initiale Komponente deklariert werden. Falls keine explizite Initialisierung angegeben ist, wird automatisch mit der obersten Komponente initialisiert.

Beispiel

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_ColorBasic :
(
    eRed,
    eYellow,
    eGreen := 10,
    eBlue,
    eBlack
) // Basic data type is INT, default initialization for all E_ColorBasic  variables is eYellow
;
END_TYPE
```

In dieser Deklaration erhalten die ersten beiden Komponenten die Standard-Initialisierungswerte: eRed = 0, eYellow = 1, der Initialisierungswert der dritten Komponente ist explizit anders definiert: eGreen = 10. Die nachfolgenden Komponenten erhalten wieder Standard-Initialisierungswerte: eBlue = 11, eBlack = 12.

Enumeration mit explizitem Basisdatentyp

Erweiterungen zur Norm IEC 61131-3

Der Basisdatentyp bei einer Enumerationsdeklaration ist standardmäßig INT. Sie können aber auch Enumerationen deklarieren, die explizit auf einen anderen ganzzahligen Datentypen basieren.

Beispiel: Enumeration mit Basisdatentyp DWORD

```
TYPE E_Color :
(
    eWhite  := 16#FFFFFF,
    eYellow := 16#FFFF00,
    eGreen  := 16#FF00FF,
    eBlue   := 16#0000FF,
```

```
    eBlack  := 16#000000
)DWORD := eBlack
; // Basic data type is DWORD, default initialization for all E_Color variables is eBlack
END_TYPE
```

Strikte Programmierregeln

Ab TwinCAT 3.1 Build 4026 wird beim Deklarieren einer Enumeration automatisch in der ersten Zeile das Pragma `{attribute 'strict'}` hinzugefügt.

Die strikten Programmierregeln werden mit Hinzufügen des Pragmas `{attribute 'strict'}` aktiviert.

Folgender Code wird dann als Compilerfehler eingestuft:

- Arithmetische Operationen mit Enumerationskomponenten

  Beispielsweise kann eine Enumerationsvariable nicht als Zählvariable in einer FOR-Schleife verwendet werden.
- Zuweisung eines konstanten Werts, der nicht einem Enumerationswert entspricht, an eine Enumerationskomponente.
- Zuweisung einer nicht-konstanten Variablen, die einen anderen Datentyp als die Enumeration hat, an eine Enumerationskomponente.

Arithmetischen Operationen können dazu führen, dass Enumerationskomponenten nicht deklarierte Werte zugewiesen werden. Ein besserer Programmierstil ist es, CASE-Anweisungen zu verwenden, um Komponentenwerte zu bearbeiten.

Deklaration und Initialisierung von Enumerationsvariablen

Syntax

```
<variable name> : <enumeration name> ( := <initialization>)? ;
```

Bei der Deklaration einer Enumerationsvariable mit benutzerdefiniertem Datentyp `<enumeration name>` kann diese mit einer Enumerationskomponenten initialisiert werden.

Beispiel

```
PROGRAM MAIN
VAR
    eColorCar  : E_Color;
    eColorTaxi : E_Color := E_Color.eYellow;
END_VAR
```

Die Variable eColorCar wird mit E_Color.eBlack initialisiert. Das ist die Standardinitialisierung für alle Enumerationsvariablen des Typs E_Color und so in der Typdeklaration festgelegt. Die Variable eColorTaxi hat eine eigene Initialisierung.

Wenn keine Initialisierungen angegeben sind, wird mit 0 initialisiert.

```
PROGRAM MAIN
VAR
    eColorFlower : E_ColorBasic;
    eColorTree   : E_ColorBasic := E_ColorBasic.eGreen;
END_VAR
```

Die Variable eColorFlower wird mit E_ColorBasic.eYellow initialisiert. Das ist die Standardinitialisierung für alle Enumerationsvariablen des Typs E_ColorBasic. Da in der Enumerationsdeklaration keine Komponente für die Initialisierung angegeben ist, wird automatisch mit der Komponente, die den Wert 0 hat, initialisiert. Das ist üblicherweise die erste der Enumerationskomponenten. Das kann aber auch eine andere Komponente sein, die nicht an erster Stelle steht, aber explizit mit 0 initialisiert wird.

Die Variable eColorTree hat eine explizite Initialisierung.

Wenn sowohl beim Typ als auch bei der Variable kein Wert angegeben ist, dann gilt folgende Regel: Wenn eine Enumeration einen Wert für 0 enthält, dann ist dieser Wert die Standardinitialisierung, wenn nicht, dann die erste Komponente in der Liste.

Beispiele

Initialisierung mit der 0-Komponente:

```
TYPE E_SampleA :
(
    e1 := 2,
    e2 := 0,
    e3
);
END_TYPE
```

```
PROGRAM MAIN
VAR
    eSampleA  : E_SampleA;
END_VAR
```

Die Variable eSampleA wird mit E_SampleA.e2 initialisiert.

Initialisierung mit der ersten Komponente:

```
TYPE E_SampleB :
(
    e1 := 3,
    e2 := 1,
    e3
);
END_TYPE
```

```
PROGRAM MAIN
VAR
    eSampleB  : E_SampleB;
END_VAR
```

Die Variable eSampleB wird mit E_SampleB.e1 initialisiert.

Eindeutiger Zugriff auf Enumerationskomponenten

Erweiterungen zur Norm IEC 61131-3

Die Enumerationskomponenten können auch als konstante Variablen mit dem Bezeichner `<enumeration name>.<component name>` verwendet werden. Enumerationskomponenten sind global im Projekt bekannt und der Zugriff auf sie ist eindeutig. Deshalb kann ein Komponentenname in unterschiedlichen Enumerationen verwendet werden.

Beispiel: Komponente blue

```
PROGRAM MAIN
VAR
    eFlower   : E_ColorBasic;
    eColorCar : E_Color;
END_VAR
```

```
// unambiguous identifiers although the component names are identical
eFlower   := E_ColorBasic.eBlue;
eColorCar := E_Color.eBlue;

// invalid code
eFlower   := eBlue;
eColorCar := eBlue;
```

16.5.19 Alias

Ein Alias ist ein benutzerdefinierter Datentyp, mit dem ein alternativer Name für einen Datentyp oder einen Funktionsblock erzeugt werden kann.

Die Deklaration eines Alias nehmen Sie in einem DUT-Objekt vor, das Sie über den Befehl **Hinzufügen > DUT** im Kontextmenü des SPS-Projektbaums im Projekt anlegen.

Syntax:

TYPE <Bezeichner> : <Zuweisungsausdruck>;
END_TYPE

Beispiel:

Das Beispiel zeigt die Deklaration eines Alias T_Message. Eine im Programm deklarierte SPS-Variable vom Typ T_Message ist immer ein String mit 50 Zeichen.

```
TYPE T_Message : STRING[50];
END_TYPE
```

Deklaration:

```
sMessageA : T_Message;
```

Programm:

```
sMessageA := 'This is a message';
```

Siehe auch:

- Objekt DUT [▶ 77]

16.5.20 UNION

Eine UNION ist eine Datenstruktur, die meist unterschiedliche Datentypen enthält. In einer Union haben alle Komponenten den gleichen Offset, wodurch sie denselben Speicherplatz belegen.

Die Deklaration einer Union nehmen Sie in einem DUT-Objekt vor, das Sie über den Befehl **Hinzufügen > DUT** im Kontextmenü des SPS-Projektbaums im Projekt anlegen.

Syntax:

TYPE <Unionname>:
UNION
  <Variablendeklaration 1>
  ...
  <Variablendeklaration n>
END_UNION
END_TYPE

Beispiel 1:

In der folgenden Beispieldeklaration einer Union betrifft eine Zuweisung auf uName.fA auch uName.nB und uName.nC.

Deklaration:

```
TYPE U_Name:
UNION
    fA : LREAL;
    nB : LINT;
    nC : WORD;
END_UNION
END_TYPE
```

Instanziierung:

```
VAR
    uName : U_Name;
END_VAR
```

Zuweisung:

```
uName.fA := 1;
```

Ergebnis:

```
fA = 1
nB = 16#3FF0000000000000
nC = 0
```

Beispiel 2:

In der folgenden Beispieldeklaration einer Union betrifft eine Zuweisung auf u2Byte.nUINT auch u2Byte.a2Byte und u2Byte.aBits.

Deklaration der Struktur ST_Bits:

```
TYPE ST_Bits :
STRUCT
    bBit1  : BIT;
    bBit2  : BIT;
    bBit3  : BIT;
    bBit4  : BIT;
    bBit5  : BIT;
    bBit6  : BIT;
    bBit7  : BIT;
    bBit8  : BIT;
END_STRUCT
END_TYPE
```

Deklaration der Union U_2Byte:

```
TYPE U_2Byte :
UNION
    nUINT  : UINT;
    a2Byte : ARRAY[1..2] OF BYTE;
    aBits  : ARRAY[1..2] OF ST_Bits;
END_UNION
END_TYPE
```

Instanziierung der Union:

```
VAR
    u2Byte : U_2Byte;
END_VAR
```

Zuweisung 1:

```
u2Byte.nUINT := 5;
```

Ergebnis 1:

Expression	Type	Value
u2Byte	U_2Byte	
nUINT	UINT	5
a2Byte	ARRAY [1..2] OF BYTE	
a2Byte[1]	BYTE	5
a2Byte[2]	BYTE	0
aBits	ARRAY [1..2] OF ST_Bits	
aBits[1]	ST_Bits	
bBit1	BIT	TRUE
bBit2	BIT	FALSE
bBit3	BIT	TRUE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE
aBits[2]	ST_Bits	
bBit1	BIT	FALSE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE

Zuweisung 2:

```
u2Byte.nUINT := 255;
```

Ergebnis 2:

Expression	Type	Value
u2Byte	U_2Byte	
nUINT	UINT	255
a2Byte	ARRAY [1..2] OF BYTE	
a2Byte[1]	BYTE	255
a2Byte[2]	BYTE	0
aBits	ARRAY [1..2] OF ST_Bits	
aBits[1]	ST_Bits	
bBit1	BIT	TRUE
bBit2	BIT	TRUE
bBit3	BIT	TRUE
bBit4	BIT	TRUE
bBit5	BIT	TRUE
bBit6	BIT	TRUE
bBit7	BIT	TRUE
bBit8	BIT	TRUE
aBits[2]	ST_Bits	
bBit1	BIT	FALSE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE

Zuweisung 3:

```
u2Byte.nUINT := 256;
```

Ergebnis 3:

Expression	Type	Value
u2Byte	U_2Byte	
nUINT	UINT	256
a2Byte	ARRAY [1..2] OF BYTE	
a2Byte[1]	BYTE	0
a2Byte[2]	BYTE	1
aBits	ARRAY [1..2] OF ST_Bits	
aBits[1]	ST_Bits	
bBit1	BIT	FALSE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE
aBits[2]	ST_Bits	
bBit1	BIT	TRUE
bBit2	BIT	FALSE
bBit3	BIT	FALSE
bBit4	BIT	FALSE
bBit5	BIT	FALSE
bBit6	BIT	FALSE
bBit7	BIT	FALSE
bBit8	BIT	FALSE

Siehe auch:

- Objekt DUT [▶ 77]

16.6 Globale Datentypen

16.6.1 Übersicht

TwinCAT 3 stellt ein Typsystem für die Verwaltung von Datentypen bereit. Das Typsystem besteht aus System-Basistypen und kann durch das Kundenprojekt um eigene Datentypen erweitert werden. (Siehe auch TwinCAT 3 Typsystem)

In diesem Abschnitt finden Sie eine Beschreibung globaler Datentypen, die vom TwinCAT-System für die PLC bereitgestellt werden:

- PlcAppSystemInfo [▶ 823]
- PlcTaskSystemInfo [▶ 825]
- ST_LibVersion [▶ 826]

16.6.2 PlcAppSystemInfo

Jede SPS beinhaltet eine Instanz des Typs 'PlcAppSystemInfo' mit dem Namen '_AppInfo'.

Der zugehörige Namensraum (namespace) ist 'TwinCAT_SystemInfoVarList'. Dieser muss beispielsweise bei Verwendung in einer Bibliothek mit angegeben werden.

```
TYPE PlcAppSystemInfo
STRUCT
    ObjId               : OTCID;
```

```
    TaskCnt            : UDINT;
    OnlineChangeCnt    : UDINT;
    Flags              : DWORD;
    AdsPort            : UINT;
    BootDataLoaded     : BOOL;
    OldBootData        : BOOL;
    AppTimestamp       : DT;
    KeepOutputsOnBP    : BOOL;
    ShutdownInProgress : BOOL;
    LicensesPending    : BOOL;
    BSODOccured        : BOOL;
    LoggedIn           : BOOL;
    PersistentStatus   : EPlcPersistentStatus;

    TComSrvPtr         : ITComObjectServer;

    AppName            : STRING(63);
    ProjectName        : STRING(63);
END_STRUCT
END_TYPE
```

ObjId	Objekt-ID der SPS-Projektinstanz
TaskCnt	Anzahl der im Laufzeitsystem befindlichen Tasks
OnlineChangeCnt	Anzahl der seit dem letzten Komplettdownload gemachten Online-Änderungen
Flags	Reserviert
AdsPort	ADS-Port der SPS-Applikation
BootDataLoaded	PERSISTENT Variablen: LOADED (fehlerfrei geladen)
OldBootData	PERSISTENT Variablen: INVALID (es wurde die Sicherungskopie geladen, weil keine gültige Datei vorhanden war)
AppTimestamp	Zeitpunkt, zu dem die SPS-Applikation übersetzt wurde
KeepOutputsOnBP	Das Flag kann gesetzt werden und verhindert, dass die Ausgänge genullt werden, wenn ein Breakpoint erreicht wird. Die Task läuft in dem Fall weiter. Allein die Ausführung des SPS Code ist unterbrochen.
ShutdownInProgress	Diese Variable hat den Wert TRUE, falls das TwinCAT-System aktuell heruntergefahren wird. Manche Teile des TwinCAT-Systems sind ggf. schon heruntergefahren worden.
LicensesPending	Diese Variable hat den Wert TRUE, falls noch nicht alle Lizenzen, die von Lizenz-Dongles zur Verfügung gestellt werden, auf Gültigkeit geprüft wurden.
BSODOccured	Diese Variable hat den Wert TRUE, falls sich Windows in einem BSOD befindet.
LoggedIn	Diese Variable hat den Wert TRUE, falls mindestens eine XAE Instanz in das Projekt eingeloggt ist.
PersistentStatus	PERSISTENT Variablen: Status der Wiederherstellung: PS_None: Keine persistenten Variablen wurden wiederhergestellt. PS_All: Alle Variablen, die im aktuellen Projektstand persistent sind, wurden wiederhergestellt. PS_Partial: Es wurden weniger Variablen wiederhergestellt, als im aktuellen Projektstand persistent sind.
TComSrvPtr	Pointer auf den TcCOM Object Server
AppName	Von TwinCAT generierter Name, welcher den Port beinhaltet.
ProjectName	Name des Projekts

Unterschiede zu TwinCAT 2

Wurde die Variable runTimeNo unter TwinCAT 2 verwendet, so muss der Programmcode für die Verwendung unter TwinCAT 3 umgestellt werden.

Beispiel:

- **Verwendung unter TwinCAT 2**: nPlcAdsPort := 801 + (SystemInfo.runTimeNo - 1) * 10;
- **Verwendung unter TwinCAT 3**: nPlcAdsPort := _AppInfo.AdsPort;

16.6.3 PlcTaskSystemInfo

Jede SPS beinhaltet ein Array von Instanzen dieses Typs. Der Name des Arrays lautet '_TaskInfo[]'. Auf die einzelnen Instanzen dieses Typs kann zugegriffen werden, indem der Index der entsprechenden Task als Arrayindex verwendet wird. Der Taskindex kann mithilfe der Funktion GETCURTASKINDEXEX ausgelesen werden, welche von der Tc2_System-Bibliothek zur Verfügung gestellt wird.

Der zugehörige Namensraum (namespace) ist 'TwinCAT_SystemInfoVarList'. Dieser muss beispielsweise bei Verwendung in einer Bibliothek mit angegeben werden.

```
{attribute 'Namespace' := 'PLC'}
TYPE PlcTaskSystemInfo
STRUCT
     ObjId                   : OTCID;
     CycleTime               : UDINT;
     Priority                : UINT;
     AdsPort                 : UINT;
     CycleCount              : UDINT;
     DcTaskTime              : LINT;
     LastExecTime            : UDINT;
     FirstCycle              : BOOL;
     CycleTimeExceeded       : BOOL;
     InCallAfterOutputUpdate : BOOL;
     RTViolation             : BOOL;

     TaskName                : STRING(63);
END_STRUCT
END_TYPE
```

ObjId	Objekt-ID der Taskreferenz, von welcher das SPS-Programm aufgerufen wird.
CycleTime	Eingestellte Taskzykluszeit in Vielfachen von 100ns
Priority	Eingestellte Priorität der Task
AdsPort	ADS-Port der Task
CycleCount	Zykluszähler Bitte beachten Sie, dass sich der Zykluszähler auf die tatsächliche System-Task bezieht und nicht auf die Taskreferenz des SPS-Projekts. Hintergrund ist, dass sich mehrere SPS-Projekte oder andere TcCOM-Module eine Task teilen können und diese Module somit auf den gleichen Zykluszähler zugreifen. Daraus ergibt sich, dass der Zykluszähler bei einem TwinCAT-Neustart im Run-Modus zurückgesetzt wird (und nicht bei einem Reset des SPS-Projekts).
DcTaskTime	Distributed Clock System Time. Sie bleibt für eine Tasklaufzeit konstant.
LastExecTime	Benötigte Zykluszeit für den letzten Zyklus in Vielfachen von 100 ns
FirstCycle	Diese Variable hat im ersten Zyklus der SPS-Task den Wert TRUE.
CycleTimeExceeded	In dieser Variablen wird ein Überschreiten der eingestellten Taskzykluszeit gemeldet. In jedem Zyklus wird der Wert der Variable CycleTimeExceeded aktualisiert. Ist der vorherige Zyklus in seinem vorgesehenen Zeitfenster fertig geworden, wird die Variable auf 0 gesetzt, andernfalls auf 1.
InCallAfterOutputUpdate	Diese Variable hat den Wert TRUE, wenn der Ursprung des aktuellen Aufrufs mit dem Attribut 'TcCallAfterOutputUpdate' deklariert ist.
RTViolation	Diese Variable hat den Wert TRUE, wenn das Echtzeitlimit auf einem gemischten Kern (Windows + Echtzeit auf einem Core) überschritten wird. Der Wert TRUE bezieht sich dann auf den Kern, auf dem die entsprechende Task läuft.
TaskName	Name des Taskreferenz-Objekts im SPS-Projekt (z.B. 'MyPlcProject_PlcTaskRef')

Beispiel:

```
VAR
    nTaskIdx     : DINT;
    nCycleTime   : UDINT;
END_VAR
```

```
nTaskIdx := GETCURTASKINDEXEX();
IF nTaskIdx > 0 THEN
    nCycleTime := _TaskInfo[nTaskIdx].CycleTime;
END_IF
```

16.6.4 ST_LibVersion

Diese Struktur definiert die Version einer SPS-Bibliothek. Jede Bibliothek enthält eine global deklarierte Instanz dieses Datentyps.

```
TYPE ST_LibVersion :
STRUCT
    iMajor     : UINT;
    iMinor     : UINT;
    iBuild     : UINT;
    iRevision  : UINT;
    nFlags     : DWORD
    sVersion   : STRING(23);
END_STRUCT
END_TYPE
```

iMajor	Major Nummer
iMinor	Minor Nummer
iBuild	Build Nummer
iRevision	Revision Nummer
nFlags	Bibliotheksfreigabe nFlags = 1, die Bibliothek ist freigegeben (Häkchen gesetzt), nFlags = 0, die Bibliothek ist nicht freigegeben.
sVersion	Komplette Version als String

16.7 Alignment

Mit TwinCAT 3 wurde ein 8-Byte-Alignment eingeführt.

System	Alignment
TwinCAT 2, x86	1 Byte
TwinCAT 2, ARM	4 Byte
TwinCAT 3	8 Byte

Besondere Beachtung ist geboten, wenn Daten als gesamter Speicherblock mit anderen Steuerungen oder Softwarekomponenten wie Visualisierungen ausgetauscht werden.

Speicherstelle

- Eine Variable liegt an der Speicherstelle, die von der Größe ihres Datentyps und vom Alignment vorgegeben wird.
  - Wenn der Datentyp kleiner gleich dem Alignment des Systems ist, entspricht die Speicherstelle der Variablen einem Vielfachen der Datentypgröße.
  - Wenn der Datentyp größer als das Alignment des Systems ist, entspricht die Speicherstelle der Variablen einem Vielfachen des Alignments.
- Ein strukturierter Datentyp (Struktur, Funktionsbaustein) liegt an der Speicherstelle, die von der Größe des größten Datentyps in der Struktur und vom Alignment vorgegeben wird.
  - Wenn der größte Datentyp kleiner gleich dem Alignment des Systems ist, entspricht die Speicherstelle der Strukturinstanz einem Vielfachen dieser Datentypgröße.
  - Wenn der größte Datentyp größer als das Alignment des Systems ist, entspricht die Speicherstelle der Strukturinstanz einem Vielfachen des Alignments.

Speicherplatz

- Eine Variable nimmt ab obiger Speicherstelle so viel Speicherplatz ein, wie es der Größe ihres Datentyps entspricht.

- Ein strukturierter Datentyp (Struktur, Funktionsbaustein) nimmt ab obiger Speicherstelle so viel Speicherplatz ein, wie es von der Größe des größten Datentyps in der Struktur und dem Alignment vorgegeben wird.
  - Wenn der größte Datentyp kleiner gleich dem Alignment des Systems ist, entspricht der Speicherplatz der Strukturinstanz einem Vielfachen dieser Datentypgröße.
  - Wenn der größte Datentyp größer als das Alignment des Systems ist, entspricht der Speicherplatz der Strukturinstanz einem Vielfachen des Alignments.

Aufgrund dieser Alignment-Regeln können ggf. implizite Füllbytes entstehen.

Bitte beachten Sie (z.B. bei einem Speichervergleich mit Hilfe der Tc2_System-Funktion MEMCMP), dass Füllbytes nicht initialisiert werden.

Größter Datentyp in einem Funktionsbaustein

Jeder Funktionsbaustein enthält implizit mindestens einen Zeiger, um die Adresse der virtuellen Methodentabelle abzulegen. Daraus folgt, dass bei einem Zielsystem mit 64 Bit Architektur ein Datentyp von 8 Bytes Größe im Funktionsbaustein enthalten ist. Dabei kann es sich je nach Funktionsbaustein um den größten enthaltenen Datentypen handeln, der somit Speicherstelle und Speicherplatz bestimmt.

Beispiel 1 (TwinCAT 3)

```
TYPE ST_Test1 :
STRUCT
    fVar   : LREAL;  // 8 Byte
    nVar1  : DINT;   // 4 Byte
    nVar2  : SINT;   // 1 Byte
    (* 3 filler bytes to reach a limit corresponding to the largest data type (divisible by 8 byte)
*)
END_STRUCT
END_TYPE
```

```
TYPE ST_Test2 :
STRUCT
    nVar2  : SINT;   // 1 Byte
    (* 3 filler bytes to reach a limit corresponding to the following data type (divisible by 4 byte
) *)
    nVar1  : DINT;   // 4 Byte
    fVar   : LREAL;  // 8 Byte
END_STRUCT
END_TYPE
```

```
TYPE ST_Test3 :
STRUCT
    nVar2  : SINT;   // 1 Byte
    (* 7 filler bytes to reach a limit corresponding to the following data type (divisible by 8 byte
) *)
    fVar   : LREAL;  // 8 Byte
    nVar1  : DINT;   // 4 Byte
    (* 4 filler bytes to reach a limit corresponding to the largest data type (divisible by 8 byte)
*)
END_STRUCT
END_TYPE
```

Aufgrund des 8-Byte-Alignment von TwinCAT 3 werden implizit Füllbytes eingefügt. Die Gesamtgröße der Strukturen kann variieren, obwohl sie drei Variablen desselben Datentyps beinhalten.

Ausdruck	Datentyp	Wert
stTest1	ST_Test1	
stTest2	ST_Test2	
stTest3	ST_Test3	
nSize1	UDINT	16
nSize2	UDINT	16
nSize3	UDINT	24

```
1  nSize1  16  :=SIZEOF(stTest1);
2  nSize2  16  :=SIZEOF(stTest2);
3  nSize3  24  :=SIZEOF(stTest3);
4  RETURN
```

Auf einem System mit 1-Byte-Alignment würden die drei Strukturen denselben Speicherbedarf von jeweils 13 Bytes einnehmen.

Beispiel 2

```
TYPE ST_Test :
STRUCT
    nDWORD : DWORD;   // 4 Byte
    (* With an 8-byte alignment: 4 filler bytes *)
    nLWORD : LWORD;   // 8 Byte
END_STRUCT
END_TYPE
```

Bei einem 4-Byte-Alignment hat die Struktur eine Größe von 12 Byte. Es werden keine Füllbytes eingefügt. Begründung:

- Die Variable nLWORD liegt an einer Speicherstelle, die einem Vielfachen des Alignments entspricht, da ihr Datentyp (8 Byte) größer als das Alignment (4 Byte) ist.
- Der strukturierte Datentyp nimmt so viel Speicherplatz ein, wie es einem Vielfachen des Alignments entspricht, da der größte enthaltene Datentyp in der Struktur (8 Byte) größer als das Alignment (4 Byte) ist.

Bei einem 8-Byte-Alignment hat die Struktur hingegen eine Größe von 16 Byte, da zwischen den beiden Variablen 4 Füllbytes eingefügt werden. Begründung:

- Die Variable nLWORD liegt an einer Speicherstelle, die einem Vielfachen der Größe ihres Datentyps entspricht, da ihr Datentyp (8 Byte) kleiner gleich dem Alignment (8 Byte) ist.
- Der strukturierte Datentyp nimmt so viel Speicherplatz ein, wie es einem Vielfaches der Größe des größten Datentypen entspricht, da der größte enthaltene Datentyp in der Struktur (8 Byte) kleiner gleich dem Alignment (8 Byte) ist.

Weiterführende Informationen + Beispielprojekt:

- Sehen Sie auch das Attribut 'pack_mode' [▶ 853], um festzulegen, wie eine Datenstruktur gepackt werden soll.
- Auf der SPS-Samples-Seite [▶ 1182] können Sie außerdem ein Beispielprojekt zum Thema Byte-Alignment herunterladen.

16.8 Pragmas

Pragma-Anweisungen beeinflussen die Eigenschaften einer oder mehrerer Variablen bezüglich der Kompilierung oder des Prä-Kompilierungsprozesses. Dafür stehen Ihnen verschiedene Kategorien von Pragmas zur Verfügung.

16.8.1 Meldungspragmas

Meldungspragmas dienen dazu, die Ausgabe von Meldungen im Meldungsfenster während des Übersetzungsvorgangs zu erzwingen.

Die Pragma-Anweisung kann in einer separaten oder in einer bereits bestehenden Zeile im Texteditor einer POU eingefügt werden.

Typen

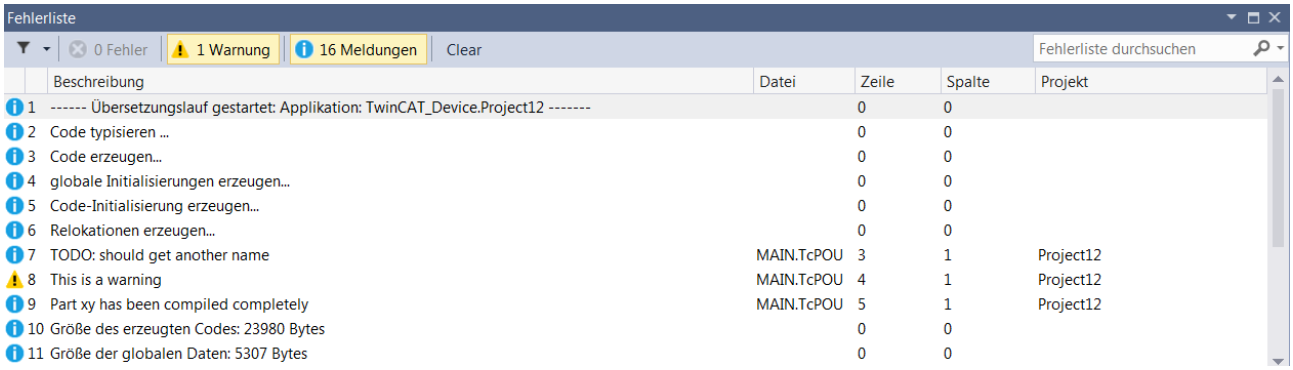
Pragma	Meldungstyp
{text <'textstring'>}	Text: Ausgabe von <textstring>.
{info <'textstring'>}	Information: Ausgabe von <textstring>.
{warning <'textstring'>}	Warnung: Ausgabe von <textstring>. Im Unterschied zum Attribut-Pragma 'obsolete' definieren Sie die Warnung lokal für die aktuelle Position.
{error <'textstring'>}	Fehler: Ausgabe von <textstring>.

Im Meldungsfenster von TwinCAT gelangen Sie von einer Meldung der Kategorie **Information**, **Warnung** und **Fehler** mithilfe der Befehle **Nächste Meldung** und **Vorherige Meldung** zur Quellposition der Meldung. Dies bedeutet, Sie gelangen an die Position, wo das Pragma im Quellcode hinzugefügt ist.

Beispiel:

```
VAR
    nVar : INT; {info 'TODO: should get another name'}
    bVar : BOOL;
    aTest : ARRAY [0..10] OF INT;
    nIdx : INT;
END_VAR

aTest[nIdx] := aTest[nIdx]+1;
nVar := nVar+1;

{warning 'This is a warning'}
{text 'Part xy has been compiled completely'}
```

Ausgabe im Meldungsfenster:



Siehe auch:

- Bedingte Pragmas [▶ 875]

16.8.2 Attribut-Pragmas

Attribut-Pragmas dienen dazu, die Kompilierung und die Vorkompilierung zu beeinflussen.

TwinCAT unterstützt eine Reihe von vordefinierten Attribut-Pragmas. Zusätzlich können Sie benutzerdefinierte Pragmas verwenden, die Sie vor der Übersetzung des Projekts mithilfe von bedingten Pragmas abfragen können.

Wenn Sie eigene Attribute definieren, achten Sie auf Eindeutigkeit. Sie erreichen dies beispielsweise, indem Sie den Attributnamen mit einem Präfix versehen.

Attribute werden im Deklarationsteil definiert. Ausnahme: In den Objekten **Aktion** und **Transition** können Sie Attribute nur dann verwenden, wenn die Implementierungssprache der Aktion oder Transition „Strukturierter Text (ST)“ ist. Da Aktionen und Transitionen keinen eigenen Deklarationsteil haben, definieren Sie die Attribute zu Beginn des Implementierungsteils.

16.8.2.1 Benutzerdefinierte Attribute

Benutzerdefinierte Attribute sind beliebige projekt- oder benutzerdefinierte Attribute, die Sie auf POUs, Aktionen, Datentypdefinitionen und Variablen anwenden können. Ein benutzerdefiniertes Attribut können Sie vor der Übersetzung des SPS-Projekts mithilfe von bedingten Pragmas abfragen.

Sie können benutzerdefinierte Attribute mit bedingten Pragmas mit dem Operator hasattribute abfragen.

Syntax: {attribute 'attribute'}

Beispiel für POUs und Aktionen:

Attribut 'vision' für Funktion F_Sample:

```
{attribute 'vision'}
FUNCTION F_Sample : INT
VAR_INPUT
    nSample : INT;
END_VAR
```

Beispiel zu Variablen:

Attribut 'DoCount' für Variable nVar:

```
PROGRAM MAIN
VAR
    {attribute 'DoCount'};
    nVar : INT;
    bVar : BOOL;
END_VAR
```

Beispiel zu Datentypen:

Attribut 'aType' für Datentyp ST_MyType:

```
{attribute 'aType'}
TYPE ST_MyType :
STRUCT
    nTest : INT;
    bTest : BOOL;
END_STRUCT
END_TYPE
```

Siehe auch:

- Bedingte Pragmas [▶ 875]

16.8.2.2 Attribut 'c++_compatible'

Das Pragma bewirkt, dass die VTable, die der SPS-Compiler erzeugt, binär kompatibel zu der eines C++-Compilers erzeugt wird. Damit ist es möglich, auch aus einem in C++ implementierten TcCom-Module auf die in der SPS implementieren Methoden einer Schnittstelle zuzugreifen.

Syntax: {attribute 'c++_compatible'}

Einfügeort:

Das Pragma muss an folgenden Stellen eingefügt werden:

- Zeile oberhalb der Deklaration der Schnittstelle
- Zeile oberhalb der Deklaration der einzelnen Methoden der Schnittstelle
- Zeile oberhalb der Deklaration des Funktionsbausteins, der die Schnittstelle implementiert
- Zeile oberhalb der Deklaration der Methoden, die aus der Schnittstelle implementiert werden

Beispiel:

Deklaration eines Funktionsbausteins, der eine C++-kompatible Schnittstelle implementiert:

```
{attribute 'c++_compatible'}
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```

Deklaration einer Methode, die in der Schnittstelle definiert ist:

```
{attribute 'c++_compatible'}
{attribute 'minimal_input_size' := '4'}
{attribute 'pack_mode' := '4'}
METHOD Method1 : HRESULT
VAR_INPUT
    nParameter1 : INT;
END_VAR
```

Siehe auch:

- Attribut 'minimal_input_size' [▶ 846]
- Attribut 'pack_mode' [▶ 853]

16.8.2.3 Attribut 'call_after_global_init_slot'

Das Pragma bewirkt, dass alle Funktionen und Programme, die dieses Attribut enthalten, nach der globalen Initialisierung aufgerufen werden. Die Aufrufreihenfolge legen Sie durch den Attributwert fest.

Syntax: {attribute 'call_after_global_init_slot' := '<slot>'}

<slot> : Ganzzahliger Wert, der den Stellenwert in der Reihenfolge der Aufrufe definiert: Je niedriger der Wert ist, desto früher erfolgt der Aufruf. Wenn mehrere Bausteine denselben Stellenwert für das Attribut haben, so bleibt die Reihenfolge ihrer Aufrufe unbestimmt.

Einfügeort: Erste Zeile über dem Deklarationsteil von Funktionen und Programmen

Wenn eine Methode das Attribut besitzt, ermittelt TwinCAT alle Instanzen des entsprechenden Funktionsbausteins und ruft alle Instanzen im angegebenen Slot auf. Auf die Reihenfolge der Instanzen untereinander haben Sie in diesem Fall keinen Einfluss.

Siehe auch:

- Methoden FB_init, FB_reinit und FB_exit [▶ 887]

16.8.2.4 Attribut 'call_after_init'

Das Pragma bewirkt, dass eine Methode implizit nach der Initialisierung einer Funktionsbausteininstanz aufgerufen wird. Genau genommen wird die Methode nach der Verarbeitung der initialen Zuweisungen und vor dem Start der Tasks eines SPS-Projekts aufgerufen und kann so auf die Vorgaben des Anwenders entsprechend reagieren. Der Name der Methode ist frei wählbar (Ausnahmen: FB_init, FB_reinit und FB_exit).

Aus Performanzgründen müssen Sie das Attribut sowohl dem Funktionsbaustein als auch der Methode in einer eigenen ersten Zeile über dem Deklarationsteil hinzufügen.

Syntax: {attribute 'call_after_init'}

Einfügeort: Erste Zeile über dem Deklarationsteil der Methode und des Funktionsbausteins

TwinCAT ruft die Methode nach der Methode FB_init auf und nachdem die Variablenwerte eines Initialisierungsausdrucks in der Instanzdeklaration gültig wurden.

call_after_init bei abgeleiteten Bausteinen

Ein Baustein, der einen anderen Baustein erweitert, der das Attribut 'call_after_init' verwendet, muss ebenfalls mit dem Attribut versehen werden.

Aus Gründen der Verständlichkeit wird empfohlen, die entsprechende Methode mit dem gleichen Namen, der gleichen Signatur und dem gleichen Attribut zu überschreiben. Das erfordert einen Aufruf von `SUPER^.MyInit`.

Methoden, die das Attribut 'call_after_init' enthalten, dürfen keine Eingänge (VAR_INPUT) haben. TwinCAT gibt einen entsprechenden Übersetzungsfehler aus.

Debugging

Das Finden von Fehlern in Methoden, die mit {attribute 'call_after_init'} deklariert sind, ist mühsam, weil unter anderem das Setzen von Haltepunkten nicht die gewünschte Wirkung haben kann.

Beispiel:

Definition:

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB_Sample
... <functionblock definition>
```

```
{attribute 'call_after_init'}
METHOD CallAfterInit
... <method definition>
```

Die Definition setzt zum Beispiel folgende Deklaration in die nachfolgende Codeabarbeitung um:

```
fbSample : FB_Sample := (nValue1 := 99);
```

Codeabarbeitung:

```
fbSample.FB_Init();
fbSample.nValue1 := 99;
fbSample.CallAfterInit();
```

Somit kann in CallAfterInit() auf die benutzerdefinierte Initialisierung reagiert werden.

Siehe auch:

- Methoden FB_init, FB_reinit und FB_exit [▶ 887]

16.8.2.5 Attribut 'call_after_online_change_slot'

Das Pragma bewirkt, dass alle Funktionen und Programme, die dieses Attribut enthalten, nach einem Online-Change aufgerufen werden. Die Aufrufreihenfolge legen Sie durch den Attributwert <slot> fest.

Syntax: {attribute 'call_after_online_change_slot' := '<slot>'}

<slot>: Ganzzahliger Wert, der den Stellenwert in der Reihenfolge der Aufrufe definiert: Je niedriger der Wert ist, desto früher erfolgt der Aufruf. Haben mehrere Bausteine denselben Stellenwert für das Attribut, so bleibt die Reihenfolge ihrer Aufrufe unbestimmt.

Einfügeort: Erste Zeile über dem Deklarationsteil von Funktionen und Programmen

Wenn eine Methode das Attribut besitzt, dann ermittelt TwinCAT alle Instanzen des betreffenden Funktionsbausteins. TwinCAT ruft alle Instanzen in dem angegebenen Slot auf. Auf die Reihenfolge der Instanzen untereinander haben Sie in diesem Fall keinen Einfluss.

Da das SPS-Programm während des Online-Change nicht laufen kann, kann jeder Code, der in dieser Situation ausgeführt wird, zu einem Jitter führen. Halten Sie den auszuführenden Code-Umfang deshalb möglichst gering.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Online-Change [▶ 1000]

16.8.2.6 Attribut 'call_on_type_change'

Mit dem Pragma kennzeichnen Sie eine Methode eines Funktionsbausteins A, die aufgerufen werden soll, sobald sich der Datentyp eines von A referenzierten Funktionsbausteins B, C, ... ändert. Die Referenzierung kann über eine Zeigervariable oder über eine Variable vom Typ REFERENCE definiert sein. Die von A referenzierten Funktionsbausteine, deren Typänderung den Methodenaufruf auslösen soll, legen Sie im Attributwert fest.

Syntax:

{attribute 'call_on_type_change':= '<Name des ersten referenzierten Funktionsbausteins>, <Name des zweiten referenzierten Funktionsbausteins>, <Name des ... referenzierten Funktionsbausteins>'}

Einfügeort: Zeile oberhalb der ersten Zeile in der Deklaration der Methode

Beispiel:

Funktionsbaustein mit Referenzen:

```
FUNCTION_BLOCK FB_A
...
VAR
    pVar   : POINTER TO FB_B;
    refVar : REFERENCE TO FB_C;
END_VAR
```

Methode zur Reaktion auf eine Typänderung in den Referenzen FB_B und FB_C:

```
{attribute 'call_on_type_change' := 'FB_B, FB_C'}
METHOD METH_react_on_type_change : INT
VAR_INPUT
```

16.8.2.7 Attribut 'conditionalshow'

Das Pragma ist relevant für Funktionsbausteine und weitere Typen, welche in einer Bibliothek enthalten sind.

Das Pragma bewirkt, dass der gesamte Funktionsbaustein, der dieses Attribut enthält, in der Benutzeroberfläche nicht angezeigt wird, wenn die verbundene Bibliothek als *.compiled-library installiert ist. Der Funktionsbaustein wird aber in der Benutzeroberfläche angezeigt, wenn die verbundene Bibliothek als *.library installiert ist.

Betroffene Features:

- Bibliotheksverwaltung
- Debugging
- Eingabehilfe
- Funktion Komponenten auflisten
- Monitoring
- Symbolkonfiguration

Das ist nützlich, wenn Sie Bibliotheken entwickeln. Als Bibliotheksentwickler zeichnen Sie Funktionsbausteine oder Variablen mit dem Pragma aus. Damit legen Sie fest, welche Bezeichner nach dem Einbinden in ein Projekt verborgen werden. Wenn Sie später die verborgenen Bezeichner beispielsweise beim Debugging oder beim Weiterentwickeln der Bibliothek vermissen, können Sie deren Sichtbarkeit wieder aktivieren.

Syntax: {attribute 'conditionalshow'}

Einfügeort: Zeile oberhalb einer Signatur

Beispiele

Verbergen einer Variablen:

```
FUNCTION_BLOCK FB_Sample
PROGRAM_MAIN
VAR
{attribute 'conditionalshow'}
    nLocal    : INT;
    nCounter  : INT;
END_VAR
```

Die Variable nLocal ist unsichtbar.

Verbergen eines Funktionsbausteins:

```
{attribute 'conditionalshow'}
FUNCTION_BLOCK FB_ Sample
VAR
    nLocal    : INT;
    nCounter  : INT;
END_VAR
```

Die Funktionsbaustein FB_Sample und seine Variablen nLocal und nCounter sind unsichtbar.

Siehe auch:

- Attribut 'conditionalshow_all_locals' [▶ 834]
- Attribut 'hide' [▶ 841]
- Attribut 'hide_all_locals' [▶ 842]

16.8.2.8 Attribut 'conditionalshow_all_locals'

Das Pragma ist relevant für Funktionsbausteine und weitere Typen, welche in einer Bibliothek enthalten sind.

Das Pragma bewirkt, dass alle lokalen Variablen in der Benutzeroberfläche nicht angezeigt werden, wenn die verbundene Bibliothek als *.compiled-library installiert ist. Die lokalen Variablen sind aber in der Benutzeroberfläche sichtbar, wenn die verbundene Bibliothek als *.library installiert ist.

Betroffene Features:

- Bibliotheksverwaltung
- Debugging
- Eingabehilfe
- Funktion **Komponenten auflisten**
- Monitoring
- Symbolkonfiguration

Das ist nützlich, wenn Sie Bibliotheken entwickeln. Als Bibliotheksentwickler zeichnen Sie Funktionsbausteine oder Variablen mit dem Pragma aus. Damit legen Sie fest, welche Bezeichner nach dem Einbinden in ein Projekt verborgen werden. Wenn Sie später die verborgenen Bezeichner beispielsweise beim Debugging oder beim Weiterentwickeln der Bibliothek vermissen, können Sie deren Sichtbarkeit wieder aktivieren.

Syntax: {attribute 'conditionalshow_all_locals'}

Einfügeort: Zeile oberhalb einer Signatur

Beispiel: Verbergen aller Variablen

```
{attribute 'conditionalshow_all_locals'}
FUNCTION_BLOCK FB_ Sample
VAR
    nLocal    : INT;
    nCounter  : INT;
END_VAR
```

Die lokalen Variablen nLocal und nCounter des Funktionsbausteins FB_Sample sind unsichtbar.

Siehe auch:

- Attribut 'conditionalshow' [▶ 833]
- Attribut 'hide' [▶ 841]
- Attribut 'hide_all_locals' [▶ 842]

16.8.2.9 Attribut 'const_replaced', Attribut 'const_non_replaced'

Das Attribut {attribute 'const_non_replaced'} bewirkt, dass die Konstante im Code ersetzt wird, unabhängig von der Einstellung der Compileroption **Konstanten ersetzen**. Das Attribut hat eine Auswirkung auf Variablen von skalaren Typen, aber nicht auf zusammengesetzte Typen wie Arrays und Strukturen.

Das Pragma {attribute 'const_non_replaced'} fügen Sie entsprechend ein, um die Compileroption Konstanten ersetzen explizit zu deaktivieren. Dies kann beispielsweise erwünscht sein, damit eine Konstante in der Symbolkonfiguration verfügbar ist.

Syntax:

{attribute 'const_replaced'}

{attribute 'const_non_replaced'}

Einfügeort: Zeile oberhalb der Deklarationszeile der globalen Variablen.

Beispiel:

Die Konstanten nTestCon und bTestCon sind in der Symbolkonfiguration verfügbar, weil die Option Konstanten ersetzen deaktiviert ist.

```
VAR_GLOBAL CONSTANT
    {attribute 'const_non_replaced'}
    nTestCon  :  INT  := 12;
    {attribute 'const_non_replaced'}
    bTestCon  :  BOOL := TRUE;
    fTestCon  :  REAL := 1.5;
END_VAR

VAR_GLOBAL
    nTestVar  :  INT  := 12;
    bTestVar  :  BOOL := TRUE;
END_VAR
```

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Eigenschaften (Projekt) > Kategorie Übersetzen [▶ 951]

16.8.2.10 Attribut 'dataflow'

Mit dem Pragma steuern Sie den Datenfluss in der Abarbeitung von Funktionsbausteinen im FUP/KOP/AWL-Editor. Das Attribut legt fest, an welchem Eingang oder Ausgang eines Funktionsbausteins die Weiterverbindung zum nächsten oder vorherigen Baustein anliegt.

Sie dürfen nur einen Eingang und einen Ausgang in der Deklaration eines Funktionsbausteins mit dem Attribut versehen.

Syntax: {attribute 'dataflow'}

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen.

Bei Funktionsbausteinen ohne Attribut 'dataflow' bestimmt TwinCAT den Datenfluss folgendermaßen: Zunächst wird die Verbindung zwischen einem Ausgang und einem Eingang gleichen Datentyps gelegt. Dabei wird zuerst immer die oberste Eingangs- oder Ausgangsvariable der Funktionsbausteine genommen. Wenn es keine Variablen mit übereinstimmendem Datentyp gibt, verbindet TwinCAT den obersten Ausgang mit dem obersten Eingang der benachbarten Bausteine.

Beispiel:

Die Verbindung zwischen FB und dem vorhergehenden Funktionsbaustein erfolgt über die Eingangsvariable i1. Die Verbindung zwischen FB und dem nachfolgenden Funktionsbaustein erfolgt über die Ausgangsvariable outRes1.

```
FUNCTION_BLOCK FB
VAR_INPUT
    r1 : REAL;
    {attribute 'dataflow'}
    i1 : INT;
    i2 : INT;
    r2 : REAL;
END_VAR
VAR_OUTPUT
    {attribute 'dataflow'}
    outRes1 : REAL;
    out1 : INT;
    g1 : INT;
    g2 : REAL;
END_VAR
```

Siehe auch:

- FUP/KOP/AWL [▶ 113]

16.8.2.11 Attribut 'displaymode'

Mit dem Pragma definieren Sie den Darstellungsmodus einer einzelnen Variablen. Diese Festlegung überschreibt die globale Einstellung für die Darstellung der Monitoring-Variablen, die über die Befehle im Menü **Debug > Darstellung** erfolgt.

Syntax: {attribute 'displaymode':=<displaymode>}

Folgende Definitionen sind möglich:

- Binärformat
  - {attribute 'displaymode':='bin'}
  - {attribute 'displaymode':='binary'}
- Dezimalformat
  - {attribute 'displaymode':='dec'}
  - {attribute 'displaymode':='decimal'}
- Hexadezimalformat
  - {attribute 'displaymode':='hex'}
  - {attribute 'displaymode':='hexadecimal'}

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

Beispiel:

```
VAR
    {attribute 'displaymode':='hex'}
    nVar1 : DWORD;
END_VAR
```

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Darstellung - Binär, Dezimal, Hexadezimal [▶ 1008]

16.8.2.12 Attribut 'enable_dynamic_creation'

Das Pragma wird benötigt, um den __NEW-Operator bei Funktionsbausteinen/DUTs zu verwenden.

Syntax: {attribute 'enable_dynamic_creation'}

Einfügeort: Erste Zeile über dem Deklarationsteil des Funktionsbausteins/DUTs

Siehe auch:

- Referenz Programmierung: __NEW [▶ 768]

16.8.2.13 Attribut 'estimated-stack-usage'

Das Pragma übergibt einen Schätzwert für den Stackgrößenbedarf.

Methoden mit rekursivem Aufruf halten einer Stackprüfung nicht stand, weil der Stackverbrauch nicht ermittelt werden kann. Folglich wird eine Warnung für diese Methoden ausgegeben. Um die Warnung zu unterdrücken, können Sie der Methode mit Hilfe des Attributs einen Schätzwert in Bytes für den Stackgrößenbedarf mitgeben. Dann durchläuft die Methode die Stackprüfung erfolgreich.

Syntax: {attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'}

Einfügeort: Erste Zeile über dem Deklarationsteil der Methode

Beispiel:

```
{attribute 'estimated-stack-usage' := '99'} // 99 bytes
METHOD SampleMethod : INT
VAR_INPUT
END_VAR
```

Ausgegebene Warnung:

Die Warnung, die für rekursive Methoden ohne Attribut 'estimated-stack-usage' ausgegeben wird, lautet:

„C0298: Berechnung des Stackverbrauchs aufgrund rekursiver Aufrufe unvollständig, beginnend bei '<FB.Methode>'“

In einem Projekt, das im Vergleich zum letzten Erstellen des Projekts unverändert ist, wird die Warnung nur bei Verwendung des Befehls Projekt neu erstellen ausgegeben (nicht bei Verwendung des Befehls Projekt erstellen).

Einstellung der Stackgröße:

Der Wert der maximal möglichen Stackgröße kann in den Real-Time Einstellungen des TwinCAT-Projekts konfiguriert werden (Registerkarte Settings > Global Task Config/Maximal Stack Size [KB]). Der eingestellte Maximalwert steht dabei nicht komplett dem SPS-Projekt zur Verfügung, sondern wird z. B. ein Teil davon vom TwinCAT-Laufzeitsystem gebraucht.

Keine Stackberechnung für Stand-alone SPS-Projekte

Aufgrund der Abkoppelung vom System Manager ist eine Berechnung des Stackverbrauchs für ein Stand-alone SPS-Projekt nicht möglich.

Rekursiver Methodenaufruf

Innerhalb der zugehörigen Implementierung kann sich eine Methode selbst aufrufen: entweder direkt mit Hilfe des THIS-Pointers oder mit Hilfe einer lokalen Variablen für den zugeordneten Funktionsbaustein.

⚠ WARNUNG

Maschinenstillstand durch möglichem Stacküberlauf

Verwenden Sie Rekursionen vorwiegend zur Bearbeitung von rekursiven Datentypen, wie beispielsweise verketteten Listen. Allgemein ist es ratsam, bei der Verwendung von Rekursion vorsichtig zu sein, da es bei unerwartet tiefen Rekursionen zu einem Stacküberlauf und damit zu einem Maschinenstillstand kommen kann.

Beispiel: Berechnung der Fakultät

Innerhalb des Funktionsbausteins `FB_Factorial` wird die Fakultät einer Zahl auf unterschiedliche Weise berechnet. Die verschiedenen Berechnungen finden jeweils in einer eigenen Methode statt.

- Methode `Iterative`: Iterativ
- Methode `Pragmaed`: Rekursiv mit Warnungsunterdrückung
- Methode `Recursive`: Rekursiv

Bei der Neuerstellung des Projekts erzeugt nur die Methode `Recursive` die Warnung C0298.

Struktur ST_FactorialResult zur Speicherung der Werte:

```
// Contains the data of the factorial calculation of nNumber.
TYPE ST_FactorialResult :
STRUCT
    nNumber    : USINT;
    nIterative : UDINT;
    nRecursive : UDINT;
    nPragmaed  : UDINT;
END_STRUCT
END_TYPE
```

Funktionsbaustein FB_Factorial zur Berechnung der Fakultät:

```
// Factorial calculation in different ways
FUNCTION_BLOCK FB_Factorial
VAR
     nNumberIterative    : UINT;
END_VAR
```

Property nNr samt Set-Funktion, zur Übergabe des Parameters für die iterative Methode:

```
{attribute 'monitoring' := 'variable'}
PROPERTY nNr : UINT
```

```
nNumberIterative := nNr;
```

Iterative Berechnungsmethode:

```
// Iterative calculation
METHOD PUBLIC Iterative : UDINT
VAR
    nCnt : UINT;
END_VAR
```

```
Iterative := 1;

IF nNumberIterative > 1 AND nNumberIterative <= 12 THEN
    FOR nCnt := 1 TO nNumberIterative DO
        Iterative := Iterative * nCnt;
    END_FOR;
    RETURN;
ELSE
    RETURN;
END_IF
```

Rekursive Berechnungsmethode mit Attribut zur Unterdrückung der Warnung C0298:

```
// Recursive calculation with suppressed warning
{attribute 'estimated-stack-usage' := '200'}
METHOD PUBLIC Pragmaed : UDINT
VAR_INPUT
    nNumber : USINT;
END_VAR
```

```
Pragmaed := 1;

IF nNumber > 1 AND nNumber <= 12 THEN
    Pragmaed := nNumber * THIS^.Pragmaed(nNumber := (nNumber - 1));
    RETURN;
ELSE
    RETURN;
END_IF
```

Der Pragma-Parameter zur Angabe des geschätzten Stackgrößenbedarfs wird beispielhaft auf 200 Byte gesetzt. Dem liegt die folgende Berechnung zugrunde:

Stackverbrauch pro Methodenaufruf:
Rückgabewert UDINT + Eingangsparameter USINT + Methodenpointer = 4 Byte + 1 Byte + 8 Byte = 13 Byte

Stackverbrauch bei maximal 12 Aufrufen:
12 * 13 Byte = 156 Byte, aufgerundet auf 200 Byte

Rekursive Berechnungsmethode:

```
// Recursive calculation
METHOD PUBLIC Recursive : UDINT
VAR_INPUT
    nNumber : USINT;
END_VAR
```

```
Recursive := 1;
IF nNumber > 1 AND nNumber <= 12 THEN
    Recursive := nNumber * THIS^.Recursive(nNumber := (nNumber - 1) );
    RETURN;
ELSE
    RETURN;
END_IF
```

Hauptprogramm MAIN:

```
PROGRAM MAIN
VAR
    fbFactorial : FB_Factorial;
    stFactorial : ST_FactorialResult := (nNumber := 9);
END_VAR
```

```
// Iterativ
fbFactorial.nNr        := stFactorial.nNumber;
stFactorial.nIterative := fbFactorial.Iterative();

// Recursive
stFactorial.nRecursive := fbFactorial.Recursive(nNumber := stFactorial.nNumber);

// Pragmaed
stFactorial.nPragmaed  := fbFactorial.Pragmaed(nNumber := stFactorial.nNumber);
```

16.8.2.14 Attribut 'ExpandFully'

Das Pragma bewirkt, dass die Komponenten eines Arrays, das als Eingabevariable für referenzierte Visualisierungen verwendet wird, im Eigenschaftendialog der Visualisierung sichtbar gemacht werden.

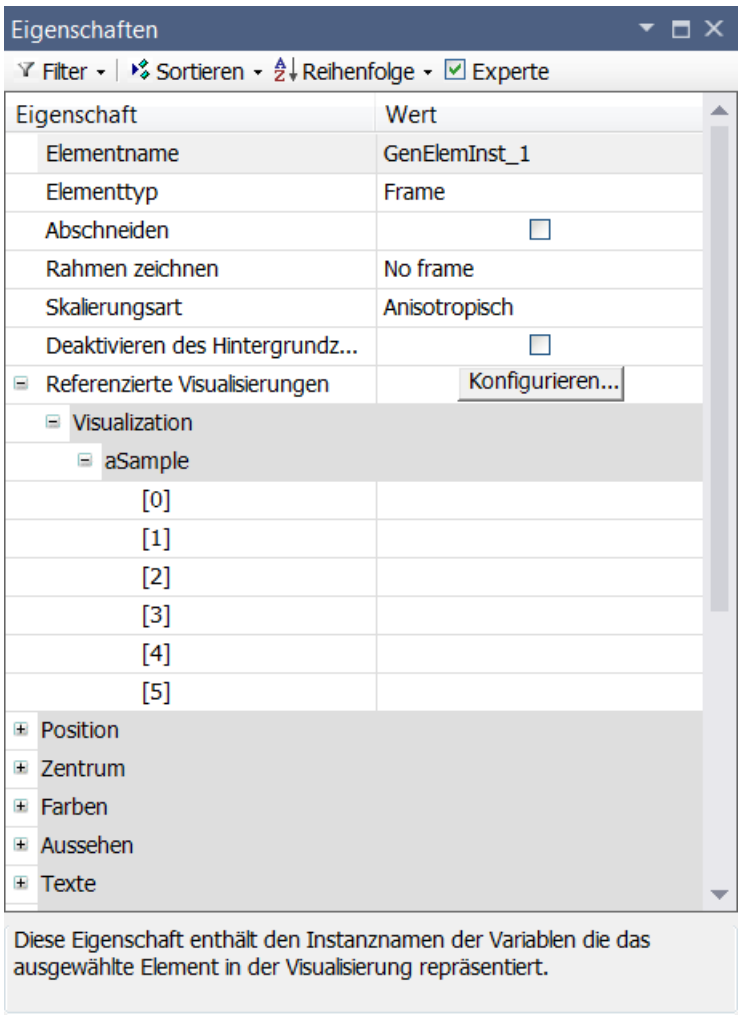
Syntax: {attribute 'ExpandFully'}

Einfügeort: Zeile oberhalb der Deklarationszeile des Arrays

Beispiel:

Visualisierung visu soll in einen Frame innerhalb von Visualisierung visu_main eingefügt werden. aSample ist als Eingabevariable im Schnittstelleneditor von visu definiert und wird somit später für Zuweisungen im Eigenschaftendialog des Frames in visu_main bereitstehen. Um auch die einzelnen Komponenten von aSample in diesem Eigenschaftendialog verfügbar zu bekommen, müssen Sie das Attribut 'ExpandFully' im Schnittstelleneditor von visu direkt vor aSample einfügen. Deklaration im Schnittstelleneditor von visu:

```
VAR_INPUT
    {attribute 'ExpandFully'}
    aSample : ARRAY[0..5] OF INT;
END_VAR
```



16.8.2.15 Attribut 'global_init_slot'

Das Pragma definiert die Reihenfolge der Initialisierung von Programmierbausteinen und globalen Variablenlisten. Variablen innerhalb einer GVL oder POU werden von oben nach unten initialisiert. Im Fall von mehreren globalen Variablenlisten ist deren Initialisierungsreihenfolge unbestimmt.

Bei Initialisierungen mit Literalwerten, beispielsweise 1, 'hallo', 3.6, oder Konstanten von Basisdatentypen spielt die Reihenfolge der Initialisierungen keine Rolle. Wenn es bei den Initialisierungen jedoch Abhängigkeiten zwischen den Listen gibt, müssen Sie die Initialisierungsreihenfolge festlegen. Dazu können Sie einer GVL oder einer POU mit dem Attribut 'global_init_slot' einen definierten Initialisierungsslot zuweisen.

Konstanten werden vor den Variablen initialisiert, und zwar in der gleichen Reihenfolge wie die Variablen. Bei der Initialisierung werden die Signaturen (Bausteine, GVLs) zunächst nach dem Wert für <slot> sortiert. Anschließend wird der Code für die Initialisierung der Konstanten erzeugt und danach der Code für die Initialisierung der Variablen.

Die Initialisierung teilt sich somit in die folgenden Phasen ein:

1. Die Signaturen werden nach den Initialisierungsslots sortiert. Dabei sind die Slots entweder implizit definiert oder explizit über das Attribut 'global_init_slot'.
2. Anschließend werden alle Konstanten initialisiert. Dies geschieht in der Reihenfolge der Slots. Signaturen ohne Konstanten werden übersprungen.
3. Dann werden die Variablen initialisiert, wieder in der Reihenfolge der Slots.

Syntax: {attribute 'global_init_slot' := '<slot>'}

<slot>: Ganzzahliger Wert, der die Position in der Reihenfolge der Aufrufe definiert. Der Standardwert für eine POU (Programm, Funktionsbaustein) ist 50000. Der Standardwert für eine GVL ist 49990. Der Standardwert für statische Variablen ist 49980. Ein niedrigerer Wert bewirkt eine frühere Initialisierung.

Einfügeort: Das Pragma wirkt immer auf die gesamte GVL oder POU und muss deshalb oberhalb der VAR_GLOBAL-Deklaration oder der POU-Deklaration stehen.

Wenn mehrere Programmierbausteine denselben Wert für das Attribut 'global_init_slot' zugewiesen bekommen haben, bleibt die Reihenfolge ihrer Initialisierung unbestimmt. Um das Systemverhalten von TwinCAT 3 nicht zu beeinflussen, verwenden Sie Werte oberhalb von 40000.

Beispiel:

Das Projekt enthält zwei globale Variablenlisten GVL1 und GVL2. Das Programm MAIN verwendet Variablen aus beiden Listen. GVL1 verwendet zur Initialisierung einer Variablen nA die Variable nB, die in GVL2 mit einem Wert von 1000 initialisiert wird:

GVL1:

```
VAR_GLOBAL   //49990
    nA : INT := GVL2.nB*3;
END_VAR
```

GVL2:

```
VAR_GLOBAL   //49990
    nB : INT := 1000;
    nC : INT := 10;
END_VAR
```

Programm MAIN:

```
PROGRAM MAIN  //50000
VAR
    nVar1 : INT := GVL1.nA;
    nVar2 : INT;
END_VAR
```

```
nVar1 := nVar + 1;
nVar2 := GVL2.nC;
```

In diesem Fall gibt der Compiler einen Fehler aus, weil GVL2.nB zur Initialisierung von GVL1.nA verwendet wird, bevor GVL2 initialisiert wurde. Dies können Sie verhindern, indem Sie mit dem Attribut global_init_slot die Position von GVL2 in der Initialisierungsreihenfolge vor GVL1 stellen.

Dazu muss der global_init_slot-Wert von GVL2 kleiner als der Wert von GVL1 (mit dem Standardwert 49990) und größer als 40000 (für Systemfunktionen reserviert) sein.
D. h.: 40001 <= global_init_slot-Wert von GVL2 <= 49989.

GVL2:

```
{attribute 'global_init_slot' := '40500'}
VAR_GLOBAL
    nB : INT := 1000;
    nC : INT := 10;
END_VAR
```

Die Verwendung von GVL2.nC im Implementierungsteil von MAIN ist auch ohne Pragmaverwendung unkritisch, da beide GVLs in jedem Fall vor dem Programm initialisiert werden.

16.8.2.16 Attribut 'hide'

Das Pragma verhindert, dass Variablen oder Programmelemente in der TwinCAT-Oberfläche angezeigt werden. Sie sind dann beispielsweise in der Eingabehilfe oder im Deklarationsteil im Onlinebetrieb nicht sichtbar, sie können nicht über die Querverweissuche gefunden werden und es können keine Debug-Funktionalitäten (wie z. B. Steppen oder Haltepunkte) darauf angewendet werden.

Beachten Sie, dass Variablen, die mit dem Attribut 'hide' oder 'hide_all_locals' deklariert sind, nicht als persistent abgespeichert werden können. Des Weiteren wird für „versteckte“ Variablen die Erzeugung des zugehörigen Prozessabbilds (allokierte Inputs/Outputs) verhindert. Zudem werden für diese Variablen keine (ADS-) Symbole erzeugt. Es kann also nicht symbolisch darauf zugegriffen werden.

Die Wirkung von 'hide' betrifft auch Variablen und Signaturen innerhalb von Bibliotheken, die als .library vorhanden sind.

Syntax: {attribute 'hide'}

Einfügeort: Zeile oberhalb einer Signatur, Zeile oberhalb der Deklarationszeile einer Variablen (dabei wird nur die direkt auf das Pragma folgende Variable unsichtbar gemacht) oder bei ST-Aktionen/-Transitionen direkt zu Beginn der Aktion/Transition

Beispiel:

Der Funktionsbaustein FB_Sample verwendet das Attribut {attribute 'hide'}:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nA         : INT;
    {attribute 'hide'}
    bInvisible : BOOL;
    bVisible   : BOOL;
END_VAR
VAR_OUTPUT
    nB : INT;
END_VAR
```

Im Hauptprogramm wird eine Instanz des Funktionsbausteins FB_Sample definiert.

```
PROGRAM MAIN
VAR
    fbSample   : FB_Sample;
END_VAR
```

Während nun ein Eingabewert für fbSample implementiert wird, zeigt die Funktion **Komponenten auflisten**, die sich beim Tippen von „fbSample.“ (im Implementierungsteil von MAIN) öffnet, die Variablen nA, bVisible und nB an, nicht aber die versteckte Variable bInvisible.

Mit dem Pragma hide_all_locals [▶ 842] können Sie alle lokalen Variablen einer Deklaration verstecken.

Wenn das Pragma hide in Compiled Libraries für Variablen und Signaturen verwendet wird, werden diese Variablen und Signaturen auch nicht im Bibliotheksverwalter angezeigt.

Siehe auch:

- Attribut 'hide_all_locals' [▶ 842]
- Attribut 'conditionalshow' [▶ 833]
- Attribut 'conditionalshow_all_locals' [▶ 834]

16.8.2.17 Attribut 'hide_all_locals'

Dieses Attribut wirkt wie das Attribut 'hide' [▶ 841]. Der einzige Unterschied ist, dass das Attribut 'hide' nur eine Variable betrifft, wohingegen von dem Attribut 'hide_all_locals' alle lokalen Variablen einer Signatur betroffen sind.

Beachten Sie, dass Variablen, die mit dem Attribut 'hide' oder 'hide_all_locals' deklariert sind, nicht als persistent abgespeichert werden können. Des Weiteren wird für „versteckte“ Variablen die Erzeugung des zugehörigen Prozessabbilds (allokierte Inputs/Outputs) verhindert. Zudem werden für diese Variablen keine (ADS-) Symbole erzeugt. Es kann also nicht symbolisch darauf zugegriffen werden.

Syntax: {attribute 'hide_all_locals'}

Einfügeort: Erste Zeile über dem Deklarationsteil der POU

Beispiel:

Der Funktionsbaustein FB_Sample verwendet das Attribut:

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nA     : INT;
END_VAR
VAR_OUTPUT
```

```
    bB      : BOOL;
END_VAR
VAR
    nC,nD   : INT;
    bE      : BOOL;
END_VAR
```

Die Wirkung, wie sie bei Verwendung des Attributs 'hide' vorliegt, betrifft bei Verwendung des Attributs 'hide_all_locals' an dem Funktionsbaustein FB_Sample alle lokalen Variablen des Bausteins (nC, nD und bE).

Siehe auch:

- Attribut 'hide' [▶ 841]

16.8.2.18 Attribut 'init_namespace'

Das Pragma bewirkt, dass eine Variable vom Typ STRING oder WSTRING, die in einem Bibliotheksbaustein mit diesem Pragma deklariert ist, bei der Verwendung im Projekt mit dem aktuellen Namensraum der Bibliothek initialisiert wird.

Syntax: {attribute 'init_namespace'}

Einfügeort: Zeile oberhalb der Deklarationszeile der Variablen in einem Bibliotheksbaustein

Beispiel:

Der Funktionsbaustein FB_Sample ist mit den nötigen Attributen versehen:

```
FUNCTION_BLOCK FB_Sample
VAR_OUTPUT
    {attribute 'init_namespace'}
    sNamespace : STRING;
END_VAR
```

Innerhalb des Hauptprogramms MAIN ist eine Instanz fbSample des Funktionsbausteins FB_Sample definiert:

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
    sMyNamespace : STRING;
END_VAR
sMyNamespace := fbSample.sNamespace;
```

Die Variable sNamespace wird mit dem aktuellen Namensraum initialisiert, zum Beispiel Tc3_TestLib. Dieser Wert wird sMyNamespace im Hauptprogramm zugewiesen.

Siehe auch:

- Bibliotheksverwalter [▶ 289]

16.8.2.19 Attribut 'init_on_onlchange'

Das Pragma bewirkt, dass die Variable, auf die das Pragma angewendet wird, bei jedem Online-Change initialisiert wird.

Schneller Online-Change

Für kleine Änderungen (z. B. kleine Änderung im Implementierungsbereich und Verschieben von Variablen nicht nötig) wird ein „schneller Online-Change“ durchgeführt. In diesem Fall wird nur der jeweils geänderte Baustein übersetzt und nachgeladen. Insbesondere wird in dem Fall kein Initialisierungscode erzeugt. Das bedeutet, dass auch kein Code zur Initialisierung von Variablen mit dem Attribut 'init_on_onlchange' erzeugt wird. In der Regel wird das keine Auswirkungen haben, da das Attribut meist dazu verwendet wird, um Variablen mit Adressen zu initialisieren, es kann aber beim schnellen Online-Change nicht dazu kommen, dass eine Variable ihre Adresse ändert.

Um die Wirkung des Attributs init_on_onlchange auf den gesamten Applikationscode sicherzustellen, schalten Sie den schnellen Online-Change mithilfe der Compiler-Definition no_fast_online_change generell für das SPS-Projekt aus. Fügen Sie die Definition zu diesem Zweck in den Eigenschaften des SPS-Projekts in der Kategorie Übersetzen [▶ 951] ein.

Keine Wirkung des Attributs 'init_on_onlchange' bei einzelnen FB-Variablen

Das Attribut 'init_on_onlchange' [▶ 843] wirkt nur bei globalen Variablen, Programmvariablen und lokalen statischen Variablen von Funktionsbausteinen.

Um einen Funktionsbaustein bei einem Online Change neu zu initialisieren muss die Funktionsbausteininstanz mit dem Attribut deklariert werden. Für eine einzelne Variable in einem Funktionsbaustein wird das Attribut nicht ausgewertet.

Syntax: {attribute 'init_on_onlchange' }

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

16.8.2.20 Attribut 'initialize_on_call'

Das Pragma kann auf Eingangsvariablen angewendet werden. Es bewirkt, dass Eingangsvariablen eines Funktionsbausteins bei jedem Aufruf des Funktionsbausteins initialisiert werden. Wenn eine Eingangsvariable betroffen ist, die einen Pointer erwartet und dieser Pointer im Zuge eines Online-Change entfernt wurde, wird die Variable mit Null initialisiert.

Syntax: {attribute 'initialize_on_call'}

Einfügeort: Immer in der ersten Zeile im Deklarationsteil für den gesamten Funktionsbaustein und zusätzlich in einer Zeile oberhalb der Deklaration der einzelnen Eingangsvariablen

Beispiel:

Wenn ein Eingang einen Zeiger erwartet und dieser Eingang mit dem Attribut versehen ist, so wird dieser Zeiger bei jedem Aufruf des Funktionsbausteins neu initialisiert. Hierdurch kann verhindert werden, dass ein im Zuge eines Online-Change ungültig gewordener Zeiger verwendet wird.

```
{attribute 'initialize_on_call'}
FUNCTION_BLOCK FB_Test
VAR_INPUT
    {attribute 'initialize_on_call'}
    pSetpoint : POINTER TO LREAL := 0;
END_VAR
VAR_OUTPUT
END_VAR
```

Der Bausteinaufruf sollte mit Zuweisung des Zeigers geschehen.

```
fbTest(pSetpoint := ADR(fSetpoint));
```

16.8.2.21 Attribut 'instance-path'

Das Pragma kann auf eine lokale STRING-Variablen angewendet werden und bewirkt, dass diese lokale STRING-Variable in Folge mit dem Gerätebaumpfad der POU, zu der sie gehört, initialisiert wird. Dies kann für Fehlermeldungen nützlich sein. Die Anwendung des Pragmas setzt die Anwendung des Attributs 'reflection' auf die zugehörige POU voraus, sowie die Anwendung des zusätzlichen Attributs 'noinit' auf die STRING-Variable selbst.

Syntax: {attribute 'instance-path'}

Einfügeort: Zeile oberhalb der Zeile mit der Deklaration der STRING-Variable

Beispiel:

Der folgende Funktionsbaustein enthält die Attribute 'reflection', 'instance-path' und 'noinit'.

```
{attribute 'reflection'}
FUNCTION_BLOCK FB_Sample
VAR
    {attribute 'instance-path'}
    {attribute 'noinit'}
    sPath : STRING;
END_VAR
```

Im Hauptprogramm MAIN ist eine Instanz fbSample des Funktionsbausteins FB_Sample definiert:

```
PROGRAM MAIN
VAR
    fbSample : FB_Sample;
    sMyPath : STRING;
END_VAR
fbSample();
sMyPath := fbSample.sPath;
```

Nach der Initialisierung von Instanz fbSample wird der String-Variablen sPath der Pfad der Instanz fbSample zugewiesen, im Beispiel <project>.MAIN.fbSample. Dieser Pfad wird im Hauptprogramm der Variablen sMyPath zugewiesen.

Die Länge einer Zeichenkette können Sie beliebig definieren (auch >255), allerdings müssen Sie bedenken, dass die Zeichenkette (von hinten her) abgeschnitten wird, wenn sie einer Variablen zugewiesen ist, deren Datentyp zu klein dafür ist.

Siehe auch:

- Attribut 'reflection' [▶ 860]
- Attribut 'noinit' [▶ 852]

16.8.2.22 Attribut 'is_connected'

Mit dem Pragma 'is_connected' kennzeichnen Sie eine boolsche Funktionsbausteinvariable, die beim Aufruf einer Funktionsbausteininstanz Information darüber gibt, ob der zugeordnete Eingang des Bausteins eine Zuweisung erhält.

Die Anwendung des Pragmas setzt die Anwendung des Attributs 'reflection' auf den betroffenen Funktionsbaustein voraus.

Syntax: {attribute 'is_connected' := '<input variable>'}

Einfügeort: Zeile oberhalb der Deklaration der einzelnen boolschen Funktionsbausteinvariablen

Beispiel:

Im Funktionsbaustein FB wird für jede Eingangsvariable (nIn1 und nIn2) eine lokale Variable deklariert und dieser jeweils das Attribut 'is connected' mit Angabe der Eingangsvariablen vorangestellt. Der Funktionsbaustein selbst erhält das Pragmaattribut 'reflection'.

Wenn eine Instanz des Funktionsbausteins aufgerufen wird, wird die lokale Variable TRUE, falls der ihr zugeordnete Eingang eine Zuweisung bekommen hat.

```
{attribute 'reflection'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nIn1: INT;
    nIn2: INT;
END_VAR
VAR
    {attribute 'is_connected' := 'nIn1'}
    bIn1Connected: BOOL;
    {attribute 'is_connected' := 'nIn2'}
    bIn2Connected: BOOL;
END_VAR
```

Annahme: nIn1 erhält beim Aufruf der Funktionsbausteininstanz eine Zuweisung von außerhalb, nIn 2 erhält zu diesem Zeitpunkt keine Zuweisung. Dann entsteht folgender Code:

```
bIn1Connected := TRUE;
bIn2Connected := FALSE;
```

16.8.2.23 Attribut 'linkalways'

Das Pragma bewirkt, dass das zugehörige Objekt beim Compiler markiert ist und damit immer in der Compilerinformation enthalten ist. Dies bedeutet, dass das Objekt immer kompiliert und auf die SPS geladen wird. Die Option **Always link** in den Objekteigenschaften, Kategorie **Advanced**, hat die gleiche Wirkung.

Syntax: {attribute 'linkalways'}

Einfügeort:

POU: Erste Zeile über dem Deklarationsteil der POU

GVL: Zeile oberhalb der Zeile mit dem Schlüsselwort VAR_GLOBAL im Deklarationsteil

Beispiel:

Implementierung einer POU (hier: Programm), die das Attribut 'linkalways' verwendet:

```
{attribute 'linkalways'}
PROGRAM PRG_Test
    bSample : BOOL;
END_VAR
```

Implementierung einer globalen Variablenliste, die das Attribut 'linkalways' verwendet:

```
{attribute 'linkalways'}
VAR_GLOBAL
    nVar1 : INT;
    nVar2 : INT;
END_VAR
```

16.8.2.24 Attribut 'minimal_input_size'

Das Pragma definiert die minimale Größe für Eingänge auf dem Stack. Für die C++-Kompatibilität mit C++-Compilern für 32-Bit-Systeme belegt jeder Eingang mindestens 4 Bytes auf dem Stack.

Syntax: {attribute 'minimal_input_size'}

Einfügeort: Zeile oberhalb der Methodendeklaration

Beispiel:

Deklaration einer Methode eines SPS-Funktionsbausteins, der eine C++-kompatible Schnittstelle implementiert:

```
{attribute 'c++_compatible'}
{attribute 'minimal_input_size' := '4'}
{attribute 'pack_mode' := '4'}
METHOD Method1 : HRESULT
VAR_INPUT
    nParameter1 : INT;
END_VAR
```

Siehe auch:

- Attribut 'c++_compatible' [▶ 830]

16.8.2.25 Attribut 'monitoring'

Das Pragma bewirkt, dass Sie Werte von Eigenschaften oder Funktionsaufrufen in der Online-Ansicht des IEC-Editors oder in einer Überwachungsliste monitoren können. Dazu gibt es zwei mögliche Attributwerte: 'variable' und 'call'.

Syntax:

{attribute 'monitoring' := 'variable'}

oder

{attribute 'monitoring' := 'call'}

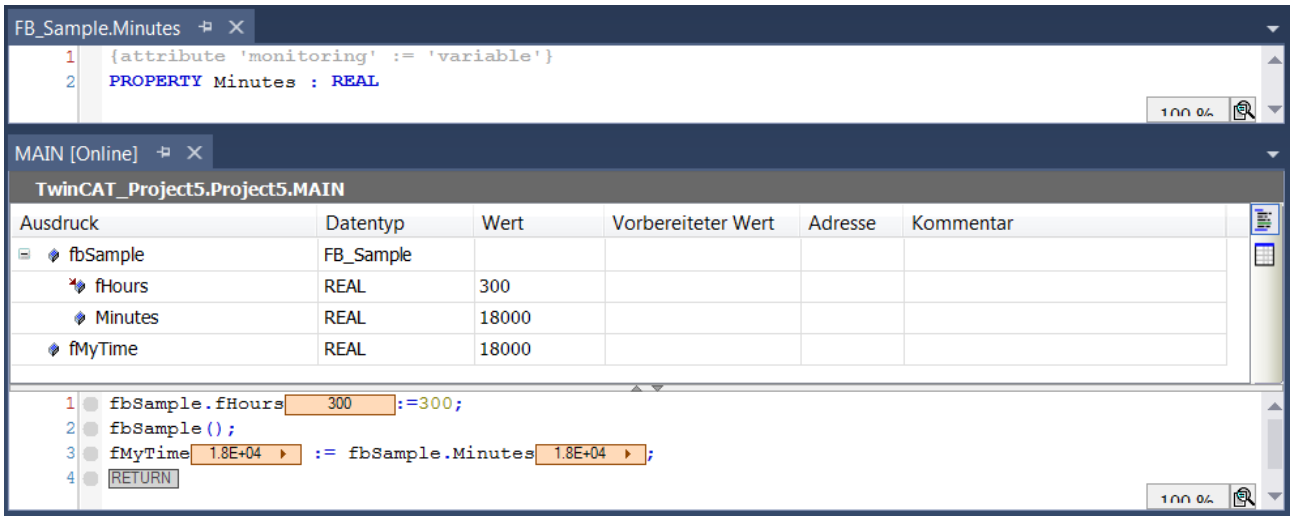
Monitoring von Eigenschaften

Sie können in der Onlineansicht eines Funktionsbausteins oder eines Programms zusätzlich zu den lokalen Variablen die untergeordneten Eigenschaften monitoren. Sie können somit die Werte der Get- und Set-Methoden überwachen.

Fügen Sie entweder das Pragma {attribute 'monitoring' := 'variable'} oder das Pragma {attribute 'monitoring' := 'call'} in der Deklaration der Eigenschaft ein. Dann werden automatisch die aktuellen Werte der Eigenschaft im IEC-Editor oder in einer Überwachungsliste angezeigt.

Beispiel:

TwinCAT zeigt während des Onlinebetriebs den Wert für Minutes an der Aufrufstelle inline an, denn in der Deklaration der Eigenschaft Minutes steht das Pragma {attribute 'monitoring' := 'variable'}.



Prüfen Sie für jeden Anwendungsfall sorgfältig, welches Attributpragma geeignet ist, den gewünschten Wert anzuzeigen. Dies hängt davon ab, ob innerhalb der Eigenschaft weitere Operationen mit den Variablen implementiert sind.

1. Pragma {attribute 'monitoring':='variable'}:

Für die Eigenschaft wird eine implizite Variable angelegt, die immer dann den aktuellen Eigenschaftswert erhält, wenn das SPS-Programm die Set- oder Get-Methode aufruft. Der zuletzt in dieser Variablen gespeicherte Wert wird im Monitoring dargestellt.

2. Pragma {attribute 'monitoring':='call'}:

Dieses Attribut können Sie nur für Eigenschaften verwenden, die einfache Datentypen oder Pointer zurückgeben, nicht für strukturierte Typen. Der zu monitorende Wert wird durch direktes Aufrufen der Eigenschaft gelesen oder geschrieben. Das bedeutet, dass der Monitoring-Dienst des Laufzeitsystems die Get- oder Set-Methode der Eigenschaft-Funktion ausführt.

Wenn Sie diesen Monitoringtyp wählen, anstatt eine implizite Variable zu verwenden (wie bei {attribute 'monitoring':='variable'}), müssen Sie mögliche Seiteneffekte bedenken. Solche Seiteneffekte können auftreten, wenn innerhalb der Eigenschaftsfunktion zusätzliche Operationen implementiert sind.

Mit dem Kontextmenü-Befehl **Zur Überwachungsliste hinzufügen** wird im Onlinebetrieb eine Variable, auf der gerade der Cursor steht, unmittelbar in eine Überwachungsliste aufgenommen.

Forcen oder Schreiben von Funktionen wird nicht unterstützt. Forcen können Sie jedoch implizit implementieren, indem Sie einen zusätzlichen Eingabeparameter für die jeweilige Funktion hinzufügen, der als internes Force-Flag dient.

Funktionsmonitoring ist im Kompaktlaufzeitsystem nicht möglich.

Hinweis: Windows CE ist kein Kompaktlaufzeitsystem. Ein Kompaktlaufzeitsystem unterstützt kein Multitasking und besitzt i.d.R. kein Datei- und Betriebssystem. Ein typischer Prozessor eines Kompaktlaufzeitsystems ist beispielsweise ein ARM Cortex™-M.

16.8.2.26 Attribut 'no_assign', Attribut 'no_assign_warning'

Das Pragma 'no_assign' bewirkt, dass Compilerfehler ausgegeben werden Solche Zuweisungen sollen oft vermieden werden, wenn der Funktionsbaustein Pointer enthält, da Pointer zu Problemen führen, weil sie bei der Wertzuweisung mit kopiert werden.

Bei Verwendung des Pragmas 'no_assign_warning' wird eine Compilerwarnung ausgegeben, wenn eine Instanz des Funktionsbausteins einer anderen Instanz desselben Bausteins zugewiesen wird.

Syntax: {attribute 'no_assign'} bzw. {attribute 'no_assign_warning'}

Einfügeort: Erste Zeile im Deklarationsteil eines Funktionsbausteins

Beispiel:

Da der Funktionsbaustein FB_Test Pointer enthält, sollte die Zuweisung einer Funktionsbausteininstanz vermieden werden, indem das Attribut 'no_assign' in der Deklaration des Funktionsbausteins FB_Test hinzufügt wird:

```
{attribute 'no_assign'}
FUNCTION_BLOCK FB_Test
VAR
    pVar      : POINTER TO LREAL;
…
```

Zuweisung von Funktionsbausteininstanzen:

```
VAR_GLOBAL
    fbInst1   : FB_Test;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbInst2   : FB_Test := fbInst1;
END_VAR
```

Dann wird der folgende Compile-Fehler ausgegeben:

```
Assignment not allowed for type FB_Test
```

16.8.2.27 Attribut 'no_check'

Das Pragma bewirkt, dass für die POU keine Check-Funktion (POUs für implizite Prüfungen) aufgerufen werden soll. Da die Check-Funktionen die Abarbeitungsgeschwindigkeit des Programms beeinflussen können, kann es sinnvoll sein, das Attribut auf Bausteine anzuwenden, die schon geprüft sind oder oft aufgerufen werden.

Syntax: {attribute 'no_check'}

Einfügeort: Erste Zeile im Deklarationsteil der POU

- Objekt POUs für implizite Prüfungen [▶ 172]

16.8.2.28 Attribut 'no_copy'

Wenn eine Instanz, zum Beispiel einer POU, bei einem Online-Change im Speicher verschoben wird, erfordert dies eine Reallozierung dieser Instanz. Dabei werden die Werte der in der Instanz enthaltenen Variablen kopiert, sodass die Variablen nach dem Online-Change die gleichen Werte besitzen wie vor dem Online-Change. Falls bei einem Online-Change Instanzen/Variablen verschoben werden müssen, informiert ein Dialog über die Effekte und ermöglicht, den Online-Change abzubrechen.

Das Pragma bewirkt, dass im Zuge des Online-Change-Kopiervorgangs einer verschobenen Instanz keine Kopie des Variablenwerts der in der Instanz enthaltenen Variablen stattfindet; stattdessen wird die Variable im Zuge des Online-Changes neu initialisiert. Das kann für eine lokale Zeigervariable sinnvoll sein, die auf eine Variable zeigt, die gerade durch den Online-Change verschoben wurde und somit eine veränderte Adresse besitzt.

Syntax: {attribute 'no_copy'}

Einfügeort: Zeile oberhalb der Deklarationszeile der betroffenen Variablen

16.8.2.29 Attribut 'no_explicit_call'

Das Pragma fügen Sie einer POU hinzu, um jeden direkten Aufruf dieser POU zu unterbinden.

Dies ist beispielsweise für objektorientierte Funktionsbausteine relevant, die ausschließlich über Methoden (Method-Bausteine) und Eigenschaften (Property-Bausteine) angesteuert werden können. Ein Aufruf des FB-Bodys wäre in diesem Fall funktionslos bzw. nicht erlaubt. Wenn der Body eines Funktionsbausteins, der mit dem Attribut 'no_explicit_call‘ deklariert ist, dennoch direkt aufgerufen wird, weist der Compiler mithilfe eines Kompilierfehlers auf diese fehlerhafte Nutzung des Funktionsbausteins hin.

Syntax: {attribute 'no_explicit_call' := '<text value>'}

Einfügeort: Oberste Zeile im Deklarationsteil eines Funktionsbausteins.

Beispiel:

In dem folgenden Beispiel wird ein Funktionsbaustein deklariert, dessen FB-Body direkt aufgerufen werden darf (FB_DirectCallAllowed). Ein weiterer Funktionsbaustein wird deklariert, dessen FB-Body hingegen nicht direkt aufgerufen werden darf (FB_DirectCallNotAllowed). Daher wird dieser Funktionsbaustein mit dem Attribut 'no_explicit_call‘ deklariert, wobei das Attribut mit dem Hinweis-Text 'do not call this POU directly‘ versehen wird. Des Weiteren wird beiden Funktionsbausteinen eine beispielhafte Methode hinzugefügt.

Die beiden Funktionsbausteine werden je einmal instanziiert und direkt aufgerufen. Auch die Beispielmethoden werden für die FB-Instanzen aufgerufen.

Beim Übersetzen dieses Programms wird für den direkten Aufruf der Instanz fbDirectCallNotAllowed ein entsprechender Kompilierfehler erzeugt (in diesem Fall: „do not call this POU directly“). Die drei anderen Aufrufe werden vom Compiler nicht unterbunden.

Funktionsbaustein FB_DirectCallAllowed:

```
FUNCTION_BLOCK FB_DirectCallAllowed
VAR
END_VAR

METHOD SampleMethod
VAR_INPUT
END_VAR
```

Funktionsbaustein FB_DirectCallNotAllowed:

```
{attribute 'no_explicit_call' := 'do not call this POU directly'}
FUNCTION_BLOCK FB_DirectCallNotAllowed
VAR
END_VAR
```

```
METHOD SampleMethod
VAR_INPUT
END_VAR
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbDirectCallAllowed     : FB_DirectCallAllowed;
    fbDirectCallNotAllowed  : FB_DirectCallNotAllowed;
END_VAR
```

```
fbDirectCallAllowed();                  // => OK
fbDirectCallAllowed.SampleMethod();     // => OK

fbDirectCallNotAllowed();               // => NOK: generates compile error "do not call this POU
directly"
fbDirectCallNotAllowed.SampleMethod();  // => OK
```

16.8.2.30 Attribut 'no_virtual_actions'

Das Pragma wird für Funktionsbausteine angewendet, die von einem in AS implementierten Funktionsbaustein abgeleitet werden und den grundsätzlichen AS-Ablauf dieser Basisklasse nutzen. Die daraus aufgerufenen Aktionen zeigen dasselbe virtuelle Verhalten wie Methoden. Dies bedeutet, dass die Implementierungen der Aktionen in der Basisklasse von der abgeleiteten Klasse durch eigene, spezifische Implementierungen ersetzt werden können.

Wenn Sie das Pragma auf die Basisklasse anwenden, dann sind ihre Aktionen vor einem Überladen geschützt.

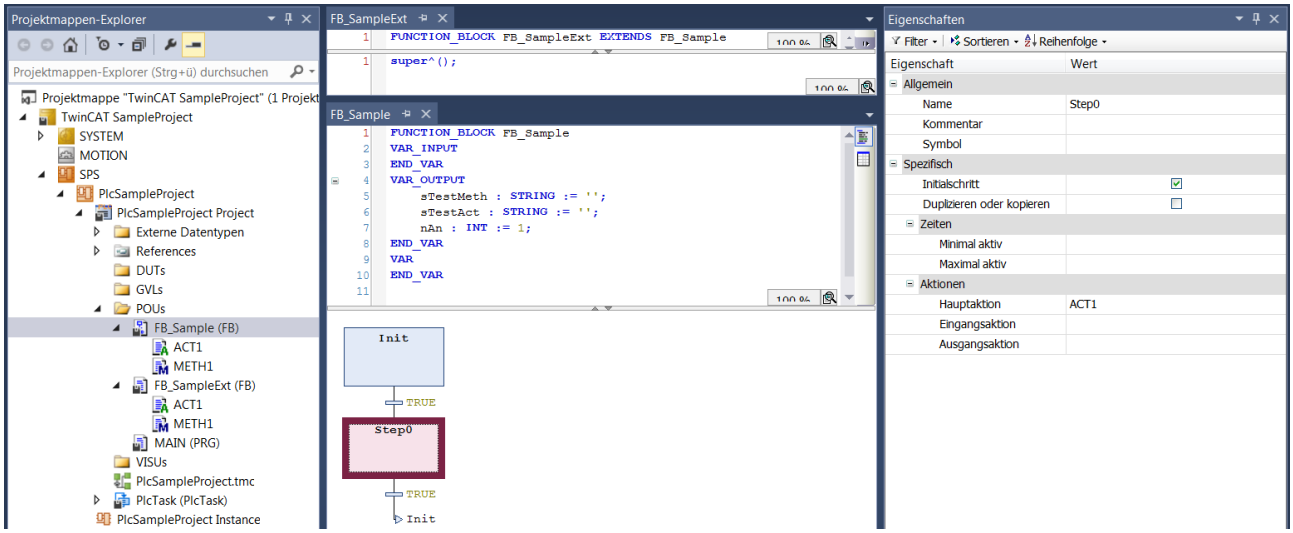
Syntax: {attribute 'no_virtual_actions'}

Einfügeort: Oberste Zeile im Deklarationsteil des Funktionsbausteins

Beispiel:

Der Funktionsbaustein FB_Sample ist die Basisklasse für den abgeleiteten Funktionsbaustein FB_SampleExt.

Mit der speziellen Variablen SUPER ruft die abgeleitete Klasse FB_SampleExt den in AS geschriebenen Ablauf der Basisklasse auf.

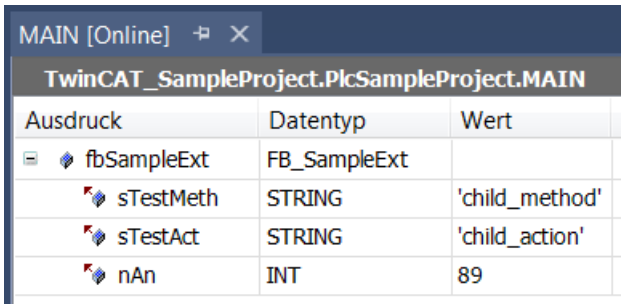


Die beispielhafte Implementierung dieses Ablaufs ist beschränkt auf den Initialschritt gefolgt von einem einzigen Schritt mit angebundener Schrittaktion ACT1. Dieser Schritt mit angebundener Schrittaktion übernimmt die Belegung der Ausgangsvariablen.

```
nAN := nAn + 1; // Counting the action calls
sTestAct:='father_action';
METH1();  // Call of the method METH1 in order to set the string variable sTestMeth
```

Im Fall der abgeleiteten Klasse FB_SampleExt wird die Schrittaktion durch eine spezielle Implementierung von ACT1 ersetzt. ACT1 unterscheidet sich vom Original nur durch Zuweisung der Zeichenkette 'child_action' anstelle von 'father_action' an die Variable `sTestAct`.

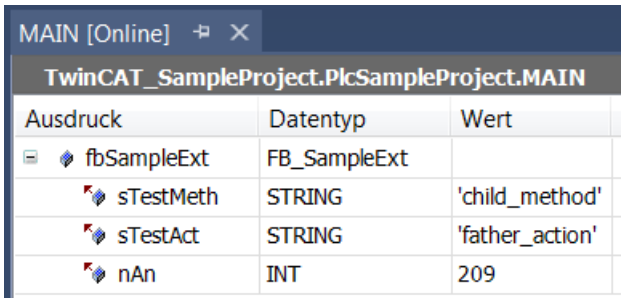
Ebenso wird die Methode `METH1`, die in der Basisklasse der Variablen `sTestMeth` den String 'father_method' zuweist, dahingehend überschrieben, dass `sTestMeth` nun den Wert 'child_method' erhält. Das Hauptprogramm MAIN ruft eine Instanz des Funktionsbausteins FB_SampleExt namens „fbSampleExt“ auf. Wie erwartet spiegelt der Wert der Strings den Aufruf von Aktion und Methode der abgeleiteten Klasse wider:



Nun stellen Sie der Basis jedoch das Pragma {attribute 'no_virtual_actions'} voran:

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK FB_Sample...
```

Dadurch ändert sich das Verhalten: Während für Methode `METH1` weiterhin die Implementierung der abgeleiteten Klasse herangezogen wird, resultiert der Aufruf der Schrittaktion nun in einem Aufruf der Aktion `ACT1` der Basisklasse. Daher erhält `sTestAct` nun den Wert 'father_action':



16.8.2.31 Attribut 'no-exit'

Das Pragma unterdrückt den Aufruf der FB_exit-Methode eines Funktionsbausteins für eine bestimmte seiner Instanzen.

Syntax: {attribute 'no-exit'}

Einfügeort: Zeile vor der Deklaration der Funktionsbaustein-Instanz

Beispiel:

Dem Funktionsbaustein FB_Sample ist die Methode FB_exit hinzugefügt. Im Hauptprogramm MAIN werden 2 Instanzen des Funktionsbausteins FB_Sample angelegt.

```
PROGRAM MAIN
VAR
    fbSample1 : FB_Sample;
    {attribute 'no-exit'}
    fbSample2 : FB_Sample;
END_VAR
```

fbSample1.FB_exit wird aufgerufen, fbSample2.FB_exit wird nicht aufgerufen.

Siehe auch:

- Methoden FB_init, FB_reinit und FB_exit [▶ 887]

16.8.2.32 Attribut 'noflow' / 'flow'

Die Pragmas weisen einen Bereich aus, der von der Darstellung der Ablaufkontrolle ausgeschlossen wird. Siehe Befehl Ablaufkontrolle [▶ 1004].

Syntax:

```
{noflow}

{flow}
```

Einfügeort: Zeile oberhalb und unterhalb des Bereichs, der von der Ablaufkontrolle nicht dargestellt werden soll.

Beispiel:

```
IF Test THEN
IF Test1 THEN
{noflow}
IF Test2 THEN
;
END_IF
{flow}
END_IF
END_IF
```

16.8.2.33 Attribut 'noinit'

Das Pragma wird auf Variablen angewendet, die nicht implizit initialisiert werden sollen.

Syntax:

{attribute 'no_init'}

{attribute 'no-init'}

{attribute 'noinit'}

Einfügeort: Zeile oberhalb der Deklarationszeile der betroffenen Variablen

Beispiel:

```
PROGRAM MAIN
VAR
    nA : INT;
    {attribute 'no_init'}
    nB : INT;
END_VAR
```

Bei einem Reset des zugehörigen SPS-Projekts wird die Integer-Variable nA erneut implizit mit 0 initialisiert, wohingegen die Variable nB ihren aktuellen Wert beibehält.

16.8.2.34 Attribut 'obsolete'

Das Pragma bewirkt, dass zu einer Datentyp-Definition beim Übersetzen eine definierte Warnung ausgegeben wird, wenn der Datentyp (Struktur, Funktionsbaustein etc.) im Projekt verwendet wird. Damit können Sie beispielsweise darauf hinweisen, dass ein Datentyp nicht mehr gültig ist, weil sich beispielsweise eine Schnittstelle geändert hat und dies im Projekt nachgezogen werden sollte.

Im Unterschied zu einem Meldungspragma wird diese Warnung zentral für alle Instanzen eines Datentyps definiert.

Syntax: {attribute 'obsolete' := 'user defined text'}

Einfügeort: Zeile der Datentyp-Definition oder in einer Zeile darüber.

Beispiel:

Das Pragma wird in der Definition Funktionsbaustein FB_Sample eingefügt:

```
{attribute 'obsolete' := 'datatype FB_Sample() not valid!'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nVar : INT;
END_VAR
```

Wenn Sie FB_Sample als Datentyp verwenden, zum Beispiel in `fbSample : FB_Sample;` wird beim Übersetzen des Projekts die Warnung ausgegeben: "datatype FB_Sample not valid".

Siehe auch:

- Meldungspragmas [▶ 828]

16.8.2.35 Attribut 'pack_mode'

Das Pragma legt fest, wie eine Datenstruktur während der Allozierung gepackt wird. Das Attribut muss oberhalb der Datenstruktur eingefügt werden und wirkt sich auf das Packen der gesamten Struktur aus.

Syntax: {attribute 'pack_mode' := '<pack mode value>'}

Mögliche Werte für <pack mode value>:

<pack mode value>	Assoziierte Packungsart	Beschreibung
0	aligned	Es liegen alle Variablen auf Byte-Adressen; es treten keine Speicherlücken auf.
1	1-byte-aligned	
2	2-byte-aligned	Es liegen • 1 Byte-Variablen auf Byte-Adressen • 2 Byte-Variablen auf durch 2-teilbare Adressen. Es entsteht maximal eine Lücke von 1 Byte • 4 Byte-Variablen auf durch 2-teilbare Adressen. Es entsteht maximal eine Lücke von 1 Byte • 8 Byte-Variablen auf durch 2-teilbare Adressen. Es entsteht maximal eine Lücke von 1 Byte • Strings immer an Byte-Adressen. Es entsteht keine Lücke.
4	4-byte-aligned	Es liegen • 1 Byte-Variablen auf Byte-Adressen • 2 Byte-Variablen auf durch 2-teilbare Adressen. Es entsteht maximal eine Lücke von 1 Byte • 4 Byte-Variablen auf durch 4-teilbare Adressen. Es entsteht maximal eine Lücke von 3 Byte • 8 Byte-Variablen auf durch 4-teilbare Adressen. Es entsteht maximal eine Lücke von 3 Byte • Strings immer an Byte-Adressen. Es entsteht keine Lücke.
8	8-byte-aligned	Es liegen • 1 Byte-Variablen auf Byte-Adressen • 2 Byte-Variablen auf durch 2-teilbare Adressen. Es entsteht maximal eine Lücke von 1 Byte • 4 Byte-Variablen auf durch 4-teilbare Adressen. Es entsteht maximal eine Lücke von 3 Byte • 8 Byte-Variablen auf durch 8-teilbare Adressen. Es entsteht maximal eine Lücke von 7 Byte • Strings immer an Byte-Adressen. Es entsteht keine Lücke.

Einfügeort: Zeile oberhalb der Deklaration der Datenstruktur

Abhängig vom Aufbau der Struktur kann es somit vorkommen, dass es keinen Unterschied in der Speicheraufteilung zwischen den einzelnen Modi gibt. Somit kann die Speicherverteilung einer Struktur mit pack_mode =4 der von pack_mode = 8 entsprechen.

Arrays von Strukturen: Werden die Strukturen in Arrays zusammengefasst, so werden am Ende der Struktur Bytes eingefügt, damit die nächste Struktur wieder aligned ist.

Beispiel 1

```
{attribute 'pack_mode' := '1'}
TYPE ST_MyStruct:
STRUCT
    bEnable         : BOOL;
    nCounter        : INT;
    nMaxSize        : INT;
    bMaxSizeReached : BOOL;
    bReset          : BOOL;
END_STRUCT
END_TYPE
```

Der Speicherbereich für eine Variable vom Datentyp ST_MyStruct wird „aligned“ alloziert: Ist die Speicheradresse ihrer Komponente bEnable beispielsweise 0x0100, dann folgt die Komponente nCounter an der Adresse 0x0101, nMaxSize an Adresse 0x0103, bMaxSizeReached an Adresse 0x0105 und bReset an Adresse 0x0106. Mit 'pack_mode' := '2' läge nCounter bei 0x0102, nMaxSize bei 0x0104, bMaxSizeReached bei 0x0106 und bReset bei 0x0107.

Beispiel 2

```
STRUCT
    bVar1 : BOOL  := 16#01;
    nVar2 : BYTE  := 16#11;
    nVar3 : WORD  := 16#22;
    nVar4 : BYTE  := 16#44;
    nVar5 : DWORD := 16#88776655;
    nVar6 : BYTE  := 16#99;
    nVar7 : BYTE  := 16#AA;
    nVar8 : DWORD := 16#AA;
END_TYPE
```

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	Variable	Wert	Variable	Wert	Variable	Wert	Variable	Wert	Variable	Wert
0	Var1	01	Var1	01	Var1	01	Var1	01	Var1	01
1	Var2	11	Var2	11	Var2	11	Var2	11	Var2	11
2	Var3	22	Var3	22	Var3	22	Var3	22	Var3	22
3	...	00	...	00	...	00	...	00	...	00
4	Var4	44	Var4	44	Var4	44	Var4	44	Var4	44
5	Var5	55	Var5	55						
6	...	66	...	66	Var5	55				
7	...	77	...	77	...	66				
8	...	88	...	88	...	77	Var5	55	Var5	55
9	Var6	99	Var6	99	...	88	...	66	...	66
10	Var7	AA	Var7	AA	Var6	99	...	77	...	77
11	Var8	AA	Var8	AA	Var7	AA	...	88	...	88
12	...	00	...	00	Var8	AA	Var6	99	Var6	99
13	...	00	...	00	...	00	Var7	AA	Var7	AA
14	...	00	...	00	...	00				
15					...	00				
16							Var8	AA	Var8	AA
17							...	00	...	00
18							...	00	...	00
19							...	00	...	00
20										
21										
22										
23										
24										
25										
26										
27										
28										
29										
30										
31										

Beispiel 3

```
STRUCT
    nVar1 : BYTE  := 16#01;
    nVar2 : LWORD := 16#11;
    nVar3 : BYTE  := 16#22;
    nVar4 : BYTE  := 16#44;
    nVar5 : DWORD := 16#88776655;
    nVar6 : BYTE  := 16#99;
    nVar7 : BYTE  := 16#AA;
    nVar8 : WORD  := 16#AA;
END_TYPE
```

	pack_mode = 0		pack_mode = 1		pack_mode = 2		pack_mode = 4		pack_mode = 8	
	Variable	Wert	Variable	Wert	Variable	Wert	Variable	Wert	Variable	Wert
0	Var1	01	Var1	01	Var1	01	Var1	01	Var1	01
1	Var2	11	Var2	11						
2	...	00	...	00	Var2	11				
3	...	00	...	00	...	00				
4	...	00	...	00	...	00	Var2	11		
5	...	00	...	00	...	00	...	00		
6	...	00	...	00	...	00	...	00		
7	...	00	...	00	...	00	...	00		
8	...	00	...	00	...	00	...	00	Var2	11
9	Var3	22	Var3	22	...	00	...	00	...	00
10	Var4	44	Var4	44	Var3	22	...	00	...	00
11	Var5	55	Var5	55	Var4	44	...	00	...	00
12	...	66	...	66	Var5	55	Var3	22	...	00
13	...	77	...	77	...	66	Var4	44	...	00
14	...	88	...	88	...	77			...	00
15	Var6	99	Var6	99	...	88			...	00
16	Var7	AA	Var7	AA	Var6	99	Var5	55	Var3	22
17	Var8	AA	Var8	AA	Var7	AA	...	66	Var4	44
18	...	00	...	00	Var8	AA	...	77		
19					...	00	...	88		
20							Var6	99	Var5	55
21							Var7	AA	...	66
22							Var8	AA	...	77
23							...	00	...	88
24									Var6	99
25									Var7	AA
26									Var8	AA
27									...	00
28										
29										
30										
31										

16.8.2.36 Attribut 'parameterstringof'

Mit dem Pragma kann auf den Instanznamen einer Variablen über die Visualisierung zugegriffen werden.

Syntax: {attribute 'parameterstringof' := '<variable>'}

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

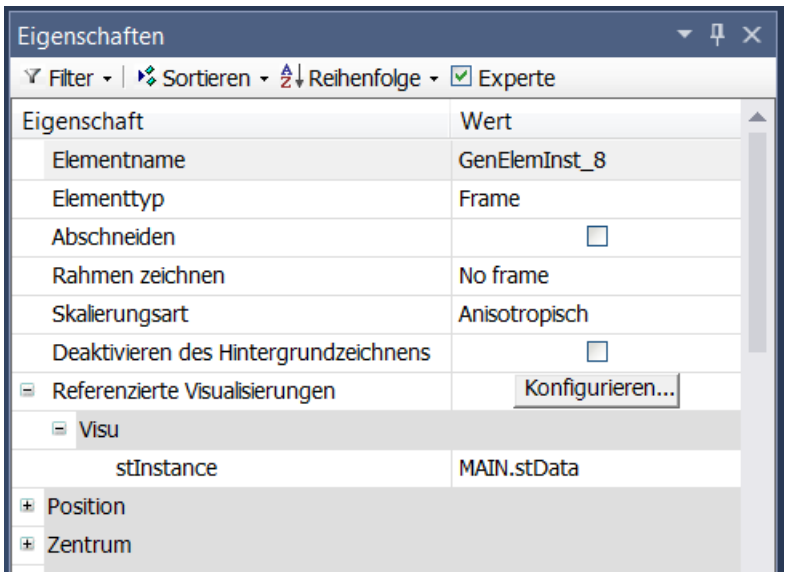
Beispiel:

Im Hauptprogramm wird die Instanz stData der benutzerdefinierten Struktur ST_Data angelegt.

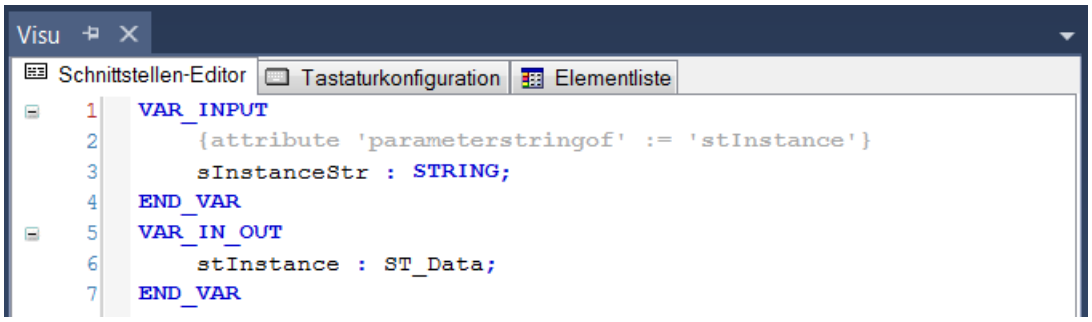
```
PROGRAM MAIN
VAR
    stData  : ST_Data;
END_VAR
```

Diese Instanz ist Eingabe einer Visualisierung „Visu“ (im Ein-/Ausgabeparameter stInstance). Die Visualisierung wird von einem Frame einer anderen Visualisierung „MainVisu“ referenziert.

Die Einstellungen des Frame-Elements in „MainVisu“ sehen wie folgt aus:



In dem zur Visualisierung „Visu“ gehörenden Schnittstelleneditor werden die Input/Output-Variable stInstance und eine weitere Input-Variable sInstanceStr deklariert:



Obwohl sInstanceStr eine Input-Variable ist, ist sie nicht als Eingang bei der Referenz mit aufgeführt (vergleiche oberes Bild). Das liegt daran, dass die Variable sInstanceStr das Attribut ‘parameterstringof‘ trägt und somit automatisch mit dem Namen der Variablen initialisiert wird, der beim Attribut angegeben wird. Im Beispiel ist stInstance die zugehörige Variable. Die String-Variable sInstanceStr wird also auf MAIN.stData gesetzt und kann nun innerhalb der Visualisierung „Visu“ verwendet werden, beispielsweise als Textvariable für einen Platzhalter „%s“.

16.8.2.37 Attribut 'pin_presentation_order_inputs/outputs'

Die Pragmas werden in den grafischen Editoren CFC und FUP/KOP ausgewertet und bewirken, dass die Eingänge/Ausgänge des betroffenen Funktionsbausteins in der angegebenen Reihenfolge dargestellt werden. Sie programmieren die Reihenfolge, indem Sie die Namen der Eingänge/Ausgänge dem Attribut in der gewünschten Reihenfolge zuweisen.

Syntax:

{attribute 'pin_presentation_order_inputs' := '<First_Input_Name>, (<Next_Input_Name>,)* (*,)? (<Next_Input_Name>,)* <Last_Input_Name>'}

{attribute 'pin_presentation_order_outputs' := '<First_Output_Name>, (<Next_Output_Name>,)* (*,)? (<Next_Output_Name>,)* <Last_Output_Name>'}

- *
  Das optionale Terminalzeichen dient als Platzhalter für alle Eingänge/Ausgänge, die selbst nicht in der Darstellungsreihenfolge angegeben sind. Wenn das Terminalzeichen fehlt, werden die fehlenden Eingänge/Ausgänge an das Ende angehängt.
- (...)?
  Der Inhalt der runden Klammer ist optional.

- (...)*
  Der Inhalt der runden Klammer ist wiederholt optional und kann somit keinmal, einmal oder mehrfach auftreten.

Einfügeort: Erste Zeile im Deklarationsteil eines Funktionsbausteins

Verwendung des Attributs 'pin_presentation_order_inputs/outputs' in Verbindung mit dem Attribut 'pingroup'

Dieses Pragma wird nicht ausgewertet, wenn das Pragma {attribute 'pingroup' := '<Group_Name>'} verwendet wird.

Beispiel:

1. Verwendung des Attributs 'pin_presentation_order_inputs/outputs'

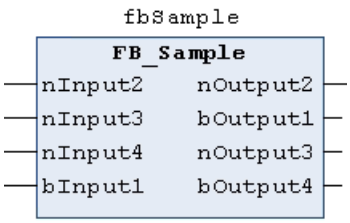
Funktionsbaustein FB_Sample:

```
{attribute 'pin_presentation_order_inputs' := 'nInput2,*,bInput1'}
{attribute 'pin_presentation_order_outputs' := 'nOutput2,nOutput1'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    bInput1 : BOOL;
    nInput2 : INT;
    nInput3 : INT;
    nInput4 : INT;
END_VAR
VAR_OUTPUT
    bOutput1 : BOOL;
    nOutput2 : INT;
    nOutput3 : INT;
    bOutput4 : BOOL;
END_VAR
```

Programm SampleProg:

```
PROGRAM SampleProg
VAR
    fbSample : FB_Sample;
END_VAR
```

Die Pragmas bewirken in der Darstellung der Funktionsbausteininstanz fbSample folgende Anordnung der Eingangs- und Ausgangspins:



Siehe auch:

- Attribut 'pingroup' [▶ 858]

16.8.2.38 Attribut 'pingroup'

Das Pragma bewirkt, dass in der Deklaration eines Funktionsbausteins die Eingangspins oder Ausgangspins (Parameter) gruppiert werden. Im FUP/KOP-Editor kann dann eine so definierte Pingruppe am eingefügten Baustein als Einheit reduziert und erweitert angezeigt werden. Mehrere Gruppen sind möglich und werden durch ihre Namen unterschieden. TwinCAT speichert den jeweiligen Zustand (reduziert) pro Bausteinbox mit den Projektoptionen.

Syntax: {attribute 'pingroup' := '<group name>'}

Einfügeort: Zeile oberhalb der Deklaration der betroffenen Eingangs- oder Ausgangsvariablen im Deklarationsteil eines Funktionsbausteins.

Beispiel:

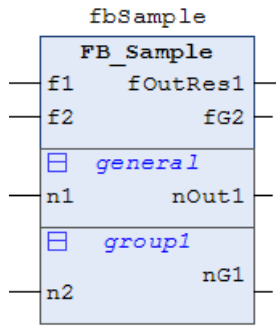
Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    f1 : REAL;
    {attribute 'pingroup' := 'general'}
    n1 : INT;
    {attribute 'pingroup' := 'group1'}
    n2 : INT;
    f2 : REAL;
END_VAR
VAR_OUTPUT
    fOutRes1 : REAL;
    {attribute 'pingroup' := 'general'}
    nOut1 : INT;
    {attribute 'pingroup' := 'group1'}
    nG1 : INT;
    fG2 : REAL;
END_VAR
```

Programm SampleProg:

```
PROGRAM SampleProg
VAR
    fbSample : FB_Sample;
END_VAR
```

Die Pragmas bewirken in der Darstellung der Funktionsbausteininstanz fbSample folgende Gruppierung der Eingangs- und Ausgangspins:



Siehe auch:

- Attribut 'pin_presentation_order_inputs/outputs' [▶ 857]

16.8.2.39 Attribut 'qualified_only'

Das Pragma bewirkt, dass Variablen einer globalen Variablenliste nur durch Angabe des globalen Variablenlistennamens angesprochen werden können (zum Beispiel GVL.nVar). Das trifft auch auf Variablen vom Typ Enumeration zu, bei denen ein Zugriff nur durch Angabe des Enumerationsnamens möglich ist (zum Beispiel E_Sample.eMember). Dies kann hilfreich sein, um eine Verwechslung mit lokalen Variablen zu vermeiden.

Bitte beachten Sie, dass eine GVL nur als Ganzes mit dem Attribut versehen werden kann. Es können nicht einzelne Variablenbereiche einer GVL mit dem Attribut deklariert werden.

Syntax: {attribute 'qualified_only'}

Einfügeort: Zeile oberhalb des ersten VAR_GLOBAL in einer GVL

Beispiel:

Globale Variablenliste GVL:

```
{attribute 'qualified_only'}
VAR_GLOBAL
    nVar : INT;
END_VAR
```

Innerhalb einer POU, zum Beispiel MAIN, kann die globale Variable nVar nur unter Verwendung des Präfix GVL angesprochen werden:

```
GVL.nVar := 5;
```

Der folgende unvollständige Aufruf der Variablen wird hingegen einen Fehler erzeugen:

```
nVar := 5;
```

16.8.2.40 Attribut 'reflection'

Das Pragma dient dazu, Bausteine zu kennzeichnen, in denen TwinCAT nach lokalen STRING-Variablen suchen soll, auf die das Attribut 'instance-path' angewendet wird. Der Compiler durchsucht nur mit 'reflection' gekennzeichnete Bausteine nach Variablen mit diesen Attributen und benötigt somit weniger Zeit.

Syntax: `{attribute 'reflection'}`

Ein Beispiel finden Sie in der Beschreibung des Attributs 'instance-path'.

Siehe auch:

- Attribut 'instance-path' [▶ 844]

16.8.2.41 Attribut 'strict'

Das Attribut 'strict' bewirkt, dass in folgenden Fällen Übersetzungsfehler entstehen:

- Arithmetische Operation mit Variablen des Enumerationstyps
- Zuweisung eines konstanten Werts, der nicht einem Enumerationswert entspricht, zu einer Variablen des Enumerationstyps
- Zuweisung eines nicht-konstanten Werts, der einen anderen Datentyp hat als der Enumerationstyp, zu einer Variablen des Enumerationstyps

Syntax: {attribute 'strict'}

Einfügeort: Zeile oberhalb der TYPE-Definition

16.8.2.42 Attribut 'subsequent'

Das Pragma wird verwendet, um Variablen direkt hintereinander an einem Speicherplatz zu allozieren. Wenn sich die Liste ändert, wird die gesamte Variablenliste an einem neuen Speicherplatz alloziiert. Dieses Pragma wird in Programmen und globalen Variablenlisten verwendet.

Syntax: {attribute 'subsequent'}

VAR_TEMP in einem Programm mit Attribut 'subsequent' führt zu einem Compiler-Fehler.

Wenn eine Variable in der Liste 'RETAIN' ist, wird die gesamte Liste als 'RETAIN' gespeichert.

16.8.2.43 Attribut 'Tc2GvlVarNames'

Das Pragma bewirkt, dass Symbole, welche in einer GVL deklariert sind, über ADS genauso angesprochen werden wie in TwinCAT 2 (ohne die Verwendung des GVL-Namens als Namespace).

Syntax: {attribute ‘Tc2GvlVarNames‘}

Beispiel:

```
{attribute 'Tc2GvlVarNames'}
VAR_GLOBAL
    nVar  AT %Q* : UINT;
END_VAR
```

16.8.2.44 Attribut 'TcCallAfterOutputUpdate'

Das Pragma definiert, ob ein Programm nach einem Output Update ausgeführt werden soll. Dieses Attribut ersetzt die TwinCAT-2-Funktionalität der Option **IO at Task begin**.

Syntax: {attribute 'TcCallAfterOutputUpdate' }

Einfügeort: Dieses Attribut muss zu allen Programm POUs hinzugefügt werden, die nach dem Output Update aufgerufen werden sollen.

Beispiel:

```
{attribute 'TcCallAfterOutputUpdate'}
PROGRAM MAIN
VAR
END_VAR
```

16.8.2.45 Attribut 'TcContextId'

Über das Pragma können Sie definieren, von welchem Task eine allokierte Variable aktualisiert werden soll.

Syntax: {attribute 'TcContextId' := '<TaskId>'}

Der in einfache Hochkommata eingeschlossene Platzhalter <TaskId> muss durch die ID des Tasks ersetzt werden. Die IDs der Tasks können in der Tabelle **Ergebnis** abgelesen werden. Die Tabelle wird in dem Reiter **Kontext** dargestellt, welcher durch Doppelklick auf das SPS-Prozessabbild (**<Projektname> Instanz**) erreicht werden kann.

Einfügeort:

- Zeile oberhalb des ersten VAR_GLOBAL in einer GVL
- Zeile oberhalb der Deklaration einer POU
- Zeile oberhalb der Deklarationszeile der allokierten Variablen

Beispiel 1:

Annahme: Der Task PlcTaskA besitzt die ID 0 und der Task PlcTaskB besitzt die ID 1.

Das Pragma wird jeweils in der Zeile oberhalb der Variablendeklaration eingefügt. Es wirkt somit nur auf die jeweils folgende Deklarationszeile.
Die Variable bVar1 wird daher von dem Task PlcTaskA aktualisiert und die Variable bVar2 von dem Task PlcTaskB.

```
VAR_GLOBAL
    {attribute 'TcContextId':='0'}
    bVar1  AT%Q* : BOOL;
    {attribute 'TcContextId':='1'}
    bVar2  AT%Q* : BOOL;
END_VAR
```

Beispiel 2:

Annahme: Der Task PlcTaskA besitzt die ID 0 und der Task PlcTaskB besitzt die ID 1.

Das Pragma wird in der GVL in der Zeile oberhalb von VAR_GLOBAL sowie in der Zeile oberhalb einer Variablendeklaration eingefügt.
Aufgrund der Anwendung des Attributs oberhalb ihrer Deklarationszeile wird die Variable bVar3 von dem Task PlcTaskB aktualisiert. Aufgrund der Verwendung oberhalb des ersten VAR_GLOBAL werden aller anderen Variablen (alle bis auf bVar3) von dem Task PlcTaskA aktualisiert.

```
{attribute 'TcContextId':='0'}
VAR_GLOBAL
    bVar1  AT%Q* : BOOL;     // => PlcTaskA
    bVar2  AT%Q* : BOOL;     // => PlcTaskA
```

```
    {attribute 'TcContextId':='1'}
    bVar3   AT%Q* : BOOL;    // => PlcTaskB

    bVar4  AT%Q* : BOOL;     // => PlcTaskA
END_VAR
```

Beispiel 3:

Annahme: Der Task PlcTaskA besitzt die ID 0 und der Task PlcTaskB besitzt die ID 1.

Das Pragma wird in der GVL in der Zeile oberhalb des ersten VAR_GLOBAL sowie in der Zeile oberhalb des zweiten VAR_GLOBAL eingefügt.
Beachten Sie, dass diese Verwendung nicht den Zweck erfüllt, um die Variablen bVar4-bVar6 von dem Task PlcTaskB aktualisieren zu lassen.

Hintergrund: Das Einfügen des Pragmas oberhalb von VAR_GLOBAL ist nur oberhalb des ersten VAR_GLOBAL einer GVL zielführend.

Das dargestellte Beispiel führt zu folgender Zuordnung: Das Pragma oberhalb des zweiten VAR_GLOBAL wird als Pragma oberhalb der Deklarationszeile von der Variablen bVar4 interpretiert, sodass die Variable bVar4 von dem Task PlcTaskB aktualisiert wird. Aufgrund der Verwendung oberhalb des ersten VAR_GLOBAL werden aller anderen Variablen (alle bis auf bVar4) von dem Task PlcTaskA aktualisiert.

Um das Attribut auf eine ganze Variablengruppe anzuwenden (z.B. bVar4-bVar6), ist die Verwendung einer eigenen GVL pro Variablengruppe zu empfehlen.

```
{attribute 'TcContextId':='0'}
VAR_GLOBAL
bVar1 AT%Q* : BOOL; // => PlcTaskA
bVar2 AT%Q* : BOOL; // => PlcTaskA
bVar3 AT%Q* : BOOL; // => PlcTaskA
END_VAR
{attribute 'TcContextId':='1'}
VAR_GLOBAL
bVar4 AT%Q* : BOOL; // => PlcTaskB
bVar5 AT%Q* : BOOL; // => PlcTaskA
bVar6 AT%Q* : BOOL; // => PlcTaskA
END_VAR
```

16.8.2.46 Attribut 'TcContextName'

Über das Pragma können Sie definieren, von welchem Task eine allokierte Variable aktualisiert werden soll.

Syntax: {attribute 'TcContextName' := '<TaskName>'}

Der in einfache Hochkommata eingeschlossene Platzhalter <TaskName> muss durch den Namen des Tasks ersetzt werden.

Einfügeort:

- Zeile oberhalb des ersten VAR_GLOBAL in einer GVL
- Zeile oberhalb der Deklaration einer POU
- Zeile oberhalb der Deklarationszeile der allokierten Variablen

Beispiel 1:

Das Pragma wird jeweils in der Zeile oberhalb der Variablendeklaration eingefügt. Es wirkt somit nur auf die jeweils folgende Deklarationszeile.
Die Variable bVar1 wird daher von dem Task PlcTaskA aktualisiert und die Variable bVar2 von dem Task PlcTaskB.

```
VAR_GLOBAL
    {attribute 'TcContextName':='PlcTaskA'}
    bVar1  AT%Q* : BOOL;
    {attribute 'TcContextName':='PlcTaskB'}
    bVar2  AT%Q* : BOOL;
END_VAR
```

Beispiel 2:

Das Pragma wird in der GVL in der Zeile oberhalb von VAR_GLOBAL sowie in der Zeile oberhalb einer Variablendeklaration eingefügt.
Aufgrund der Anwendung des Attributs oberhalb ihrer Deklarationszeile wird die Variable bVar3 von dem Task PlcTaskB aktualisiert. Aufgrund der Verwendung oberhalb des ersten VAR_GLOBAL werden aller anderen Variablen (alle bis auf bVar3) von dem Task PlcTaskA aktualisiert.

```
{attribute 'TcContextName':='PlcTaskA'}
VAR_GLOBAL
    bVar1  AT%Q* : BOOL;     // => PlcTaskA
    bVar2  AT%Q* : BOOL;     // => PlcTaskA

    {attribute 'TcContextName':='PlcTaskB'}
    bVar3  AT%Q* : BOOL;     // => PlcTaskB

    bVar4  AT%Q* : BOOL;     // => PlcTaskA
END_VAR
```

Beispiel 3:

Das Pragma wird in der GVL in der Zeile oberhalb des ersten VAR_GLOBAL sowie in der Zeile oberhalb des zweiten VAR_GLOBAL eingefügt.
Beachten Sie, dass diese Verwendung nicht den Zweck erfüllt, um die Variablen bVar4-bVar6 von dem Task PlcTaskB aktualisieren zu lassen.

Hintergrund: Das Einfügen des Pragmas oberhalb von VAR_GLOBAL ist nur oberhalb des ersten VAR_GLOBAL einer GVL zielführend.

Das dargestellte Beispiel führt zu folgender Zuordnung: Das Pragma oberhalb des zweiten VAR_GLOBAL wird als Pragma oberhalb der Deklarationszeile von der Variablen bVar4 interpretiert, sodass die Variable bVar4 von dem Task PlcTaskB aktualisiert wird. Aufgrund der Verwendung oberhalb des ersten VAR_GLOBAL werden aller anderen Variablen (alle bis auf bVar4) von dem Task PlcTaskA aktualisiert.

Um das Attribut auf eine ganze Variablengruppe anzuwenden (z.B. bVar4-bVar6), ist die Verwendung einer eigenen GVL pro Variablengruppe zu empfehlen.

```
{attribute 'TcContextName':= 'PlcTaskA'}
VAR_GLOBAL
bVar1 AT%Q* : BOOL; // => PlcTaskA
bVar2 AT%Q* : BOOL; // => PlcTaskA
bVar3 AT%Q* : BOOL; // => PlcTaskA
END_VAR
{attribute 'TcContextName':='PlcTaskB'}
VAR_GLOBAL
bVar4 AT%Q* : BOOL; // => PlcTaskB
bVar5 AT%Q* : BOOL; // => PlcTaskA
bVar6 AT%Q* : BOOL; // => PlcTaskA
END_VAR
```

16.8.2.47 Attribut 'TcDisplayScale'

Das Pragma kann angewendet werden, um den Wert einer Variablen für die Darstellung zu skalieren.

Syntax: {attribute ‘TcDisplayScale‘ := '<Value>'}

Folgende Werte für <Value> sind zulässig:

- "+/-10"
- "0-10"
- "0-20"
- "4-20"
- "0-10(16)"
- "0-20(16)"
- "4-20(16)"
- "0.1°"
- "0.01°"
- "0-5"
- "0-30"

- "0-50"
- "+/-5"
- "+/-2.5"
- "+/-100"
- "0-5(16)"
- "0-30(16)"
- "0-50(16)"
- "+/-75mV"

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

Beispiel:

```
PROGRAM MAIN
VAR
    {attribute 'TcDisplayScale' := '0-10'}
    nVar  AT %Q* : UINT;
END_VAR
```

16.8.2.48 Attribut 'TcEncoding'

Über das Pragma definieren Sie, wie eine Variable vom Typ STRING interpretiert werden soll.

Syntax: {attribute 'TcEncoding' := 'UTF-8'}

Einfügeort: Zeile oberhalb der Deklarationszeile der STRING-Variablen

Eine so deklarierte Variable vom Typ STRING nutzt das UTF-8-Format für die Zeichenformatierung des Unicode-Zeichensatzes. Die 0-Terminierung wird, wie bei jeder STRING-Variablen, auch hier angewendet.

Um Sonderzeichen und Texte verschiedener Sprachen zu unterstützen, wird der Zeichensatz also nicht auf den typischen Zeichensatz vom Datentyp STRING (oder WSTRING) beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet. Diese Formatierung ist in der IT allgemein üblich.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung eines STRING und der UTF-8-Formatierung eines STRING.

Ein einzelnes Zeichen wird bei einer STRING-Variablen immer innerhalb eines Bytes und bei einer WSTRING-Variablen immer innerhalb zwei Bytes abgespeichert. Ein einzelnes Zeichen wird bei der UTF-8-Formatierung jedoch innerhalb 1-4 Bytes abgespeichert, je nachdem, um welches Zeichen es sich handelt. Deshalb stimmt die Anzahl der Zeichen oft nicht mit der Variablengröße in Bytes überein.

Beispiel 1:

Das Pragma wird jeweils in der Zeile oberhalb der Variablendeklaration eingefügt. Es wirkt somit nur auf die jeweils folgende Deklarationszeile.

```
VAR
    sMyStringText   : STRING;
    wsMyWStringText : WSTRING;
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text     : STRING;
END_VAR
```

Beispiel 2:

Die Zuweisung eines Literals bedarf einer Hilfsfunktion, wenn der Text Zeichen beinhaltet, die nicht im ASCII-Zeichensatz definiert sind.

```
VAR
    sMyStringText   : STRING  := 'The dinner costs 30 €.';
    wsMyWStringText : WSTRING := "The dinner costs 30 €.";

    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text1 : STRING  := sLiteral_TO_UTF8('The dinner costs 30 €.');
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text2 : STRING  := wsLiteral_TO_UTF8("The dinner costs 30 €.");
    {attribute 'TcEncoding':='UTF-8'}
```

```
    sMyUTF8Text3 : STRING  := 'Hello World.';

    // verfügbar ab TC3.1 Build 4026
    {attribute 'TcEncoding':='UTF-8'}
    sMyUTF8Text4 : STRING  := UTF8#'The dinner costs 30 €.';
END_VAR
```

Weitere Hilfsfunktionen für STRING-Variablen und Konvertierungsmöglichkeiten für UTF-8 werden in der Dokumentation TwinCAT 3 PLC Lib TC2_Utilities beschrieben.

16.8.2.49 Attribut 'TcHideSubItems'

Über dieses Pragma definieren Sie, welche Unterelemente einer Variablen versteckt werden sollen. Diese sind dann im Prozessabbild des Projektmappen-Explorers nicht mehr sichtbar.

Syntax: {attribute 'TcHideSubItems'}

Einfügeort: Zeile oberhalb der Deklaration des Unterelements

Beispiel:

```
TYPE DUT :
STRUCT
a : INT;
{attribute 'TcHideSubItems'}
b : ARRAY [0..20000] OF BYTE;
END_STRUCT
END_TYPE
```

Durch das Attribut „TcHideSubItems“ wird nur die Variable „b“ und nicht mehr zusätzlich `b[0], b[1], b[2]… b[19999]` im Projektmappen-Explorer angelegt.

16.8.2.50 Attribut 'TcIgnorePersistent'

Das Pragma kann angewendet werden, um die Wiederherstellung eines persistent gespeicherten Wertes für eine Variable zu verhindern.

Syntax: {attribute ‘TcIgnorePersistent ‘}

Einfügeort: Zeile oberhalb der Variablendeklaration

16.8.2.51 Attribut 'TcInitOnReset'

Mit Hilfe des Pragmas können Sie definieren, ob eine persistente Variable bei einem **Reset kalt** neuinitialisiert werden soll.

Syntax: {attribute 'TcInitOnReset'}

Einfügeort: Zeile oberhalb der Deklarationszeile der betroffenen Variablen

Verfügbar ab TC3.1 Build 4024

Beispiele:

```
VAR PERSISTENT
    nVar1 : UINT;
    {attribute 'TcInitOnReset'}
    nVar2 : UINT;
END_VAR
```

Bei einem **Reset kalt** wird nur nVar2 neu initialisiert.

16.8.2.52 Attribut 'TcInitSymbol'

Mit dem Pragma legen Sie fest, ob eine Variable als Init-Symbol verwendet wird. Die über dieses Attribut ausgewählten Variablen sind nach dem Build-Prozess auf dem Reiter „Symbolinitialisierung“ der SPS-Instanz verfügbar. In der Spalte „Wert“ können Sie die gewünschten Werte eintragen, die den Variablen vor Beginn der Codeausführung zugewiesen werden sollen. Die Werte werden vor Beginn der Codeausführung in den Variablenwert kopiert und überschreiben die Initialwerte, die ggf. bei der Variablendeklaration angegeben sind (z.B. `nVar : INT := 123;`).

Ab TC3.1 Build 4024.4 wird der bei der Deklaration angegebene Initialwert als Anfangswert in die Tabelle übernommen. Weitere Informationen hierzu finden Sie unten.

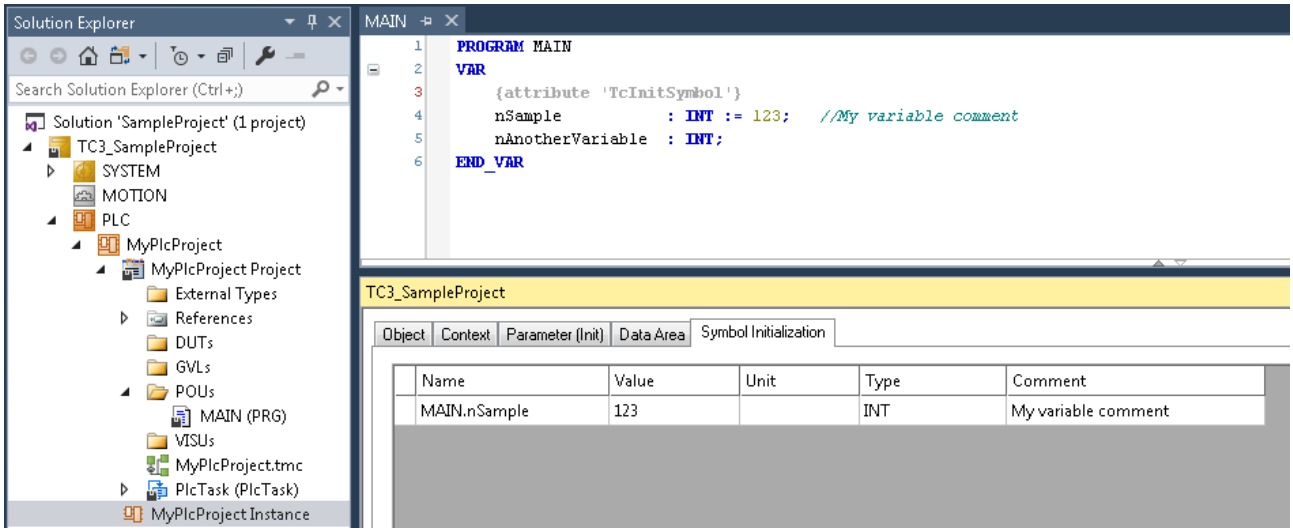
Syntax: {attribute 'TcInitSymbol'}

Einfügeort: Zeile oberhalb der Deklarationszeile der Variablen

Beispiel:

```
PROGRAM MAIN
VAR
    {attribute 'TcInitSymbol'}
    nSample           : INT := 123;   //My variable comment
    nAnotherVariable  : INT;
END_VAR
```

Diese Deklaration führt zu folgender Darstellung auf der Registerkarte **Symbol Initialisierung** der SPS-Instanz:



Spalten-Befüllung:

Die Einträge der folgenden Spalten werden von der Variablendeklaration innerhalb des SPS-Editors übernommen (siehe auch obiges Beispiel):

- Name (= Deklarationspfad + Symbolname)
- Wert: Initialwert der Variablendeklaration wird nur beim initialen Hinzufügen der Variablen zur Tabelle in die Tabelle übernommen, weitere Infos s.u.
- Typ
- Kommentar

Anfänglicher Eintrag in der Tabellenspalte „Wert“:

Szenario: Eine Variable ist mit dem Attribut 'TcInitSymbol' deklariert. Wenn diese Deklaration das erste Mal übersetzt wird, wird die Variable zu der Symbolinitialisierungstabelle hinzugefügt.

Bei diesem Hinzufügen der Variablen zu der Tabelle wird ab TC3.1 Build 4024.4 der Initialwert der Variablendeklaration (im Beispiel oben: 123) in die Tabellenspalte „Wert“ übernommen. Mit vorherigen TC3.1-Versionen wurde eine Nullinitialisierung verwendet, sodass die „Wert“-Spalte mit dem Wert 0 versehen wurde.

Nachdem die Variable zu der Symbolinitialisierungstabelle hinzugefügt wurde, können Sie den gewünschten Init-Wert in die Tabelle eintragen, indem Sie den anfänglichen, automatischen Eintrag ändern. Die Variable nVar aus dem obigen Beispiel wird also zunächst mit dem Wert 123 zu der Tabelle hinzugefügt. Wenn Sie den Tabellenwert anschließend auf 456 ändern, bekommt die Variable nVar vor Beginn der Codeausführung den Wert 456 zugewiesen. Der Initialwert der Variablendeklaration (123) wird somit von dem Eintrag in der Tabelle (456) überschrieben.

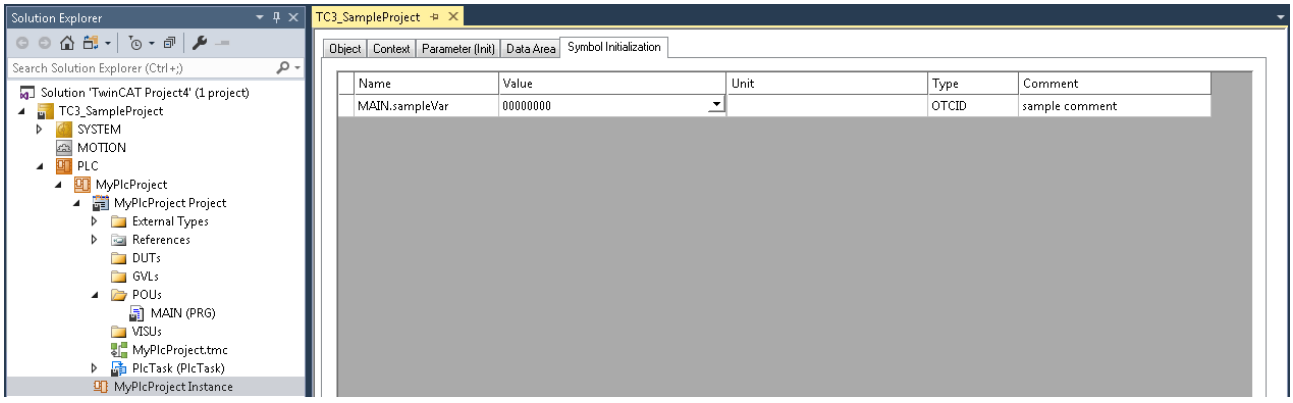
Befüllung der Tabellenspalte „Einheit“:

Die Spalte „Einheit“ wird für OTCID-Datentypen verwendet und wird für diese Datentypen automatisch mit dem Namen des TcCOM-Objekts befüllt, welches zu dem OTCID-Wert in der „Wert“-Spalte gehört.

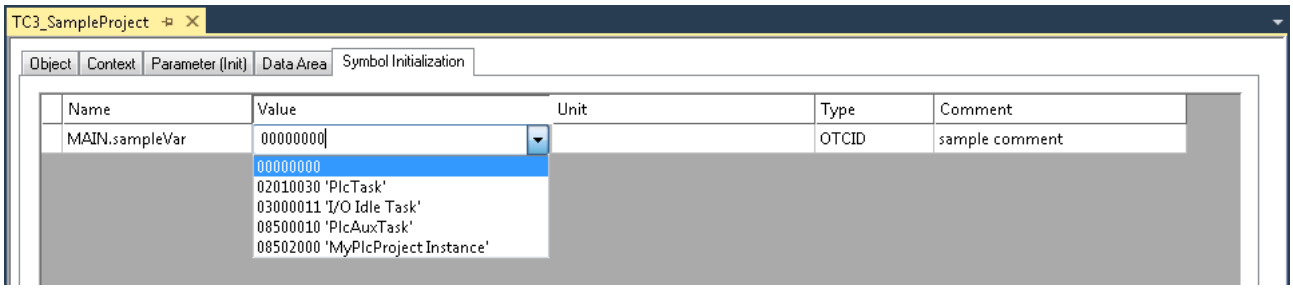
Beispiel:

```
PROGRAM MAIN
VAR
    {attribute 'TcInitSymbol'}
    sampleVar         : OTCID;        //sample comment
END_VAR
```

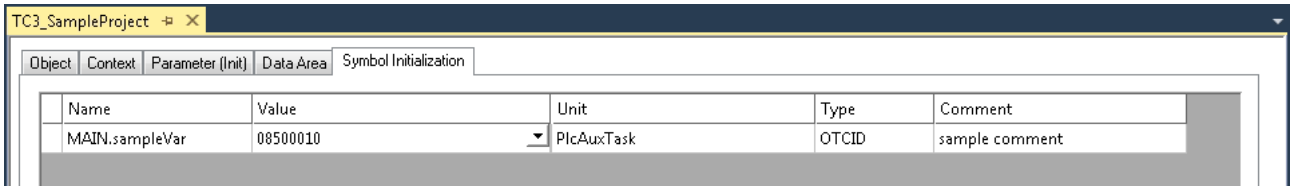
Diese Deklaration führt zu folgender Darstellung auf der Registerkarte **Symbol Initialisierung** der SPS-Instanz:



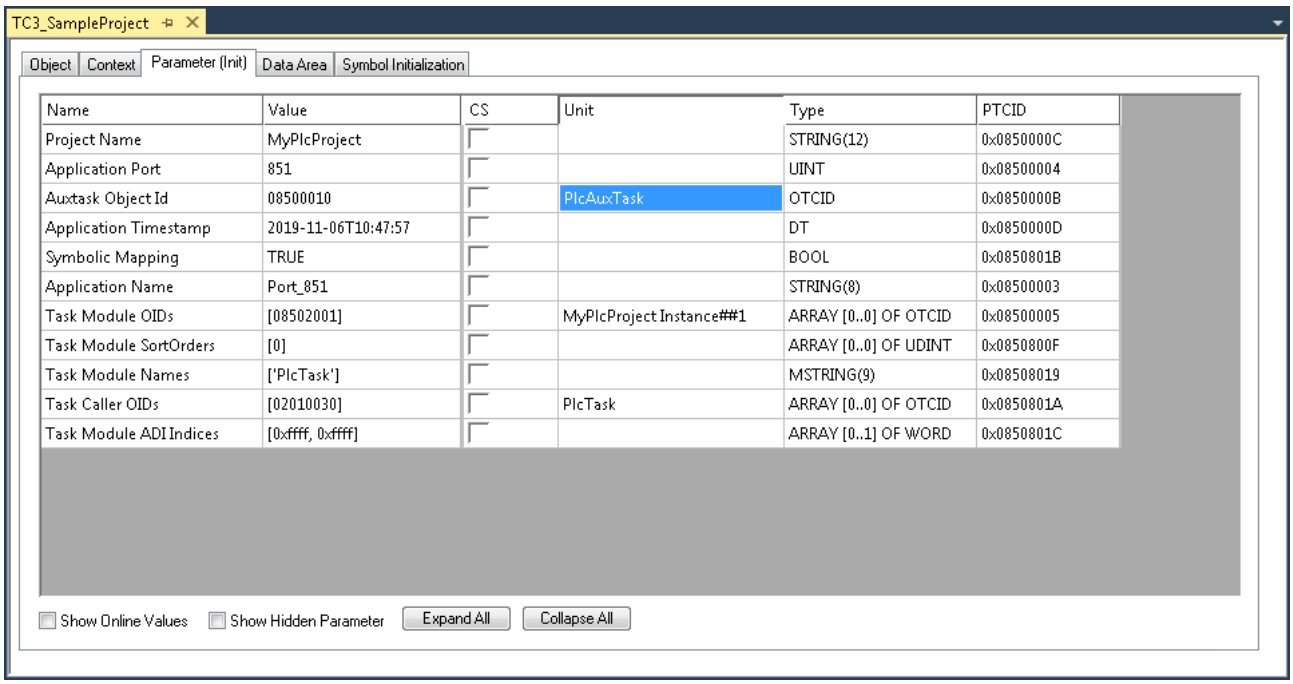
Um die Variable mit einem Wert zu versehen, kann ein Auswahlmenü geöffnet werden, welches die möglichen Werte zur Auswahl anbietet.



Wenn in diesem Fall beispielsweise das Wertepaar „08500010 'PlcAuxTask'“ ausgewählt wird, wird die ID (08500010) in die Spalte „Wert“ und der Name (PlcAuxTask) in die Spalte „Einheit“ eingetragen.



Das Wertepaar „Wert“ und „Einheit“ entspricht dem zugehörigen Wertepaar, welches auf der Registerkarte **Parameter (Init)** dargestellt ist.



16.8.2.53 Attribut 'TcLinkTo' / 'TcLinkToOSO'

Die beiden Pragmas können angewendet werden, um Variablen direkt Ein- und Ausgängen anderer Prozessabbilder zuzuweisen.

Syntax:

{attribute 'TcLinkTo' := '<I/O point name>'}

Der in einfache Hochkommata eingeschlossene Platzhalter <I/O point name> muss durch den Namen des Ein- oder Ausgangs ersetzt werden.

{attribute 'TcLinkToOSO' := '<x,y,z>'I/O point name'}

x: Bit-Offset der SPS-Variablen

y: Anzahl der anzuschließenden Bits

z: Offset der Zielvariablen, auf denen diese Bits abgebildet werden sollen

Der in einfache Hochkommata eingeschlossene Platzhalter <I/O point name> muss durch den Namen des Ein- oder Ausgangs ersetzt werden.

Verlinkung allokierter Variablen eines Funktionsbausteins:

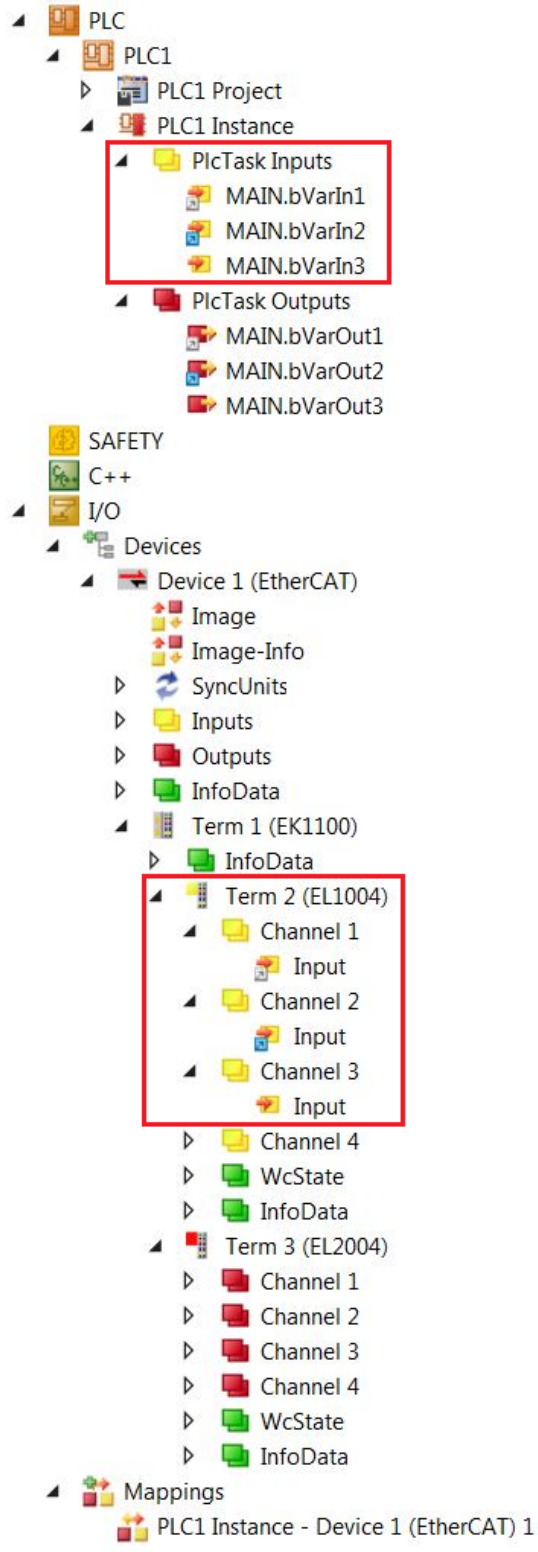
Allokierte Variablen eines Funktionsbausteins können ebenfalls mit Hilfe des Pragmas verknüpft werden. Dies geschieht für Funktionsbausteine nicht an der Deklaration der allokierten Variablen, sondern an der Deklarationsstelle der Funktionsbausteininstanz. Weitere Informationen hierzu befinden sich unten im Abschnitt „Beispiele zur Verwendung des Attributs“.

Darstellung der Verknüpfungsart:

Wird eine Variable manuell mit einem Ein- oder Ausgang verknüpft, dann erhält die zugehörige Variable im Prozessabbild ein weißes Symbol, ebenso wie der verknüpfte Kanal. Wird die Variable hingegen über das Attribut 'TcLinkTo' / 'TcLinkToOSO' verknüpft, dann ist das zugehörige Symbol blau. Bei einer nicht verknüpften Variable ist kein Symbol vorhanden.

```
PROGRAM MAIN
VAR
    bVarIn1    AT%I*  : BOOL;  // linked manually

    {attribute 'TcLinkTo' := 'TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL1004)^Channel
2^Input'}
    bVarIn2    AT%I*  : BOOL;  // linked via attribute
```

```
    bVarIn3    AT%I*   : BOOL;  // not linked

    bVarOut1   AT%Q*   : BOOL;  // linked manually

    {attribute 'TcLinkTo' := 'TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2004)^Channel
2^Output'}
    bVarOut2   AT%Q*   : BOOL;  // linked via attribute

    bVarOut3   AT%Q*   : BOOL;  // not linked
END_VAR
```



Beispiele zur Verwendung des Attributs:

```
PROGRAM MAIN
VAR
    nVar1          : INT;

     {attribute 'TcLinkTo' := 'TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 5 (EL4132)^Channel1^Out
put'}
    // Link when using the full path of an input or output
    nVar2   AT%Q*  : INT;

    {attribute 'TcLinkTo' := '[1] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 4 (EL2008)^Channe
l 4^Output;
                              [2] := TIID^Device 1
(EtherCAT)^Term 1 (EK1100)^Term 4 (EL2008)^Channel 5^Output'}
    aVar3   AT%Q*  : ARRAY [1..2] OF BOOL;

    {attribute 'TcLinkTo' := '.bIn  := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL1008)^Chan
nel 2^Input;
                              .bOut := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)^Chan
nel 2^Output'}
    // Link when using function blocks (FB_Module with allocated input bIn and allocated output bOut
)
    fbVar4         : FB_Module;

    {attribute 'TcLinkTo' := '[1].bIn  := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL1008)^C
hannel 4^Input;
                              [1].bOut := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2008)^C
hannel 4^Output;
                              [2].bIn  := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL1008)^C
hannel 5^Input;
                              [2].bOut := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2008)^C
hannel 5^Output'}
    aModule         : ARRAY[1..2] OF FB_Module;

    {attribute 'TcLinkTo' := 'TIIB(2)^Channel 5^Output'}
    // Linking with the keyboard shortcuts
    bVar5   AT%Q*  : BOOL;

    {attribute 'TcLinkTo' := 'TIIB[TerminalXY]^Channel 5^Output'}
    // TerminalXY is the name of the terminal shown in the IO tree
    nVar6   AT%Q*  : INT;

    {attribute 'TcLinkToOSO' := '<1,2,3>TIIB(5)^Channel 2^Output'}
    // Link when using Offset/Size/Offset
    nVar7   AT%Q*  : BYTE;

    {attribute 'TcLinkToOSO' := '[1] := <1,2,3>TIIB(5)^Channel 2^Output;
                                 [2] := <4,5,6>TIIB(5)^Channel 3^Output'}
    aVar8   AT%Q*  : ARRAY [1..2] OF WORD;

    {attribute 'TcLinkTo' := '[0]
[0] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)^InfoData^State;
                              [0]
[1] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 3 (EL2008)^InfoData^State;
                              [0]
[2] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 4 (EL2008)^InfoData^State;
                              [1]
[0] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 5 (EL2008)^InfoData^State;
                              [1]
[1] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 6 (EL2008)^InfoData^State;
                              [1]
[2] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 7 (EL2008)^InfoData^State'}
    aVar9   AT%Q*  : ARRAY[0..1, 0..2] OF UINT;


    (*************************************************************************************)
      Available from TC3.1 Build 4026
    (*************************************************************************************)

    // ARRAY info on the left is "%d..%d" and on the right side "%d..#"
    // Link info will be generate from 1 TO 4 (4 times) and from 3 TO 6 on channel side
    {attribute 'TcLinkTo' := '[1..4] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 2 (EL2008)^Cha
nnel 3..#^Output'}
    bVar10 AT %Q* : ARRAY[1..4] OF BOOL;

    {attribute 'TcLinkTo' := '[1..4] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 4 (EL4014)^AO
Outputs Channel 1..#^Analog output'}
    bVar11 AT %Q* : ARRAY[1..4] OF INT;

    // Also additional syntax for EtherCAT PLC structures to PLC mappings -
 see "PLC" tab on EtherCAT terminals:
```

```
    // ARRAY info on the left is "%d..%d" and on the right side "%d..#" -
 channel index is added via "^^%d" and the value is 0-based
    {attribute 'TcLinkTo' := '[1..4] := TIID^Device 1 (EtherCAT)^Term 1 (EK1100)^Term 5 (EL2819)^^ 0
..#'}
    bVar12 : ARRAY [1..16] OF MDP5001_280_700116F7;

END_VAR
```

Tabelle aller verfügbaren Tastenkombinationen:

Tastenkombination	To
TIIC	Knoten I/O-Konfiguration
TIID	Knoten I/O Konfiguration^I/O Geräte oder Knoten I/O Konfiguration TAB "I/O Geräte"
TIRC	Knoten Echtzeit-Konfiguration
TIRT	Knoten Echtzeit-Konfiguration^Zusätzliche Tasks oder Knoten "Echtzeit-Konfiguration TAB "Zusätzliche Tasks"
TIRS	Knoten Echtzeit-Konfiguration^Echtzeit-Einstellungen oder Knoten "Echtzeit-Konfiguration TAB "Echtzeit-Einstellungen"
TIPC	Knoten SPS-Konfiguration
TINC	Knoten NC-Konfiguration
TICC	Knoten CNC-Konfiguration
TIAC	Knoten CAM-Konfiguration
TING	Knoten Achsen
TINT	Knoten Tabellen
TINS	Knoten SAF Task
TIIT(%d)	Klemme %d (nur für KL Bus)
TIIB(%d)	Box %d
TIIF(%d)	Gerät %d
TIIT[%s]	Klemme mit Namen %s (nur für KL Bus)
TIIB[%s]	Box mit Namen %s
TIIF[%s]	Gerät mit Namen %s

16.8.2.54 Attribut 'TcNcAxis'

Das Pragma kann angewendet werden, um NC-Achsen außerhalb des Quellcodes, Variablen des Typs Axis_Ref zuzuordnen.

Syntax: {attribute 'TcNcAxis' := '<AxisName>'}

Der in einfache Hochkommata eingeschlossene Platzhalter <AxisName> muss durch den Namen der Achse, wie im Teil NC beschrieben, ersetzt werden.

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

Beispiele:

```
PROGRAM MAIN
VAR
    {attribute 'TcNcAxis' := 'Axis1'}
    // Using the axis in the same POU of type Program.
    axis1     : AXIS_REF;

    {attribute 'TcNcAxis' := '.axis1:=Axis2;.axis2:=Axis3'}
    // Use of the axis in POU fbSample of type Function block.
    // The internal axis names (axis1 and axis2) of the POU fbSample must be assigned to the NC axes
 used by the function block instance (axis 2 and axis 3).
    fbSample  : FB_Sample;

    {attribute 'TcNcAxis' := '[0]:=Axis4;[1]:=Axis5;[2]:=Axis6'}
    // Using an axis in an array.
    aAxis     : ARRAY [0..2] OF AXIS_REF;
END_VAR
```

16.8.2.55 Attribut 'TcNoSymbol' / 'tc_no_symbol'

Das Pragma kann angewendet werden, um festzulegen, dass für eine Variable kein (ADS-) Symbol erzeugt wird. Es kann also nicht symbolisch darauf zugegriffen werden. Wird dem Pragma der Wert „unused“ zugewiesen, wirkt das Pragma nur für Variablen die nicht verwendet werden.

Beachten Sie, dass Variablen, die mit dem Attribut 'TcNoSymbol'/'tc_no_symbol' deklariert sind, nicht als persistent abgespeichert werden können. Des Weiteren wird für Variablen, für die kein (ADS-) Symbol erzeugt wird, die Erzeugung des zugehörigen Prozessabbilds (allokierte Inputs/Outputs) verhindert.

Syntax: {attribute 'tc_no_symbol'} bzw. {attribute 'TcNoSymbol'}
(Die Schreibweise ‘TcNoSymbol‘ ist erst mit der TwinCAT-Version 3.1.4022 verwendbar.)

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

Beispiele:

```
VAR_GLOBAL
    {attribute 'tc_no_symbol'}
    nVar1  : INT;
    {attribute 'tc_no_symbol'}
    var2   : OTCID;
END_VAR
```

Beziehungsweise für die neue Schreibweise seit der TwinCAT-Version 3.1.4022:

```
VAR_GLOBAL
    {attribute 'TcNoSymbol'}
    nVar1  : INT;
    {attribute 'TcNoSymbol':= 'unused'}
    var2   : OTCID;
END_VAR
```

16.8.2.56 Attribut 'TcRpcEnable'

Das Pragma kann verwendet werden, um eine Methode für einen Remote-Methodenaufruf zu aktivieren (RPC = Remote Procedure Call). Dadurch kann die Methode per ADS aufgerufen werden. Der TwinCAT OPC UA Server benötigt dieses Pragma, um die Methode auch als OPC UA Methode zur Verfügung zu stellen.

Syntax: {attribute 'TcRpcEnable'}

Einfügeort: Erste Zeile über dem Deklarationsteil der Methode.

Beispiel:

```
{attribute 'TcRpcEnable'}
METHOD M_Sum : INT
VAR_INPUT
    nInput1 : INT;
    nInput2 : INT;
END_VAR
```

Weitere Beispiele:

- ADS:
  - Link: TcAdsClient.InvokeRpcMethod Method (ITcAdsSymbol, Int32, .Object.)
  - Dokupfad: TwinCAT 3 \ TE1000 XAE \ Technologien \ ADS \ TwinCAT ADS .NET \ TwinCAT.Ads Namespaces \ TwinCAT.Ads Namespace \ TcAdsClient Class \ TcAdsClient Methods \ TcAdsClient.InvokeRpcMethod Method \ TcAdsClient.InvokeRpcMethod Method (ITcAdsSymbol, Int32, .Object.)
- OPC UA:
  - Link: Methodenaufruf
  - Dokupfad: TwinCAT 3 \ TFxxxx | TC3 Functions \ TF6xxx – Connectivity \ TF6100 TC3 OPC UA \ Technische Einführung \ Server \ SPS \ Methodenaufruf

16.8.2.57 Attribut 'TcRetain'

Dieses Pragma bewirkt, dass die folgenden Variablen als Retain-Variablen angelegt werden und nach einem unkontrollierten Beenden (Spannungsausfall) ihren Wert behalten. Das Pragma stellt damit eine Alternative zur Nutzung des Schlüsselworts VAR RETAIN [▶ 727] dar.

Der sogenannte Retain-Handler sorgt dafür, dass die Retain-Variablen am Ende eines SPS-Zyklus und nur bei Änderung in den entsprechenden Bereich des NovRams geschrieben werden. Der Umgang mit dem Retain-Handler ist in der Dokumentation C/C++ im Kapitel „Retain Daten“ beschrieben.

Wenn selbstangelegte Datentypen (DUTs) als Retain-Variablen verwendet werden sollen, müssen die Datentypen im TwinCAT Typsystem vorhanden sein. Hierfür kann entweder die Option **Convert to Global Type** verwendet werden oder Strukturen können direkt als STRUCT RETAIN angelegt werden, womit dann jedoch alle Vorkommen der Struktur über den Retain Handler behandelt werden.

Für POUs (Funktionsbausteine) als Ganzes ist die Verwendung von Retain-Daten nicht möglich. Einzelne Elemente eines POUs können hingegen verwendet werden.

Syntax: {attribute 'TcRetain'}

Einfügeort: Zeile oberhalb der Deklarationszeile einer Variablen

Beispiel:

```
PROGRAM MAIN
VAR
    {attribute 'TcRetain'}
    nVar1   : INT;
END_VAR
```

16.8.2.58 Attribut 'TcSwapDWord'

Über das Pragma können Sie definieren, dass für diese allokierte Ausgangsvariable die Word-Reihenfolge geändert werden soll (Tausch zwischen Low-Word und High-Word). Wenn das Pragma vorhanden ist, wird für den Ausgang innerhalb des SPS-Prozessabbilds die Checkbox „Swap LOWORD und HIWORD“ aktiviert.

Syntax: {attribute 'TcSwapDWord'}

Einfügeort: Zeile oberhalb der Deklarationszeile einer allokierten Ausgangsvariablen (AT%Q)

Beispiele:

```
VAR
    {attribute 'TcSwapWord'}
    nVar1  AT%Q* : WORD;
    {attribute 'TcSwapDWord'}
    nVar2  AT%Q* : DWORD;
END_VAR
```

16.8.2.59 Attribut 'TcSwapWord'

Über das Pragma können Sie definieren, dass für diese allokierte Ausgangsvariable die Byte-Reihenfolge geändert werden soll (Tausch zwischen Low-Byte und High-Byte). Wenn das Pragma vorhanden ist, wird für den Ausgang innerhalb des SPS-Prozessabbilds die Checkbox „Swap LOBYTE und HIBYTE“ aktiviert.

Syntax: {attribute 'TcSwapWord'}

Einfügeort: Zeile oberhalb der Deklarationszeile einer allokierten Ausgangsvariablen (AT%Q)

Beispiele:

```
VAR
    {attribute 'TcSwapWord'}
    nVar1  AT%Q* : WORD;
    {attribute 'TcSwapDWord'}
    nVar2  AT%Q* : DWORD;
END_VAR
```

16.8.2.60 Attribut 'to_string'

Das Pragma wirkt sich darauf aus, wie das Ergebnis der Konvertierung einer Enumerationskomponente mit dem Operator TO_STRING/TO_WSTRING [▶ 762] ausgegeben wird: Wenn die Enumerationsdeklaration mit dem Pragma versehen ist, erscheint anstelle des numerischen Werts der Name der Enumerationskomponente als Zeichenfolge.

Syntax: {attribute 'to_string'}

Einfügeort: Zeile oberhalb der Deklaration der Enumeration.

Verfügbar ab TC3.1 Build 4024

Beispiel:

Enumeration E_Sample

```
{attribute 'qualified_only'}
{attribute 'strict'}
{attribute 'to_string'}
TYPE E_Sample :
(
    eInit := 0,
    eStart,
    eStop
);
END_TYPE
```

Programm MAIN

```
PROGRAM MAIN
VAR
    eSample         : E_Sample;
    nCurrentValue   : INT;
    sCurrentValue   : STRING;
    wsCurrentValue  : WSTRING;

    sComponent      : STRING;
    wsComponent     : WSTRING;
END_VAR
```

```
nCurrentValue  := eSample;
sCurrentValue  := TO_STRING(eSample);
wsCurrentValue := TO_WSTRING(eSample);

sComponent     := TO_STRING(E_Sample.eStart);
wsComponent    := TO_WSTRING(E_Sample.eStop);
```

Ergebnis der Zuweisungen/Konvertierungsfunktionen:

- Wert von nCurrentValue: 0
- Wert von sCurrentValue: 'eInit'
- Wert von wsCurrentValue: "eInit"
- Wert von sComponent: 'eStart'
- Wert von wsComponent: "eStop"

Ergebnis, falls die Enumeration nicht mit dem Attribut 'to_string' deklariert wäre:

- Wert von nCurrentValue: 0
- Wert von sCurrentValue: '0'
- Wert von wsCurrentValue: "0"
- Wert von sComponent: '1'
- Wert von wsComponent: "2"

Siehe auch:

- Aufzählungen / Enumerationen [▶ 816]

16.8.3 Bedingte Pragmas

Bedingte Pragmas dienen dazu, die Codegenerierung im Vorübersetzungsprozess oder Übersetzungsprozess zu beeinflussen. Die Programmiersprache ST unterstützt diese Pragmas.

> Ab TC 3.1 Build 4024 können Sie die bedingten Pragmas sowohl im Deklarations- als auch im Implementierungsteil verwenden. Dabei werden im Deklarationsteil ausschließlich die Operatoren „defined (<identifier>)“ und „hasvalue (<identifier>, ‘<value>‘)“ für einfache Compilerdefinitionen unterstützt. Die unten genannten Sonderverwendungen der beiden Operatoren sind davon ausgeschlossen. Zudem ist es nicht möglich, die Komponenten einer Enumeration oder den Datentyp eines Alias mit Hilfe von bedingten Pragmas zu definieren.
>
> Bedingte Pragamas werden im SPS Projekt nicht aber in Bibliotheken unterstützt. Bei vorherigen TC 3.1-Versionen werden die Pragmas, die im Deklarationsteil verwendet werden, nicht ausgewertet.

Mit bedingten Pragmas beeinflussen Sie, ob Deklarations- oder Implementierungscode für die Übersetzung berücksichtigt wird. Dies können Sie beispielsweise davon abhängig machen, ob eine bestimmte Compilerdefinition definiert ist oder einen bestimmten Wert hat, ob eine bestimmte Variable deklariert ist, oder ob ein bestimmter Baustein vorhanden ist etc..

Pragma	Beschreibung
Definition im Code: • {define <identifier>} • {define <identifier> '<value>'} Beispiele: • {define variantA} • {define variantA '123'} • {define variantA 'true'} **Definition in den SPS-Projekteigenschaften:** • <identifier> • <identifier> := '<value>' Beispiele: • variantA • variantA := '123' • variantA := 'true' • variantA := '123', variantB := '456'	Die Definition kann später mit dem defined-Operator abgefragt werden. Außerdem kann der optional angegebene Wert später mit dem hasvalue-Operator abgefragt und verglichen werden.
{undefine <identifier>}	Die {define}- Anweisung des Bezeichners <identifier> wird aufgehoben, der Bezeichner ist ab jetzt wieder „undefiniert“. Wenn der angegebene Bezeichner gerade gar nicht definiert ist, wird das Pragma ignoriert.
{IF <expr>}... {ELSIF <expr>}... {ELSE}... END_IF}	Dies sind Pragmas für die bedingte Kompilierung. Die angegebenen Ausdrücke <expr> müssen zur Kompilierungszeit konstant sein; sie werden in der Reihenfolge ausgewertet, in der sie hier erscheinen, bis einer der Ausdrücke einen nicht-Null-Wert zeigt. Der mit der Anweisung verknüpfte Text wird übersetzt; die anderen Zeilen werden ignoriert. Die Reihenfolge der Abschnitte ist festgelegt. Die ELSIF und ELSE Abschnitte sind optional. Die ELSIF-Abschnitte dürfen beliebig oft vorkommen. Innerhalb der Konstanten <expr> können Sie mehrere bedingte Übersetzungsoperatoren verwenden.
<expr>}	Innerhalb des konstanten Ausdrucks <expr> innerhalb eines bedingten Übersetzungspragmas {IF} oder {ELSIF} können Sie einen oder mehrere Operatoren verwenden.

Unterschiedliche Gültigkeitsbereiche

Globale define-Definitionen können Sie als Compilerdefinitionen in den SPS-Projekteigenschaften (Kategorie Übersetzen) oder auf System Manager-Level einen Kontenpunkt darüber (Gruppierung Compiler Defines, siehe Dokumentation Variantenmanagement) eintragen. Die hier definierten Compilerdefinitionen sind im gesamten SPS-Projekt gültig. Außerdem können Sie in den SPS-Projekteigenschaften mehrere define-Definitionen durch Kommas getrennt angeben.

Des Weiteren können Sie im Deklarations- und Implementierungsteil der SPS-Elemente lokale Compilerdefinitionen definieren. Diese sind dann im jeweiligen Deklarations- oder Implementierungseditor gültig.

Siehe auch:

- Strukturierter Text (ST), Erweiterter Strukturierter Text (ExST) [▶ 128]
- Variantenmanagement

Operator defined (<identifier>)

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält. Voraussetzung ist, dass der Bezeichner <identifier> mithilfe einer {define}-Anweisung definiert wurde und danach nicht mit einer {undefine}-Anweisung wieder undefiniert wurde; ansonsten wird FALSE geliefert.

Beispiel 1:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Variable pdef1 ist durch eine {define}-Anweisung in Plc1 definiert, aber nicht in Plc2.

```
{IF defined (pdef1)}
    (* This code is processed in Plc1 *)
    {info 'pdef1 defined'}
    hugo := hugo + SINT#1;
{ELSE}
    (* the following code is only processed in Plc2 *)
    {info 'pdef1 not defined'}
    hugo := hugo - SINT#1;
{END_IF}
```

Hier ist zusätzlich ein Beispiel eines Meldungspragmas enthalten: Nur die Information pdef1 defined wird im Meldungsfenster angezeigt, wenn das SPS-Projekt kompiliert wird, weil pdef1 tatsächlich definiert ist. Die Meldung 'pdef1 not defined' wird ausgegeben, wenn pdef1 nicht definiert ist.

Beispiel 2:

Im folgenden Beispiel wird die Variable sVariantUsed je nach geltendem Compiler-define mit unterschiedlichen Werten initialisiert. Außerdem wird entweder eine allokierte Eingangs- oder eine Ausgangsvariable deklariert. Die Variable nCounter wird in jedem Fall deklariert, d.h. unabhängig von den geltenden Compilerdefinitionen.

```
{IF defined (Variant1)}
    sVariantUsed         : STRING := 'Variant1';
    bOutput        AT%Q* : BOOL;
{ELSE}
    sVariantUsed         : STRING := 'NotVariant1';
    bInput         AT%I* : BOOL;
{END_IF}

nCounter                 : INT;
```

Operatoren für den Implementierungsteil

Die im folgenden vorgestellten Operatoren können ausschließlich im Implementierungsteil verwendet werden und werden im Deklarationsteil nicht ausgewertet.

Operator defined (variable: <variable>)

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn die Variable <variable> innerhalb des aktuellen Gültigkeitsbereichs deklariert ist; ansonsten wird FALSE geliefert.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Variable g_bTest ist in Plc1 deklariert, nicht aber in Plc2.

```
{IF defined (variable: g_bTest)}
    (* the following code is only processed in Plc2*)
    g_bTest := x > 300;
{END_IF}
```

Operator defined (type: <identifier>)

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn ein Datentyp mit Bezeichner <identifier> deklariert ist; ansonsten wird FALSE geliefert.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Datentyp DUT ist in Plc1 deklariert, nicht aber Plc2.

```
{IF defined (type: DUT)}
    (* the following code is only processed in Plc1*)
    bDutDefined := TRUE;
{END_IF}
```

Operator defined (pou: <pou name>)

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn ein Baustein mit dem Namen <pou name> vorhanden ist, ansonsten wird FALSE geliefert.

Mit Hilfe der Syntax "pou: <pou name>.<method name>" können Sie außerdem überprüfen, ob der genannte Baustein über Methoden oder Aktionen mit dem genannten Namen verfügt.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Baustein CheckBounds ist in Plc1 vorhanden, nicht aber in Plc2.

```
{IF defined (pou: CheckBounds)}
    (* the following code is only processed in Plc1 *)
    arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
{ELSE}
    (* the following code is only processed in Plc2 *)
    arrTest[i] := arrTest[i]+1;
{END_IF}
```

Operator defined (IsLittleEndian)

Der Operator bewirkt, dass der Ausdruck den Wert FALSE erhält, wenn die CPU „BigEndian (Motorola Byte Order)“ ist.

Operator defined (IsFPUSupported)

Wenn der Ausdruck den Wert TRUE liefert, erzeugt der Codegenerator FPU (Floating point unit) Code für die Berechnungen mit REAL-Werten. Ansonsten emuliert TwinCAT FPU-Operationen, was jedoch bedeutend langsamer ist.

Operator hasvalue (RegisterSize, '<register size>')

<register size>: Größe eines CPU-Registers in Bit

Der Operator bewirkt, dass der Ausdruck den Wert TRUE liefert, wenn die Größe eines CPU-Registers gleich <register size> ist.

Mögliche Werte für <register size>

- 16 für C16x,
- 64 für X86-64 Bit
- 32 für X86

Operator hasvalue (PackMode, '<pack mode value>')

Der abgeprüfte PackMode hängt von der Gerätebeschreibung ab, nicht vom Pragma, das für einzelne DUTs angegeben werden kann.

Operator hasattribute (pou: <pou name>, '<attribute>')

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn das Attribut <attribute> in der ersten Zeile des Deklarationsteils des Bausteins pou-name angegeben ist; ansonsten wird FALSE geliefert.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Eine Funktion fun1 ist in Plc1 und Plc2 definiert, aber in Plc1 ist ihr außerdem das Attribut vision zugewiesen.

In Plc1:

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
  i : INT;
END_VAR
VAR
END_VAR
```

In Plc2:

```
FUNCTION fun1 : INT
VAR_INPUT
  i : INT;
END_VAR
VAR
END_VAR
```

Pragma-Anweisung:

```
{IF hasattribute (pou: fun1, 'vision')}
(* the following code is only processed in Plc1 *)
ergvar := fun1(ivar);
{END_IF}
```

Siehe auch:

- Benutzerdefinierte Attribute [▶ 830]

Operator hasattribute (variable: <variable>, '<attribute>')

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn der Variablen das Attribut '<attribute>' mithilfe der Anweisung {attribute '<attribute>'} in der Zeile vor der Variablendeklaration zugewiesen ist; ansonsten wird FALSE geliefert

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Variable g_globalInt wird in Plc1 und Plc2 verwendet, aber in Plc1 ist ihr zusätzlich das Attribut 'DoCount' zugewiesen.

Deklaration g_GlobalInt in Plc1

```
VAR_GLOBAL
   {attribute 'DoCount'}
  g_globalInt : INT;
  g_multiType : STRING;
END_VAR
```

Deklaration g_GlobalInt in Plc2

```
VAR_GLOBAL
  g_globalInt : INT;
  g_multiType : STRING;
END_VAR
```

Pragmaanweisung:

```
{IF hasattribute (variable: g_globalInt, 'DoCount')}
    (* the following code is only processed in Plc1 *)
    g_globalInt := g_globalInt + 1;
{END_IF}
```

Siehe auch:

- Benutzerdefinierte Attribute [▶ 830]

Operator hastype (variable: <variable>, <type-spec>)

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn die Variable <variable> vom Datentyp <type-spec> ist; ansonsten wird FALSE geliefert.

Mögliche Datentypen für <type-spec>:

BOOL | BYTE | DATE | DATE_AND_TIME | DINT | DWORD | INT | LDATE | LDATE_AND_TIME | LINT | LREAL | LTIME | LTIME_OF_DAY | LWORD | REAL | SINT | STRING | TIME | TIME_OF_DAY | ULINT | UDINT | UINT | USINT | WORD | WSTRING

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Variable g_multitype ist in Plc1 mit Datentyp LREAL deklariert, in Plc2 mit Datentyp STRING.

```
{IF (hastype (variable: g_multitype, LREAL))}
    (*  the following code is only processed in Plc1 *)
    g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
    (* the following code is only processed in Plc2 *)
    g_multitype := 'this is a multitalent';
{END_IF}
```

Operator hasvalue (<define-ident>, '<char-string>')

Der Operator bewirkt, dass der Ausdruck den Wert TRUE erhält, wenn eine Variable mit Bezeichner <define-ident> definiert ist und den Wert <char-string> hat; ansonsten wird FALSE geliefert.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. Die Variable test wird in den SPS-Projekten Plc1 und Plc2 verwendet; in Plc1 erhält sie den Wert 1, in Plc2 den Wert 2.

Die Compiler-Definition kann entweder in den SPS-Projekteigenschaften über `test := '1'` oder im Implementierungsteil der POU über `{define test '1'}` gesetzt werden.

```
{IF hasvalue(test,'1')}
    (* the following code is only processed in Plc1 *)
    x := x + 1;
{ELSIF hasvalue(test,'2')}
    (* the following code is only processed in Plc2 *)
    x := x + 2;
{END_IF}
```

Operator NOT <operator>

Der Ausdruck erhält den Wert TRUE, wenn der Umkehrwert von <operator> den Wert TRUE liefert. <operator> kann einer der in diesem Kapitel beschriebenen Operatoren sein.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. PLC_PRG1 ist in Plc1 und Plc2 vorhanden, POU CheckBounds gibt es nur in Plc1.

```
{IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))}
    (* the following code is only processed in Plc2 *)
    bANDNotTest := TRUE;
{END_IF}
```

Operator <operator> AND <operator>

Der Ausdruck erhält den Wert TRUE, wenn die beiden angegebenen Operatoren TRUE liefern. <operator> kann einer der in diesem Kapitel beschriebenen Operatoren sein.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. PLC_PRG1 ist in Plc1 und Plc2 vorhanden, die POU CheckBounds nur in Plc1.

```
{IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))}
    (* the following code is only processed in Plc1,  *)
    bANDTest := TRUE;
{END_IF}
```

Operator <operator> OR <operator>

Der Ausdruck liefert TRUE, wenn einer der beiden angegebenen Operatoren TRUE liefert. <operator> kann einer der hier beschriebenen Operatoren sein.

Beispiel:

Voraussetzung: Es gibt zwei SPS-Projekte Plc1 und Plc2. PLC_PRG1 ist in Plc1 und Plc2 vorhanden, POU CheckBounds nur in Plc1.

```
{IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))}
    (* the following code is only processed in Plc1 and in Plc2 *)
    bORTest := TRUE;
{END_IF}
```

Operator (<operator>)

() klammert die Operatoren.

Siehe auch:

- Pragmas verwenden [▶ 143]
- Benutzerdefinierte Attribute [▶ 830]

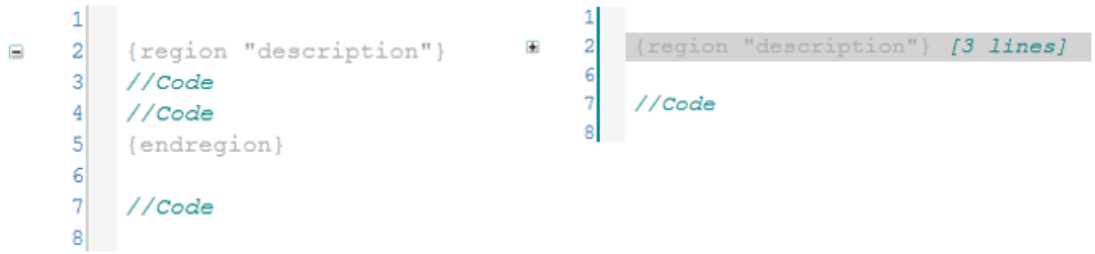
16.8.4 Region-Pragma

Das Pragma dient dazu, in einem Texteditor mehrere Zeilen zu einem Block zusammenzufassen. Der Block kann mit einem Namen versehen werden. Region-Pragmas können auch ineinander verschachtelt sein.

Syntax: {region "description"} … {endregion}

Achten Sie darauf, diese Syntax einzuhalten, damit das Pragma berücksichtigt wird.

Code mit Region-Pragma: Erweiterte und reduzierte Ansicht



Das Pragma kann im ST-Editor und allen Deklarationseditoren verwendet werden. Die Syntaxhervorhebung kann in den Optionen angepasst werden.

16.8.5 Pragmas zur Warnungsunterdrückung

16.8.5.1 Pragmas 'warning disable', 'warning restore'

Das Pragma {warning disable <compiler ID>} bewirkt, dass bestimmte Warnungen unterdrückt werden.

Das Pragma {warning restore <compiler ID>} bewirkt, dass eine unterdrückte Meldung wieder aktiviert wird.

Syntax:

{warning disable <compiler ID>}

{warning restore <compiler ID>}

<compiler ID>: ID, die am Anfang einer Warnungsmeldung bzw. in der Übersicht der Compiler-Warnungen in den SPS-Projekteigenschaften steht.

Beispiel:

Compiler-Warnung:

```
C0195: Implicit conversion from signed type 'SINT' to unsigned type 'UINT': possible change of sign
```

Das Pragma auf eine Variablendeklaration anwenden:

```
VAR
    {warning disable C0195}
    test1 : UINT := -1;
    {warning restore C0195}
    test2 : UINT := -1;
END_VAR
```

test1 erzeugt keine Fehlermeldung, test2 erzeugt eine Fehlermeldung.

Pragmas im Implementierungseditor

Wenn Sie die Pragmas zur Unterdrückung von Warnungen im Implementierungseditor verwenden möchten, ist dies aktuell im ST- sowie im FUP/KOP/AWL-Editor möglich.

In FUP/KOP/AWL muss das gewünschte Pragma in eine Sprungmarke [▶ 696] eingetragen werden.

Siehe auch:

- Dokumentation TC3 User Interface: Befehl Eigenschaften (SPS-Projekt) > Kategorie Compiler Warnings [▶ 960]

16.8.5.2 Attribut 'suppress_wrn_C0410'

Für die Compiler-Warnung mit der ID C0410 muss nicht das Pragma 'warning disable', sondern das Attribut 'suppress_wrn_C0410' verwendet werden.

Syntax: {attribute 'suppress_wrn_C0410'}

Beispiel:

Compiler-Warnung für eine Eigenschaft:

```
C0410: COMPATIBILITY WARNING: A write access to a Property of type REFERENCE calls the SET-Accessor
for versions < 3.1.4022.0 and writes the reference, but calls the GET-Accessor for versions >=
3.1.4022.0 and writes the value! Use the operator REF= if you want to assign the reference.
```

Das Attribut auf eine Eigenschaft anwenden:

```
{attribute 'suppress_wrn_C0410'}
PROPERTY refSample : REFERENCE TO BOOL
```

Aufgrund des Attributs wird für die Eigenschaft refSample die Warnung C0410 nicht erzeugt.

16.9 Bezeichner

Beachten Sie auch die TwinCAT-3-Programmmierkonventionen (Abschnitt „TwinCAT-3-Programmierkonventionen“)

Regeln für die Bezeichnervergabe

Regeln für den Bezeichner einer Variablen

- Ein Bezeichner darf folgende Buchstaben und Zahlen enthalten:
  - 0…9
  - A…Z
  - a…z
- TwinCAT beachtet die Groß-/Kleinschreibung nicht, das heißt: VAR1 und var1 bezeichnen dieselbe Variable.
- Ein Bezeichner darf nicht mit einer Zahl beginnen.
- Ein Bezeichner darf keine Leerzeichen, Sonderzeichen (!, &, ß, ...) oder Bindestriche enthalten.

- Die Verwendung von Unterstrichen ist hingegen erlaubt und TwinCAT erkennt diesen auch. Das bedeutet, dass zum Beispiel A_BCD und AB_CD als zwei verschiedene Bezeichner behandelt werden. In der Regel werden keine Trennzeichen, wie zum Beispiel der Unterstrich, zwischen den Wörtern verwendet.
- Doppelte Unterstriche ('__') dürfen nicht verwendet werden, da sie für interne Variablen genutzt werden.
- Die Länge eines Bezeichners ist unbegrenzt.

Regeln für die Mehrfachverwendung von Bezeichnern (Namensräume)

- Sie dürfen einen Bezeichner lokal nicht doppelt verwenden.
- Ein Bezeichner darf nicht identisch mit einem Schlüsselwort (zum Beispiel Variablentyp) sein.
- Sie dürfen einen Bezeichner global mehrfach verwenden. Wenn eine lokale Variable denselben Namen hat wie eine globale Variable, hat die lokale Variable innerhalb einer POU den Vorrang.
- Eine Variable, die in einer globalen Variablenliste definiert ist, kann denselben Namen haben wie eine Variable, die in einer anderen GVL definiert ist. TwinCAT bietet folgende norm-erweiternden Funktionalitäten bezüglich des Namensraums/Gültigkeitsbereichs von Variablen:
  - Globaler Namensraum-Operator: Ein Instanzpfad, der mit einem Punkt beginnt, öffnet immer einen globalen Namensraum. Wenn es eine lokale Variable, zum Beispiel nVar, gibt, die denselben Namen hat wie eine globale Variable, sprechen Sie mit .nVar die globale Variable an.
  - Der Name einer globalen Variablenliste kann den Namensraum für die enthaltenen Variablen eindeutig definieren. Somit können Sie Variablen mit demselben Namen in verschiedenen globalen Variablenlisten deklarieren und dennoch durch das Voranstellen des Listennamens eindeutig ansprechen.
    Beispiel:
    ```
    GVL1.nVar := GVL2.nVar; (*nVar aus GVL2 wird auf nVar in GVL1 kopiert*)
    ```
  - Variablen, die in der globalen Variablenliste einer ins Projekt eingebundenen Bibliothek definiert sind, können Sie gemäß der Syntax <Namensraum Bibliothek>.<Name der GVL>.<Variablenname> eindeutig ansprechen.
    Beispiel:
    ```
    GVL1.nVar := Lib1.GVL1.nVar (* nVar aus GVL1 in Bibliothek Lib1 wird auf
     nVar in GVL1 kopiert*)
    ```
- Für eine Bibliothek wird beim Einfügen via Bibliotheksverwalter auch ein Namensraum definiert. Somit können Sie einen Bibliotheksbaustein oder eine Bibliotheksvariable über <Namensraum Bibliothek>.<Bausteinname|Variablenname> eindeutig ansprechen. Beachten Sie im Fall von verschachtelten Bibliotheken, dass die Namensräume aller beteiligten Bibliotheken in Folge anzugeben sind.
  Beispiel: Wenn Lib1 durch Lib0 referenziert wird, so wird die in Lib1 enthaltene Funktion F_Sample über `Lib0.Lib1.F_Sample` angesprochen:
  ```
  nVar := Lib0.Lib1.F_Sample(nValue:=4); (*Rückgabewert von F_Sample wird auf
  Variable nVar im Projekt kopiert*)
  ```

Siehe auch:

- Variablen [▶ 718]
- Datentypen [▶ 793]

16.10 Verschattungsregeln

In TwinCAT ist es prinzipiell erlaubt, den gleichen Bezeichner für verschiedene Elemente zu verwenden. So können beispielsweise ein Baustein und eine Variable gleich benannt werden. Um Verwechslungen vorzubeugen, sollte dies jedoch vermieden werden.

Negativbeispiel:

Im folgenden Codeausschnitt hat eine lokale Funktionsbaustein-Instanz den gleichen Namen wie eine Funktion:

```
FUNCTION Sample : INT
```

```
FUNCTION_BLOCK FB_Sample
```

```
PROGRAM MAIN
VAR
    Sample : FB_Sample;
END_VAR
```

```
Sample();
```

In einem solchen Fall ist unklar, ob im Programm die Instanz oder die Funktion aufgerufen wird.

Namenskonventionen:

Um sicherzustellen, dass Namen immer eindeutig sind, sollten Namenskonventionen, beispielsweise für die Festlegung bestimmter Präfixe für Variablen, beachtet werden. Eine mögliche Definition solcher Präfixe finden Sie im Abschnitt Bezeichner/Namen der TwinCAT-3-Programmierkonventionen.

Namenskonventionen können mit Hilfe von TE1200 | PLC Static Analysis automatisch überprüft werden. Die statische Codeanalyse könnte durch die Prüfung der Regel SA0027 auch die doppelte Verwendung des Namens `Sample` aufdecken und als Fehler melden.

Qualifizierter Zugriff:

Auch durch die konsequente Verwendung des Attributs 'qualified_only' [▶ 859] für Enumerationen und globale Variablenlisten und durch die Verwendung von qualifizierten Bibliotheken können nicht-eindeutige Situationen vermieden werden.

Verschattung:

Der Compiler meldet grundsätzlich weder Fehler noch Warnungen, wenn derselbe Bezeichner für verschiedene Elemente verwendet wird. Stattdessen durchsucht der Compiler den Code in einer bestimmten Reihenfolge nach der Deklaration des Bezeichners. Wenn eine Deklaration gefunden wurde, dann sucht der Compiler nicht nach eventuellen weiteren Deklarationen an anderer Stelle. Wenn weitere Deklarationen existieren, dann sind diese für den Compiler „verschattet“. Im Folgenden werden die Verschattungsregeln beschrieben, das heißt die Suchreihenfolgen, die der Compiler bei der Suche nach der Deklaration für Bezeichner verwendet. Im Abschnitt „Uneindeutige Zugriffe und qualifizierte Zugriffe“ werden Möglichkeiten aufgezeigt, um uneindeutige Zugriffe zu vermeiden und die Verschattungsregeln zu umgehen.

Suchreihenfolge in der Applikation

Wenn der Compiler im Code einer Applikation auf einen einzelnen Bezeichner trifft, dann sucht er die zugehörige Deklaration in der folgenden Reihenfolge:

1. Lokale Variablen einer Methode
2. Lokale Variablen im Funktionsbaustein, Programm oder Funktion und in eventuellen Basisfunktionsbausteinen
3. Lokale Methoden des Bausteins
4. Globale Variablen im Projekt, wenn in der Variablenliste, in der die globalen Variablen deklariert sind, nicht das Attribut 'qualified_only' gesetzt ist.
5. Globale Variablen in angezogenen Bibliotheken, wenn weder die Bibliothek noch die Variablenliste qualifizierten Zugriff erfordert.
6. Baustein- oder Typnamen aus dem Projekt (das heißt: Namen von Globalen Variablenlisten, Funktionsbausteinen etc.)
7. Baustein- oder Typnamen aus einer Bibliothek
8. Namensräume von lokal angezogenen Bibliotheken und Bibliotheken, die von Bibliotheken veröffentlicht werden

Suchreihenfolge in der Bibliothek

Wenn der Compiler im Code einer Bibliothek auf einen einzelnen Bezeichner trifft, dann sucht er die zugehörige Deklaration in der folgenden Reihenfolge:

1. Lokale Variablen einer Methode
2. Lokale Variablen im Funktionsbaustein, Programm oder Funktion und in eventuellen Basis-Funktionsbausteinen
3. Lokale Methoden des Bausteins

4. Globale Variablen in der lokalen Bibliothek, wenn die Variablenliste, in der die globalen Variablen deklariert sind, nicht das Attribut 'qualified_only' gesetzt hat.
5. Globale Variablen in angezogenen Bibliotheken, wenn weder die Bibliothek noch die Variablenliste qualifizierten Zugriff erfordert.
6. Baustein- oder Typnamen aus der lokalen Bibliothek (das heißt: Namen von globalen Variablenlisten, Funktionsbausteinen etc.)
7. Baustein- oder Typnamen aus einer angezogenen Bibliothek
8. Namensräume von lokal angezogenen Bibliotheken und Bibliotheken, die von lokal angezogenen Bibliotheken veröffentlicht werden.

Uneindeutige Zugriffe und qualifizierte Zugriffe

Trotz dieser Suchreihenfolgen kann es zu uneindeutigen Zugriffen kommen. Das ist zum Beispiel der Fall, wenn eine Variable mit dem gleichen Namen in zwei globalen Variablenlisten vorkommt, die nicht qualifizierten Zugriff erfordern. Einen solchen Fall meldet der Compiler als Fehler (zum Beispiel: nicht eindeutige Verwendung des Namens <Variable>).

Eine solche uneindeutige Verwendung lässt sich durch einen qualifizierten Zugriff, also beispielsweise durch den Zugriff über den Namen der globalen Variablenliste, eindeutig machen (zum Beispiel: `GVL.<Variable>`).

Ein qualifizierter Zugriff lässt sich auch immer dazu nutzen, um Verschattungsregeln zu umgehen.

- Mit dem Namen der globalen Variablenliste kann eindeutig auf eine Variable in dieser Liste zugegriffen werden.
- Mit dem Namen einer Bibliothek kann eindeutig auf Elemente in dieser Bibliothek zugegriffen werden.
- Mit dem Pointer THIS [▶ 730] kann eindeutig auf Variablen in einem Funktionsbaustein zugegriffen werden, auch wenn eine lokale Variable mit gleichem Namen in einer Methode des Funktionsbausteins existiert.

Um jederzeit die Deklarationsstelle eines Bezeichners zu finden, wählen Sie den Befehl Gehe zur Definition [▶ 920], der im Kontextmenü des Editorfensters zur Verfügung steht. Dies kann insbesondere dann hilfreich sein, wenn der Compiler eine scheinbar unverständliche Fehlermeldung produziert.

Suchen in Instanzpfaden

Die oben beschriebenen Suchreihenfolgen gelten nicht für Bezeichner, die in einem Instanzpfad als Komponente auftauchen, oder für Bezeichner, die als Eingänge in Aufrufen verwendet werden.

Für einen Zugriff der Art `yy.Komponente` hängt es von der Entität ab, die durch `yy` beschrieben wird, wo nach der Deklaration von `Komponente` gesucht wird.

- Wenn `yy` eine Variable mit strukturiertem Datentyp bezeichnet (also vom Typ STRUCT oder UNION), dann wird `Komponente` in dieser Reihenfolge gesucht:
  - Lokale Variablen des Funktionsbausteins
  - Lokale Variablen des Basis-Funktionsbausteins
  - Methoden des Funktionsbausteins
  - Methoden des Basis-Funktionsbausteins
- Wenn `yy` eine globale Variablenliste bezeichnet oder ein Programm, dann wird `Komponente` nur in dieser Liste gesucht.
- Wenn `yy` einen Namensraum einer Bibliothek bezeichnet, dann wird `Komponente` in dieser Bibliothek genauso gesucht, wie es im obigen Abschnitt „Suchreihenfolge in der Bibliothek“ beschrieben ist.

Erst in zweiter Instanz entscheidet der Compiler, ob der Zugriff auf das gefundene Element erlaubt ist, das heißt, ob die Variable möglicherweise nur lokal zugreifbar ist, oder ob eine Methode privat ist. Wenn der Zugriff nicht erlaubt ist, wird ein Fehler ausgegeben.

Siehe auch:

- Bezeichner [▶ 882]

- Attribut 'qualified_only' [▶ 859]
- THIS [▶ 730]

16.11 Schlüsselwörter

Sie müssen Schlüsselwörter, die beispielweise Gültigkeitsbereiche, Datentypen oder Operatoren bezeichnen, in allen Editoren in Großbuchstaben schreiben.

Schlüsselwörter können nicht als Variablennamen verwendet werden.

Beispiele:

ABS, ACOS, ADD, ADR, AND, ANDN, ARRAY, ASIN, AT, ATAN

BITADR, BOOL, BY, BYTE

CAL, CALC, CALCN, CASE, CONSTANT, COS

DATE, DINT, DIV, DO, DT, DWORD

ELSE, ELSIF, END_CASE, END_FOR, END_IF, END_REPEAT, END_STRUCT, END_TYPE, END_VAR, END_WHILE, EQ, EXIT, EXP, EXPT

FALSE, FOR, FUNCTION, FUNCTION_BLOCK

GE, GT

IF, INDEXOF, INT

JMP, JMPC, JMPCN

LD, LDN, LE, LINT, LN, LOG, LREAL, LT, LTIME, LWORD

MAX, METHOD, MIN, MOD, MOVE, MUL, MUX

NE, NOT

OF, OR, ORN

PARAMS, PERSISTENT, POINTER, PROGRAM

R, READ_ONLY, READ_WRITE, REAL, REFERENCE, REPEAT, RET, RETAIN, RETC, RETCN, RETURN, ROL, ROR

S, SEL, SHL, SHR, SIN, SINT, SIZEOF, SUPER, SQRT, ST, STN, STRING, STRUCT, SUPER, SUB

TAN, THEN, THIS, TIME, TO, TOD, TRUE, TRUNC, TYPE

UDINT, UINT, ULINT, UNTIL, USINT

VAR, VAR_CONFIG, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, VAR_OUTPUT, VAR_STAT, VAR_TEMP

WHILE, WORD, WSTRING

XOR, XORN

TwinCAT überprüft die korrekte Verwendung von Schlüsselwörtern automatisch und unterkringelt Fehler unmittelbar bei der Eingabe.

Wenn TwinCAT impliziten Code anlegt, dann werden Variablen und Funktionen im Regelfall mit einem Namen versehen der „_“ enthält. Die Verwendung von doppelten Unterstrichen im Implementierungscode wird automatisch verhindert. Somit kann es nicht zu Konflikten zwischen systeminternen und den vom Programmierer vergebenen Bezeichnern kommen.

Folgende Schlüsselwörter werden im TwinCAT-Export-Format verwendet. Sie dürfen sie deshalb nicht als Bezeichner verwenden:

- ACTION
- END_ACTION
- END_FUNCTION
- END_FUNCTION_BLOCK
- END_PROGRAM

16.12 Methoden FB_init, FB_reinit und FB_exit

Die Methoden können Sie explizit einsetzen, um Einfluss auf die Initialisierung von Funktionsbausteinvariablen und auf das Verhalten beim Beenden von Funktionsbausteinen zu nehmen.

- FB_init [▶ 888]
- FB_reinit [▶ 892]
- FB_exit [▶ 892]

Sehen Sie auch die Information zu den verschiedenen Betriebsfällen [▶ 893] und dem Verhalten dieser Methoden bei abgeleiteten Bausteinen [▶ 898].

Der Typ des Rückgabewertes für die impliziten Methoden ist BOOL. Auch wenn der Wert nicht ausgewertet wird, sollte der Rückgabetyp nicht geändert werden.

Fügen Sie den Methoden keine Ausgangsvariablen hinzu. Sie können lediglich zusätzliche Eingangsvariablen definieren.

Expliziter Aufruf nicht empfohlen

Bei den Methoden FB_init, FB_reinit und FB_exit handelt es sich um Systemfunktionen, die zu unterschiedlichen Zeitpunkten implizit aufgerufen werden (weitere Informationen hierzu finden Sie unter Betriebsfälle [▶ 893]). Ein expliziter Aufruf dieser Methoden kann unbeabsichtigte Folgen haben und wird daher nicht empfohlen.

Automatisches Core Dump bei Exception in FB_init/FB_reinit/FB_exit

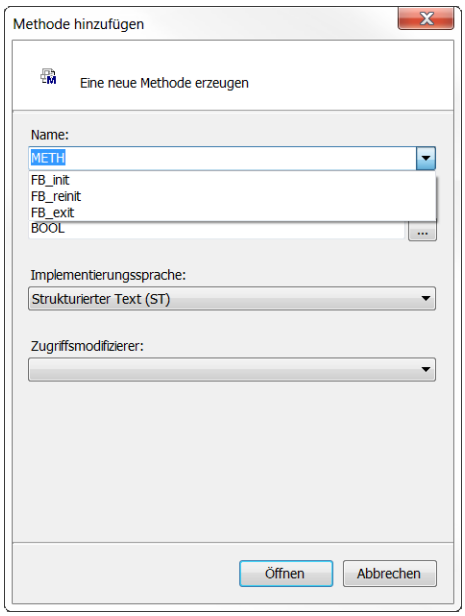
Falls innerhalb des Codes von FB_init/FB_reinit/FB_exit ein Ausnahmefehler auftritt, z.B. aufgrund eines Programmierfehlers, legt das Laufzeitsystem automatisch einen Core Dump auf dem Zielsystem ab (ab TC3.1 Build 4024.25). Dieser Core Dump wird als *.core-Datei im Boot-Ordner des Zielsystems abgelegt (standardmäßig unter *C:\TwinCAT\3.1\Boot\Plc*) und kann zur Ursachensuche verwendet werden.

Weitere Informationen zum Laden eines Core Dumps finden Sie unter: Fehleranalyse mit Core Dump [▶ 259]

Methoden hinzufügen

1. Selektieren Sie im **Projektmappen-Explorer** im SPS-Projektbaum einen Funktionsbaustein.
2. Wählen Sie im Kontextmenü den Befehl **Hinzufügen > Methode…**
   ⇨ Der Dialog **Methode hinzufügen** öffnet sich.
3. Öffnen Sie die Drop-down-Liste in dem Feld für die Benennung der Methode.
4. Wählen Sie eine der Methoden FB_init, FB_reinit oder FB_exit aus.
5. Klicken Sie auf **Öffnen**.

⇨ Die Methode wird zum SPS-Projektbaum hinzugefügt und im Editor geöffnet. Die Definition der Methode (Rückgabewert, Parameter) erfolgt automatisch.



Rückgabewert

Implizite Aufrufe

Bei impliziten Aufrufen der Methoden wird der Rückgabewert vom System nicht ausgewertet. Auch wenn Sie den Rückgabewert anpassen, wird dieser bei einem impliziten Aufruf nicht ausgewertet.

Explizite Aufrufe

Bei expliziten Aufrufen der Methoden können Sie den Rückgabewert auswerten. Dafür können Sie einen sinnvollen Rückgabewert zurückliefern.

Siehe auch:

- Objekt Methode [▶ 93]
- Attribut 'call_after_init' [▶ 831]
- Attribut 'call_after_online_change_slot' [▶ 832]
- Attribut 'call_on_type_change' [▶ 833]
- Attribut 'no_copy' [▶ 849]
- Attribut 'no-exit' [▶ 851]

16.12.1 FB_init

FB_init ist immer implizit verfügbar und wird grundsätzlich aufgerufen, um die Standardinitialisierung durchzuführen (impliziter Aufruf). Für eine gezielte Einflussnahme können Sie die Methode außerdem explizit deklarieren und zusätzlichen Code zum Standardinitialisierungscode ergänzen.

Wenn während der Ausführung der explizit definierte Initialisierungscode erreicht wird, ist die Funktionsbausteininstanz bereits über den impliziten Initialisierungscode vollständig initialisiert, sowohl bei automatischer Nullinitialisierung als auch bei individueller Werteinitialisierung. Es darf deshalb kein Aufruf `SUPER^.FB_init` erfolgen.

Debugging

Das Finden von Fehlern in den Methoden FB_init ist mühsam, weil unter anderem das Setzen von Haltepunkten nicht die gewünschte Wirkung haben kann.

Automatisches Core Dump bei Exception in FB_init/FB_reinit/FB_exit

Falls innerhalb des Codes von FB_init/FB_reinit/FB_exit ein Ausnahmefehler auftritt, z.B. aufgrund eines Programmierfehlers, legt das Laufzeitsystem automatisch einen Core Dump auf dem Zielsystem ab (ab TC3.1 Build 4024.25). Dieser Core Dump wird als *.core-Datei im Boot-Ordner des Zielsystems abgelegt (standardmäßig unter *C:\TwinCAT\3.1\Boot\Plc*) und kann zur Ursachensuche verwendet werden.

Weitere Informationen zum Laden eines Core Dumps finden Sie unter: Fehleranalyse mit Core Dump [▶ 259]

Initialisierungsreihenfolge beachten

Bitte beachten Sie, dass die Werte, die den Eingangsvariablen der Funktionsbausteininstanz zugewiesen werden, bei der Ausführung von FB_init noch nicht bekannt sind. Wenn Sie die Ausführung von FB_init parametrieren möchten, können Sie in der FB_init-Methode zusätzliche Methodeneingänge deklarieren.
Weiterführende Informationen finden Sie unter Betriebsfälle [▶ 893].

Expliziter Aufruf von FB_init

Expliziter Aufruf nicht empfohlen

Bei den Methoden FB_init, FB_reinit und FB_exit handelt es sich um Systemfunktionen, die zu unterschiedlichen Zeitpunkten implizit aufgerufen werden (weitere Informationen hierzu finden Sie unter Betriebsfälle [▶ 893]). Ein expliziter Aufruf dieser Methoden kann unbeabsichtigte Folgen haben und wird daher nicht empfohlen.

Die Methode FB_init kann prinzipiell auch explizit aufgerufen werden. Dies wird jedoch nicht empfohlen.

Bei einem expliziten Aufruf wird nicht nur der explizite Initialisierungscode der FB_init-Methode erneut ausgeführt, sondern auch die implizite Initialisierung findet erneut statt. Es werden z. B. alle Variablen neu initialisiert, sowohl bei automatischer Nullinitialisierung als auch bei individueller Werteinitialisierung.

Letzteres bewirkt, dass beide Initialisierungsschritte auch für die in dieser Instanz deklarierten FB-Instanzen durchgeführt werden. Dieser Vorgang setzt sich über die weiteren enthaltenen Instanzebenen fort.

Wenn FB-Instanzen neu initialisiert werden, ohne vorher de-initialisiert worden zu sein, kann dies zu Problemen im weiteren Programmablauf führen, beispielsweise wenn die FB_init-Methode dynamische Speicherallokationen oder einen Instanzzähler beinhaltet.

Da die FBs dabei nicht entsprechend ihrer vorgesehenen Verwendung genutzt werden und da die problematischen Folgen möglicherweise nicht abzuschätzen sind, wird ausdrücklich davon abgeraten, FB_init-Methoden explizit aufzurufen.

Schnittstelle der Methode FB_init

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online c
hange)
END_VAR
```

Durch die Auswertung der FB_init-Methodenparameter können Sie die Betriebsfälle unterscheiden und die Implementierung gegebenenfalls anpassen. (Siehe Betriebsfälle [▶ 893])

Methodenparameter	(erster/erneuter) Download	Online Change	TwinCAT Neustart (ohne erneuten Download)
bInitRetains	TRUE	FALSE	FALSE
bInCopyCode	FALSE	TRUE	FALSE

Sie können in einer FB_init-Methode zusätzliche Methodeneingänge deklarieren. Diese Eingänge müssen Sie dann in der Deklaration der Funktionsbausteininstanz setzen.

Rückgabewert

Implizite Aufrufe

Bei impliziten Aufrufen der Methoden wird der Rückgabewert vom System nicht ausgewertet. Auch wenn Sie den Rückgabewert anpassen, wird dieser bei einem impliziten Aufruf nicht ausgewertet.

Explizite Aufrufe

Bei expliziten Aufrufen der Methoden können Sie den Rückgabewert auswerten. Dafür können Sie einen sinnvollen Rückgabewert zurückliefern.

FB_init bei abgeleiteten Bausteinen

Wenn ein Funktionsbaustein von einem anderen Funktionsbaustein abgeleitet ist, dann wird für diesen Funktionsbaustein automatisch die FB_init-Methode des Basisfunktionsbausteins ausgeführt. Falls die FB_init-Methode des abgeleiteten Funktionsbausteins explizit hinzugefügt wird, wird diese im Anschluss an die FB_init-Methode des Basisfunktionsbausteins ausgeführt (siehe Verhalten bei abgeleiteten Bausteinen [▶ 898]).

Wenn die FB_init-Methode des abgeleiteten Funktionsbausteins in expliziter Form vorhanden sein soll, muss diese dieselben Parameter definieren wie die FB_init-Methode des Basisfunktionsbausteins. Sie können jedoch weitere Parameter hinzufügen, um für die abgeleitete Instanz eine spezielle Initialisierung einzurichten.

Die FB_init-Methode ist nicht zu vergleichen mit dem Konstrukt eines Konstruktors, wie er aus C#, C++ oder auch Java bekannt ist, da ein Funktionsbaustein in der SPS keinen Konstruktor benötigt, um seine deklarierten Variablen zu initialisieren. Dies findet bereits in den Deklarationszeilen implizit oder auch explizit statt.

Die sich daraus ergebenden Konsequenzen für Funktionsbausteine, die andere Funktionsbausteine erweitern, werden im Abschnitt Verhalten bei abgeleiteten Bausteinen [▶ 898] beschrieben.

Beispiele für die Zuweisung von zusätzlichen FB_init-Parametern

Beispiel 1:

Methode FB_SerialDevice.FB_init

```
METHOD PUBLIC FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online c
hange)
    nComNum      : INT;  // additional input: number of the COM-interface that is to be observed
END_VAR
```

Deklaration des Funktionsbausteins FB_SerialDevice:

```
fbCom1  : FB_SerialDevice(nComNum := 1);
fbCom0  : FB_SerialDevice(nComNum := 0);
```

Beispiel 2:

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR
    nStartValue : INT;
END_VAR
```

Methode FB_Sample.FB_init:

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL; // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online c
hange)
    nValue       : INT;  // parameter to initialize the start value
END_VAR
```

```
nStartValue := nValue;
```

Deklaration des Funktionsbausteins FB_Sample:

```
PROGRAM MAIN
VAR
    fbSample1 : FB_Sample(123); // => fbSample1.nStartValue is set to 123
    fbSample2 : FB_Sample(456); // => fbSample2.nStartValue is set to 456
END_VAR
```

Beispiel 3:

Im folgenden Beispiel werden eine Eingangsvariable und eine Eigenschaft von einem Funktionsbaustein initialisiert, welcher über eine FB_init-Methode [▶ 888] mit einem zusätzlichen Parameter verfügt.

Funktionsbaustein FB_Sample:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    nInput           : INT;
END_VAR
VAR
    nLocalInitParam  : INT;
    nLocalProp       : INT;
END_VAR
```

Methode FB_Sample.FB_init:

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online
change)
    nInitParam   : INT;
END_VAR
```

```
nLocalInitParam := nInitParam;
```

Eigenschaft FB_Sample.nMyProperty und die zugehörige Set-Funktion:

```
PROPERTY nMyProperty : INT
```

```
nLocalProp := nMyProperty;
```

Programm MAIN:

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample(nInitParam := 1) := (nInput := 2, nMyProperty := 3);
    aSample   : ARRAY[1..2] OF FB_Sample[(nInitParam := 4), (nInitParam := 7)]
                            := [(nInput := 5, nMyProperty := 6), (nInput := 8, nMyProperty := 9)];
END_VAR
```

Initialisierungsergebnis:

- fbSample
  - nInput = 2
  - nLocalInitParam = 1
  - nLocalProp = 3
- aSample[1]
  - nInput = 5
  - nLocalInitParam = 4
  - nLocalProp = 6
- aSample[2]
  - nInput = 8
  - nLocalInitParam = 7
  - nLocalProp = 9

Siehe auch:

- Attribut 'call_after_init' [▶ 831]
- Betriebsfälle [▶ 893]

16.12.2 FB_reinit

FB_reinit müssen Sie bei Bedarf explizit implementieren. Wenn diese Methode vorliegt, wird sie automatisch aufgerufen, nachdem die Instanz des entsprechenden Funktionsbausteins kopiert wurde (impliziter Aufruf). Das geschieht während eines Online-Change nach Änderungen an der Funktionsbausteindeklaration (Signaturänderung), um das neue Instanzmodul zu reinitialisieren.

Expliziter Aufruf nicht empfohlen

Bei den Methoden FB_init, FB_reinit und FB_exit handelt es sich um Systemfunktionen, die zu unterschiedlichen Zeitpunkten implizit aufgerufen werden (weitere Informationen hierzu finden Sie unter Betriebsfälle [▶ 893]). Ein expliziter Aufruf dieser Methoden kann unbeabsichtigte Folgen haben und wird daher nicht empfohlen.

Automatisches Core Dump bei Exception in FB_init/FB_reinit/FB_exit

Falls innerhalb des Codes von FB_init/FB_reinit/FB_exit ein Ausnahmefehler auftritt, z.B. aufgrund eines Programmierfehlers, legt das Laufzeitsystem automatisch einen Core Dump auf dem Zielsystem ab (ab TC3.1 Build 4024.25). Dieser Core Dump wird als *.core-Datei im Boot-Ordner des Zielsystems abgelegt (standardmäßig unter *C:\TwinCAT\3.1\Boot\Plc*) und kann zur Ursachensuche verwendet werden.

Weitere Informationen zum Laden eines Core Dumps finden Sie unter: Fehleranalyse mit Core Dump [▶ 259]

Schnittstelle der Methode FB_reinit

```
METHOD FB_reinit : BOOL
```

Rückgabewert

Implizite Aufrufe

Bei impliziten Aufrufen der Methoden wird der Rückgabewert vom System nicht ausgewertet. Auch wenn Sie den Rückgabewert anpassen, wird dieser bei einem impliziten Aufruf nicht ausgewertet.

Explizite Aufrufe

Bei expliziten Aufrufen der Methoden können Sie den Rückgabewert auswerten. Dafür können Sie einen sinnvollen Rückgabewert zurückliefern.

FB_reinit bei abgeleiteten Bausteinen

Um eine Reinitialisierung der Basisimplementierung des Funktionsbausteins zu erreichen, müssen Sie FB_reinit explizit für den Basisbaustein aufrufen (über `SUPER^.FB_reinit()`). Dabei können Sie den Rückgabewert auswerten.

16.12.3 FB_exit

FB_exit müssen Sie bei Bedarf explizit implementieren. Wenn diese Methode vorliegt, wird sie automatisch (implizit) aufgerufen, bevor der Code der Funktionsbausteininstanz von der Steuerung entfernt wird (z. B. auch, wenn TwinCAT vom Run-Modus in den Konfigurationsmodus geschaltet wird).

Expliziter Aufruf nicht empfohlen

Bei den Methoden FB_init, FB_reinit und FB_exit handelt es sich um Systemfunktionen, die zu unterschiedlichen Zeitpunkten implizit aufgerufen werden (weitere Informationen hierzu finden Sie unter Betriebsfälle [▶ 893]). Ein expliziter Aufruf dieser Methoden kann unbeabsichtigte Folgen haben und wird daher nicht empfohlen.

Automatisches Core Dump bei Exception in FB_init/FB_reinit/FB_exit

Falls innerhalb des Codes von FB_init/FB_reinit/FB_exit ein Ausnahmefehler auftritt, z.B. aufgrund eines Programmierfehlers, legt das Laufzeitsystem automatisch einen Core Dump auf dem Zielsystem ab (ab TC3.1 Build 4024.25). Dieser Core Dump wird als *.core-Datei im Boot-Ordner des Zielsystems abgelegt (standardmäßig unter *C:\TwinCAT\3.1\Boot\Plc*) und kann zur Ursachensuche verwendet werden.

Weitere Informationen zum Laden eines Core Dumps finden Sie unter: Fehleranalyse mit Core Dump [▶ 259]

Schnittstelle der Methode FB_exit

```
METHOD FB_exit : BOOL
VAR_INPUT
    bInCopyCode : BOOL; // if TRUE, the exit method is called for exiting an instance that is copied
 afterwards (online change)
END_VAR
```

Durch die Auswertung des FB_exit-Methodenparameters können Sie die Betriebsfälle unterscheiden und die Implementierung gegebenenfalls anpassen. (Siehe Betriebsfälle [▶ 893])

Methodenparameter	(erster/erneuter) Download	Online Change
bInCopyCode	FALSE	TRUE

Rückgabewert

Implizite Aufrufe

Bei impliziten Aufrufen der Methoden wird der Rückgabewert vom System nicht ausgewertet. Auch wenn Sie den Rückgabewert anpassen, wird dieser bei einem impliziten Aufruf nicht ausgewertet.

Explizite Aufrufe

Bei expliziten Aufrufen der Methoden können Sie den Rückgabewert auswerten. Dafür können Sie einen sinnvollen Rückgabewert zurückliefern.

FB_exit bei abgeleiteten Bausteinen

Wenn ein Funktionsbaustein von einem anderen Funktionsbaustein abgeleitet ist, dann wird für diesen Funktionsbaustein automatisch die FB_exit-Methode des Basisfunktionsbausteins ausgeführt. Falls die FB_exit-Methode des abgeleiteten Funktionsbausteins explizit hinzugefügt wird, wird diese zuerst ausgeführt und anschließend die FB_exit-Methode des Basisfunktionsbausteins (siehe Verhalten bei abgeleiteten Bausteinen [▶ 898]).

Siehe auch:

- Attribut 'no-exit' [▶ 851]

16.12.4 Betriebsfälle

Die Methoden FB_init, FB_reinit und FB_exit werden zu unterschiedlichen Zeitpunkten implizit aufgerufen. Zu diesen verschiedenen Betriebsfällen finden Sie im Folgenden Informationen:

- Betriebsfall „Erster Download“
- Betriebsfall „Erneuter Download“
- Betriebsfall „Online-Change“

Je nach Betriebsfall unterscheidet sich das Aufrufverhalten der Methoden. Durch die Abfrage der Methodenparameter „bInCopyCode“ und „bInitRetains“ von FB_init und FB_exit können Sie innerhalb der Methoden zwischen den Betriebsfällen unterscheiden. Das ermöglicht eine Anpassung der Implementierung an den jeweiligen Betriebsfall.

Betriebsfall „Erster Download“	Betriebsfall „Erneuter Download“	Betriebsfall „Online-Change“
1. FB_init (impliziter und expliziter Initialisierungscode) 2. explizite externe Variableninitialisierung über Instanzdeklaration des Funktionsbausteins 3. Methode, die mit Attribut 'call_after_init' deklariert ist	1. FB_exit 2. FB_init (impliziter und expliziter Initialisierungscode) 3. explizite externe Variableninitialisierung über Instanzdeklaration des Funktionsbausteins 4. Methode, die mit Attribut 'call_after_init' deklariert ist	1. FB_exit 2. FB_init (impliziter und expliziter Initialisierungscode) 3. explizite externe Variableninitialisierung über Instanzdeklaration des Funktionsbausteins 4. Methode, die mit Attribut 'call_after_init' deklariert ist 5. Kopiervorgang copy 6. FB_reinit

Methodenparameter:
`FB_init(bInitRetains := TRUE, bInCopyCode := FALSE);`
	Methodenparameter:
	`FB_exit(bInCopyCode := FALSE);`
	`FB_init(bInitRetains := TRUE, bInCopyCode := FALSE);`
		Methodenparameter:
		`FB_exit(bInCopyCode := TRUE);`
		`FB_init(bInitRetains := FALSE, bInCopyCode := TRUE);`

Betriebsfall „Erster Download“

Beim Download eines SPS-Projekts auf eine Steuerung, die sich im Auslieferungszustand befindet, müssen die Speicherplätze aller Variablen durch Initialisierung in den gewünschten Ausgangszustand versetzt werden. Dabei werden die Datenbereiche von Funktionsbausteininstanzen mit den gewünschten Werten belegt. Durch die explizite Implementierung von FB_init [▶ 888] für Funktionsbausteine können Sie die Initialisierung in dieser Situation gezielt beeinflussen.

Durch die Auswertung der FB_init-Methodenparameter „bInitRetains“ (TRUE) und „bInCopyCode“ (FALSE) können Sie diesen Betriebsfall eindeutig detektieren. (Siehe auch Betriebsfall „Online-Change“)

Die FB_init-Methode ist nicht zu vergleichen mit dem Konstrukt eines Konstruktors, wie er aus C#, C++ oder auch Java bekannt ist, da ein Funktionsbaustein in der SPS keinen Konstruktor benötigt, um seine deklarierten Variablen zu initialisieren. Dies findet bereits in den Deklarationszeilen implizit oder auch explizit statt.

Die sich daraus ergebenden Konsequenzen für Funktionsbausteine, die andere Funktionsbausteine erweitern, werden im Abschnitt Verhalten bei abgeleiteten Bausteinen [▶ 898] beschrieben.

Attribut 'call_after_init' als Alternative zu FB_init

Nach dem Aufruf von FB_init und vor dem ersten Zyklus der Tasks eines SPS-Projekts werden die externen initialen Zuweisungen innerhalb der Deklaration der Funktionsbausteininstanz verarbeitet.

```
fbTimer : TON := (PT := T#500MS);
```

Solche Zuweisungen werden erst nach dem Aufruf von FB_init ausgeführt. Hätte TON eine FB_init-Methode, so wäre in dieser der zugewiesene Zeitwert von T#500MS nicht bekannt.

Um die Auswirkungen dieser Zuweisungen kontrollieren zu können, können Sie eine Methode eines Funktionsbausteins mit dem Attribut {attribute 'call_after_init'} [▶ 831] versehen. Sie müssen das Attribut sowohl über dem Deklarationsteil der entsprechenden Methode als auch über dem Deklarationsteil des Funktionsbausteinrumpfes einfügen.

Die Methode wird nach der Verarbeitung der externen initialen Zuweisungen und vor dem Start der Tasks eines SPS-Projekts aufgerufen und kann so auf die Vorgaben des Anwenders (bei der externen Initialisierung) entsprechend reagieren.

Beispiel

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB_Sample
VAR_INPUT
```

```
    nInput1   : INT;
    nInput2   : INT := 100;
END_VAR
```

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains   : BOOL; // if TRUE, the retain variables are initialized (warm start / cold
start)
    bInCopyCode    : BOOL; // if TRUE, the instance afterwards gets moved into the copy code (online
change)
END_VAR
```

```
{attribute 'call_after_init'}
METHOD MyCallAfterInit
```

```
PROGRAM MAIN
VAR
    fbSample  : FB_Sample := (nInput2 := 700);
END_VAR
```

Beim Download erfolgen nacheinander folgende Aufrufe:

1. FB_init	Instanz initialisieren

	1. impliziter Initialisierungscode (implizite Nullinitialisierung und explizite interne Werteinitialisierung der Variablen)
	`nInput1 := 0;`
	`nInput2 := 100;`

	2. expliziter Initialisierungscode (explizit in FB_init definierter Initialisierungscode)
	`fbSample.FB_init(bInitRetains := TRUE, bInCopyCode := FALSE);`
	Durch die Auswertung der FB_init-Methodenparameter können Sie diesen Betriebsfall eindeutig detektieren. Über die Methode FB_init können Sie beispielsweise Variablen mithilfe von zusätzlichem Initialisierungscode initialisieren oder andere Teile des SPS-Projekts über die Position bestimmter Variablen im Speicher informieren.
2. explizite externe Variableninitialisierung über Instanzdeklaration des Funktionsbausteins	Externe initiale Zuweisungen verarbeiten
	`nInput2 := 700;`
	Wenn den Eingangsvariablen des Funktionsbausteins Werte zugewiesen sind, werden diese kopiert. Bei Variablen, denen von außen kein Wert zugewiesen ist, bleibt der ursprüngliche Wert erhalten.
3. Methode, die mit Attribut call_after_init deklariert ist	Alternative Initialisierung
	`fbSample.MyCallAfterInit();`
	Um diesen Betriebsfall eindeutig zu detektieren, können Sie den Wert von bInCopyCode zuvor in FB_init in eine Hilfsvariable kopieren und diese Hilfsvariable in der 'call_after_init'-Methode auswerten. Sie können das Attribut als Alternative zu FB_init nutzen oder beispielsweise die Auswirkungen der expliziten externen Variableninitialisierung kontrollieren. Gestalten Sie die Implementierung möglichst unabhängig. Die Methode kann nämlich auch aus dem SPS-Projekt heraus jederzeit aufgerufen werden, um eine Funktionsbausteininstanz in den ursprünglichen Zustand zurückzusetzen.

Betriebsfall „Erneuter Download“

Beim erneuten Download eines SPS-Projekts wird eventuell ein bereits vorhandenes Projekt auf der Steuerung ersetzt. Deshalb muss der Speicherplatz für die vorhandenen Funktionsbausteine zunächst geregelt freigegeben werden. Dafür können Sie die Methode FB_exit [▶ 892] verwenden. Ist diese Methode angelegt, wird sie aufgerufen, bevor das alte Projekt entfernt wird. Anschließend wird das neue Projekt auf die Steuerung geladen und FB_init aufgerufen. In FB_exit können Sie beispielsweise externe Ressourcen (mit Socket- oder File-Handles) in einen definierten Zustand versetzen oder dynamisch allokierten Speicher freigeben (__NEW- bzw. in diesem Fall __DELETE-Operator).

Durch die Auswertung des FB_exit-Methodenparameters „bInCopyCode“ (FALSE) können Sie diesen Betriebsfall eindeutig detektieren.

Betriebsfall „Online-Change“

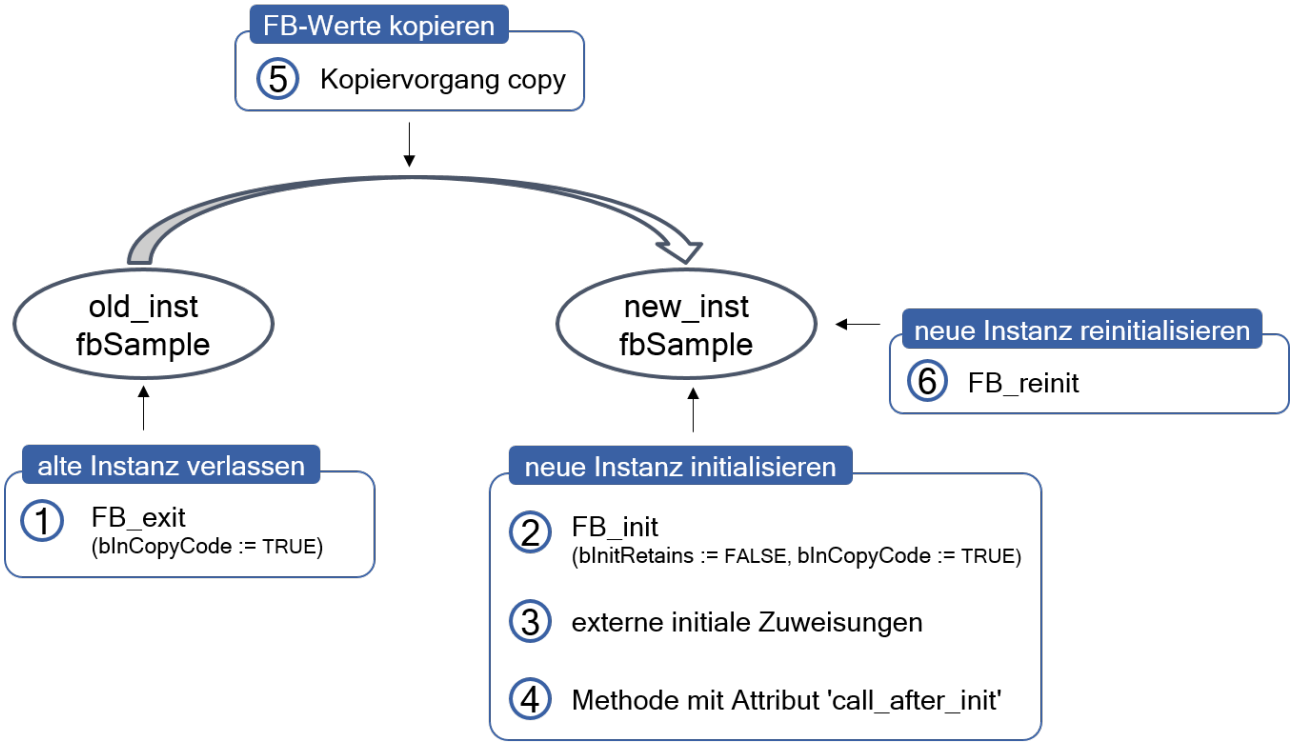
Bei einem Online-Change können Sie über die Methoden FB_exit, FB_init und FB_reinit die Initialisierung von Funktionsbausteininstanzen beeinflussen. Im Rahmen des Online-Change werden die im Offlinebetrieb vollzogenen Änderungen an dem SPS-Projekt in der laufenden Steuerung nachvollzogen. Deshalb werden die „alten“ Instanzen der Funktionsbausteine durch ihre „neuen Geschwister" möglichst stoßfrei ersetzt.

Wenn vor dem Login in dem SPS-Projekt keine Änderungen im Deklarationsteil eines Funktionsbausteins vorgenommen wurden, sondern nur in der Implementierung, dann findet keine Ersetzung der Datenbereiche statt. Es werden nur Codeblöcke ersetzt. Die Methoden FB_exit, FB_init und FB_reinit werden in diesem Fall nicht aufgerufen.

Wenn Sie Änderungen in der Deklaration eines Funktionsbausteins vorgenommen haben, die zum oben beschriebenen Kopiervorgang führen, erhalten Sie beim Online-Change eine Meldung zu den „möglicherweise unbeabsichtigten Auswirkungen“. In den **Details** der Meldungsbox sehen Sie eine Liste aller zu kopierenden Instanzen aufgelistet.

Im Code der Methoden FB_init und FB_exit kann durch Auswertung der Parameter „bInitRetains“ (FALSE) und „bInCopyCode“ (TRUE) ermittelt werden, ob gerade ein Online-Change ausgeführt wird, der die Funktionsbausteininstanz betrifft und sie an einen anderen Speicherplatz verschiebt.

Beim Online-Change erfolgen nacheinander folgende Aufrufe:



1. FB_exit	Alte Instanz verlassen
	`old_inst.FB_exit(bInCopyCode := TRUE);`
	Durch die Auswertung des FB_exit-Methodenparameters können Sie diesen Betriebsfall eindeutig detektieren. Sie können den Aufruf von FB_exit beim Verlassen der „alten" Instanz verwenden, um vor dem Kopiervorgang bestimmte Aufräumarbeiten anzustoßen. So können Sie die Daten für den folgenden Kopiervorgang vorbereiten und den Zustand der „neuen“ Instanz beeinflussen. Andere Teile des SPS-Projekts können Sie über die bevorstehende Lageänderung im Speicher informieren. Achten Sie besonders auf Variablen vom Typ POINTER oder REFERENCE. Diese verweisen nach dem Online-Change eventuell nicht mehr auf die gewünschten Speicherstellen. (Hinweis: Sie können das Attribut 'call_on_type_change' [▶ 833] nutzen, um auf die Datentypänderung eines referenzierten Funktionsbausteins zu reagieren.) Schnittstellenvariablen (INTERFACE) werden vom Compiler gesondert behandelt und beim Online-Change entsprechend angepasst. Externe Ressourcen wie beispielsweise Sockets, Files oder andere Handles können eventuell unverändert von der neuen Instanz übernommen werden. Sie müssen während des Online-Change oft nicht gesondert behandelt werden. (Siehe Betriebsfall „Erneuter Download")
2. FB_init	Neue Instanz initialisieren
	1. impliziter Initialisierungscode (implizite Nullinitialisierung und explizite interne Werteinitialisierung der Variablen) 2. expliziter Initialisierungscode (explizit in FB_init definierter Initialisierungscode)
	`new_inst.FB_init(bInitRetains := FALSE, bInCopyCode := TRUE);`
	Durch die Auswertung der FB_init-Methodenparameter können Sie diesen Betriebsfall eindeutig detektieren. Der Aufruf von FB_init erfolgt vor dem Kopiervorgang und kann verwendet werden, um für den Online-Change spezifische Operationen auszuführen. Beispielsweise können Sie darüber andere Teile des SPS-Projekts über die neue Position bestimmter Variablen im Speicher informieren.
3. explizite externe Variableninitialisierung über Instanzdeklaration des Funktionsbausteins	Externe initiale Zuweisungen verarbeiten
	`new_inst : <FB-Name> := (<Variable>:=<value>);`
	Wenn den Eingangsvariablen des Funktionsbausteins Werte zugewiesen sind, werden diese kopiert. Bei Variablen, denen von außen kein Wert zugewiesen ist, bleibt der ursprüngliche Wert erhalten.
4. Methode, die mit Attribut call_after_init deklariert ist	Alternative Initialisierung
	`new_inst.<Methodenname der gekennzeichneten Methode>();`
	Um diesen Betriebsfall eindeutig zu detektieren, können Sie den Wert von bInCopyCode zuvor in FB_init in eine Hilfsvariable kopieren und diese Hilfsvariable in der 'call_after_init'-Methode auswerten. Sie können das Attribut beispielsweise als Alternative zu FB_init nutzen. Gestalten Sie die Implementierung möglichst unabhängig. Die Methode kann nämlich auch aus dem SPS-Projekt heraus jederzeit aufgerufen werden, um eine Funktionsbausteininstanz in den ursprünglichen Zustand zurückzusetzen.
5. Kopiervorgang copy	Funktionsbausteinwerte kopieren (Copy-Code)
	`copy(&old_inst, &new_inst);`
	Bestehende Werte bleiben erhalten. Zu diesem Zweck werden sie aus der alten Instanz in die neue kopiert.

6. FB_reinit	Neue Instanz reinitialisieren
	`new_inst.FB_reinit();`
	Diese Methode wird nach dem Kopiervorgang aufgerufen und setzt die Variablen der Instanz auf definierte Werte. Beispielsweise können Sie darüber Variablen an der „neuen“ Position im Speicher entsprechend initialisieren oder andere Teile des SPS-Projekts über die neue Position bestimmter Variablen im Speicher informieren. Gestalten Sie die Implementierung unabhängig vom Online-Change. Die Methode kann nämlich auch aus dem SPS-Projekt heraus jederzeit aufgerufen werden, um eine Funktionsbausteininstanz in den ursprünglichen Zustand zurückzusetzen.

Mit dem Attribut {attribute 'no_copy'} [▶ 849], können Sie für eine einzelne Variable des Funktionsbausteins verhindern, dass diese beim Online-Change kopiert wird. Sie behält dann immer den Initialwert.

Siehe auch:

- Attribut 'call_after_init' [▶ 831]
- Attribut 'call_on_type_change' [▶ 833]
- Attribut 'no_copy' [▶ 849]
- Attribut 'no-exit' [▶ 851]

16.12.5 Verhalten bei abgeleiteten Bausteinen

Beachten Sie bei Verwendung der Methoden FB_init, FB_reinit und FB_exit im Kontext von abgeleiteten Bausteinen die folgenden Hinweise.

FB_init bei abgeleiteten Bausteinen

Wenn ein Funktionsbaustein von einem anderen Funktionsbaustein abgeleitet ist, dann wird für diesen Funktionsbaustein automatisch die FB_init-Methode des Basisfunktionsbausteins ausgeführt. Falls die FB_init-Methode des abgeleiteten Funktionsbausteins explizit hinzugefügt wird, wird diese im Anschluss an die FB_init-Methode des Basisfunktionsbausteins ausgeführt (siehe Verhalten bei abgeleiteten Bausteinen [▶ 898]).

Wenn die FB_init-Methode des abgeleiteten Funktionsbausteins in expliziter Form vorhanden sein soll, muss diese dieselben Parameter definieren wie die FB_init-Methode des Basisfunktionsbausteins. Sie können jedoch weitere Parameter hinzufügen, um für die abgeleitete Instanz eine spezielle Initialisierung einzurichten.

FB_reinit bei abgeleiteten Bausteinen

Um eine Reinitialisierung der Basisimplementierung des Funktionsbausteins zu erreichen, müssen Sie FB_reinit explizit für den Basisbaustein aufrufen (über `SUPER^.FB_reinit()`). Dabei können Sie den Rückgabewert auswerten.

FB_exit bei abgeleiteten Bausteinen

Wenn ein Funktionsbaustein von einem anderen Funktionsbaustein abgeleitet ist, dann wird für diesen Funktionsbaustein automatisch die FB_exit-Methode des Basisfunktionsbausteins ausgeführt. Falls die FB_exit-Methode des abgeleiteten Funktionsbausteins explizit hinzugefügt wird, wird diese zuerst ausgeführt und anschließend die FB_exit-Methode des Basisfunktionsbausteins (siehe Verhalten bei abgeleiteten Bausteinen [▶ 898]).

Beispiel für die Aufrufabfolge bei abgeleiteten Bausteinen

Die Funktionsbausteine FB_Base, FB_Sub und FB_SubSub leiten voneinander ab. Dabei gilt:

- FB_Sub EXTENDS FB_Base

- FB_SubSub EXTENDS FB_Sub

Situation:

- Alle drei Funktionsbausteine verfügen jeweils über eine eigene FB_init-, FB_reinit- und FB_exit-Methode.
- Der Baustein FB_SubSub ist instanziiert. Dem Baustein wird per Online-Change eine zusätzliche Variable hinzugefügt.

Annahme - Fall 1:

- Es wird keine Reinitialisierung der Basisimplementierung gewünscht. Das heißt, dass in der Methode FB_SubSub.FB_reinit der Aufruf SUPER^.FB_reinit() nicht vorhanden ist.

Aufrufabfolge:

Die Aufrufabfolge der Methoden FB_exit, FB_init und FB_reinit für die Funktionsbausteininstanz fbSubSub ist dann wie folgt:

1. `Impliziter Aufruf von FB_SubSub.FB_exit(TRUE) für fbSubSub`
2. `Impliziter Aufruf von FB_Sub.FB_exit(TRUE) für fbSubSub`
3. `Impliziter Aufruf von FB_Base.FB_exit(TRUE) für fbSubSub`
4. `Impliziter Aufruf von FB_Base.FB_init(FALSE, TRUE) für fbSubSub`
5. `Impliziter Aufruf von FB_Sub.FB_init(FALSE, TRUE) für fbSubSub`
6. `Impliziter Aufruf von FB_SubSub.FB_init(FALSE, TRUE) für fbSubSub`
7. `Impliziter Aufruf von FB_SubSub.FB_reinit() für fbSubSub`

Annahme – Fall 2:

- Es wird eine Reinitialisierung der Basisimplementierung gewünscht. Das heißt, dass in den Methoden FB_SubSub.FB_reinit und FB_Sub.FB_reinit der Aufruf SUPER^.FB_reinit() vorhanden ist.

Aufrufabfolge:

Die Aufrufabfolge der Methoden FB_exit, FB_init und FB_reinit für die Funktionsbausteininstanz fbSubSub ist dann wie folgt:

1. `Impliziter Aufruf von FB_SubSub.FB_exit(TRUE) für fbSubSub`
2. `Impliziter Aufruf von FB_Sub.FB_exit(TRUE) für fbSubSub`
3. `Impliziter Aufruf von FB_Base.FB_exit(TRUE) für fbSubSub`
4. `Impliziter Aufruf von FB_Base.FB_init(FALSE, TRUE) für fbSubSub`
5. `Impliziter Aufruf von FB_Sub.FB_init(FALSE, TRUE) für fbSubSub`
6. `Impliziter Aufruf von FB_SubSub.FB_init(FALSE, TRUE) für fbSubSub`
7. `Impliziter Aufruf von FB_SubSub.FB_reinit() für fbSubSub`
8. `Expliziter Aufruf von FB_Sub.FB_reinit() für fbSubSub`
9. `Expliziter Aufruf von FB_Base.FB_reinit() für fbSubSub`

17 Referenz Benutzeroberfläche

Standardmäßig sind die wichtigsten Befehle in der Benutzeroberfläche von TwinCAT verfügbar. Wenn Sie die Menükonfiguration individuell anpassen möchten, wählen Sie den Befehl **Anpassen** im Menü **Extras**.

Siehe auch:

- Benutzeroberfläche anpassen

17.1 Datei

17.1.1 Archivierungsmöglichkeiten

Um ein TwinCAT-Projekt z.B. für Archivierungszwecke oder für die Weitergabe an Kollegen zu speichern, bietet die TwinCAT-Entwicklungsumgebung drei verschiedene Archivierungsdateitypen an: tnzip, tszip und tpzip.

Welcher Archivierungsdateityp verwendet werden sollte, hängt davon ab, welche Projekte in dem Archivordner gespeichert werden sollen.

Archivierungsdateityp	Fokus	Inhalt	Befehle
tnzip	Projektmappe [▶ 900]	Der Archivordner *.tnzip enthält alle in der Projektmappe enthaltenen TwinCAT-Projekttypen. Dies können TwinCAT-, TwinCAT HMI-, Scope- und Connectivity-Projekte sein.	Erstellen [▶ 900] Öffnen [▶ 901]
tszip	TwinCAT-Projekt [▶ 901]	Der Archivordner *.tszip enthält das TwinCAT-Projekt, welches archiviert wird.	Erstellen [▶ 1113] Öffnen [▶ 901]
tpzip	SPS-Projekt [▶ 902]	Der Archivordner *.tpzip enthält das SPS-Projekt, welches archiviert wird.	Erstellen [▶ 902] Öffnen [▶ 903]

17.1.1.1 Projektmappe

- TwinCAT-Projektmappenarchiv *.tnzip erstellen: Befehl Save <Projektmappenname> as Archive... [▶ 900]
- TwinCAT-Projektmappenarchiv *.tnzip öffnen: Befehl Open Solution from Archive [▶ 901]

17.1.1.1.1 Befehl Save <Projektmappenname> as Archive...

Funktion: Der Befehl öffnet den Standarddialog zum Speichern einer Datei als Archiv. Die Projektmappe kann unter dem gewünschten Speicherpfad als ***.tnzip**-Archiv abgelegt werden.

Aufruf: Menü **Datei**, Kontextmenü

Voraussetzung: Die Projektmappe ist im **Projektmappen-Explorer** ausgewählt.

Inhalt vom *.tnzip	Der Archivordner *.tnzip enthält alle in der Projektmappe enthaltenen TwinCAT-Projekttypen. Dies können TwinCAT-, TwinCAT HMI-, Scope- und Connectivity-Projekte sein.
Befehl zum Öffnen	Ein tnzip-Archiv kann über den folgenden Befehl wieder geöffnet werden: Befehl Open Solution from Archive [▶ 901].
Hinweis zu SPS-Projekten	Falls die Projektmappe ein oder mehrere SPS-Projekte enthält, sind die Dateien und Ordner, die bezüglich dieser SPS-Projekte in dem Archivordner gespeichert werden, abhängig von den SPS-Projekteinstellungen des jeweiligen SPS-Projekts. Registerkarte Settings [▶ 969]

17.1.1.1.2 Befehl Open Solution from Archive

Funktion: Der Befehl extrahiert ein TwinCAT-Projektmappenarchiv ***.tnzip**.

Aufruf: Menü **Datei > Öffnen**

Nachdem Sie den Befehl ausgeführt haben, öffnet sich der Dialog **Öffnen**. Wählen Sie aus dem Dateisystem die Archivdatei aus und bestätigen Sie den Dialog. Anschließend öffnet sich der Dialog **Wähle Verzeichnis für neue Arbeitsmappe.** Wählen Sie einen Ordner aus, in dem Sie die extrahierten Projektmappendateien speichern wollen.

Inhalt vom *.tnzip	Der Archivordner *.tnzip enthält alle in der Projektmappe enthaltenen TwinCAT-Projekttypen. Dies können TwinCAT-, TwinCAT HMI-, Scope- und Connectivity-Projekte sein.
Befehl zum Erstellen	Ein tnzip-Archiv kann über den folgenden Befehl erstellt werden: Befehl Save <Projektmappenname> as Archive... [▶ 900]
Hinweis zu SPS-Projekten	Falls die Projektmappe ein oder mehrere SPS-Projekte enthält, sind die Dateien und Ordner, die bezüglich dieser SPS-Projekte in dem Archivordner gespeichert werden, abhängig von den SPS-Projekteinstellungen des jeweiligen SPS-Projekts. Registerkarte Settings [▶ 969]

17.1.1.2 TwinCAT-Projekt

- TwinCAT-Projektarchiv *.tszip erstellen: Befehl Sichern <TwinCAT-Projektname> als Archiv... [▶ 1113]
- TwinCAT-Projektarchiv *.tszip öffnen: Befehl Projekt/Projektmappe (Projekt/Projektmappe öffnen) [▶ 901]

17.1.1.2.1 Befehl Projekt/Projektmappe (Projekt/Projektmappe öffnen)

Symbol: 

Tastaturkürzel: **[Strg] + [Umschalt] + [O]**

Funktion: Der Befehl öffnet den Standarddialog zum Öffnen einer Datei. Hier können Sie das Dateisystem nach einer TwinCAT-Projektdatei durchsuchen und im Entwicklungssystem öffnen.

Aufruf: Menü **Datei > Öffnen**

Dialog Projekt öffnen

Dateityp	Auswahlliste zum Filtern des Dateityps • Dateien aller unterstützten Formate können geöffnet werden.
Optionen	• Hinzufügen (Verwenden Sie diese Option nur, um einer Projektmappe z. B. noch ein Measurement-Projekt hinzuzufügen. Verwenden Sie diese Option nicht, um einer Projektmappe mehrere TwinCAT-Projekte hinzuzufügen.) • Mappe schließen
Öffnen	TwinCAT öffnet die gewählte Projektdatei. Wenn erforderlich, wird sie zuvor konvertiert.

TwinCAT-Projektarchiv *.tszip

Inhalt vom *.tszip	Der Archivordner *.tszip enthält das TwinCAT-Projekt, welches archiviert wird.
Befehl zum Erstellen	Ein tszip-Archiv kann über den folgenden Befehl erstellt werden: Befehl Sichern <TwinCAT-Projektname> als Archiv... [▶ 1113]
Hinweis zu SPS-Projekten	Falls das TwinCAT-Projekt ein oder mehrere SPS-Projekte enthält, sind die Dateien und Ordner, die bezüglich dieser SPS-Projekte in dem Archivordner gespeichert werden, abhängig von den SPS-Projekteinstellungen des jeweiligen SPS-Projekts. Registerkarte Settings [▶ 969]

Siehe auch:

- Dokumentation PLC: Ihr erstes TwinCAT-3-SPS-Projekt [▶ 25]
- Dokumentation PLC: Standardprojekt anlegen [▶ 55]

17.1.1.3 SPS-Projekt

- SPS-Projektarchiv *.tpzip erstellen: Befehl Sichern <SPS-Projektname> als Archiv... [▶ 902]
- SPS-Projektarchiv *.tpzip öffnen: Befehl Vorhandenes Element hinzufügen (Projekt) [▶ 903]

17.1.1.3.1 Befehl Sichern <SPS-Projektname> als Archiv...

Funktion: Der Befehl öffnet den Standarddialog zum Speichern einer Datei als Archiv. Das SPS-Projekt kann unter dem gewünschten Speicherpfad als ***.tpzip**-Archiv abgelegt werden.

Aufruf: Menü **Datei**, Kontextmenü

Voraussetzung: Das TwinCAT-SPS-Projekt (<SPS-Projektname>) ist im **Projektmappen-Explorer** ausgewählt.

Die Dateien und Ordner in dem Archivordner sind abhängig von den SPS-Projekteinstellungen.

Inhalt vom *.tpzip	Der Archivordner *.tpzip enthält das SPS-Projekt, welches archiviert wird.
Befehl zum Öffnen	Ein tpzip-Archiv kann über den folgenden Befehl wieder geöffnet werden: Befehl Vorhandenes Element hinzufügen (Projekt) [▶ 903]
Hinweis zu SPS-Projekten	Die Dateien und Ordner, die bezüglich des SPS-Projekts in dem Archivordner gespeichert werden, sind abhängig von den SPS-Projekteinstellungen dieses SPS-Projekts. Registerkarte Settings [▶ 969]

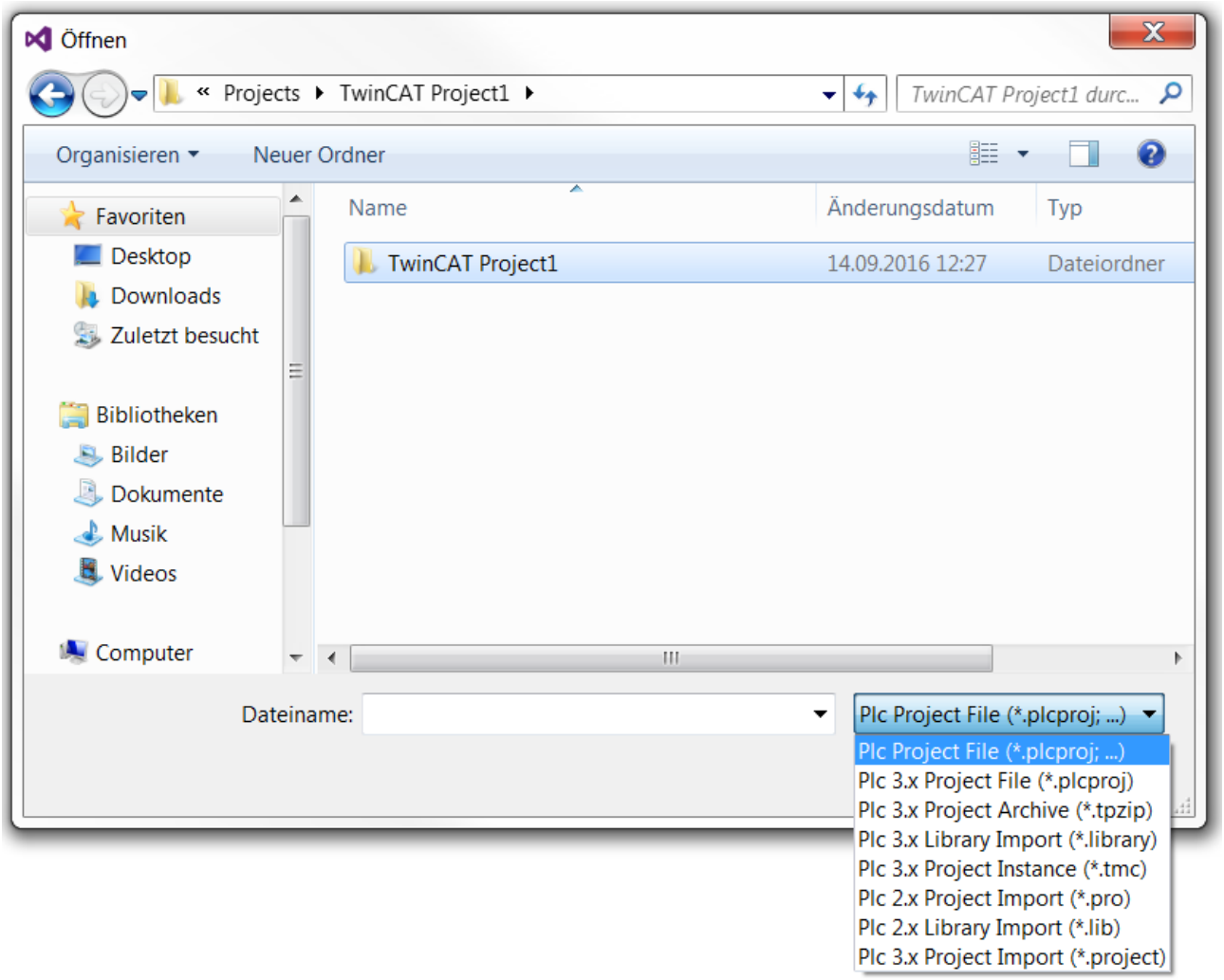
17.1.1.3.2 Befehl Vorhandenes Element hinzufügen (Projekt)

Symbol: 

Funktion: Der Befehl öffnet den Standard-Browserdialog, über den eine SPS-Projektdatei gesucht und im Programmiersystem geöffnet werden kann. Wenn ein entsprechender Konverter installiert ist, können auch Projekte in einem anderen Format geöffnet werden.

Aufruf: Menü **Projekt** oder Kontextmenü SPS-Objekt im **Projektmappen-Explorer**

Voraussetzung: Der SPS-Knoten ist im TwinCAT-Projektbaum selektiert.



Dateityp	Standardmäßig können Sie den Filter auf einen der folgenden Dateitypen setzen: • PLC 3.x Projektdatei (*.PLCproject): TwinCAT-3-PLC-Projekte mit der Erweiterung „.PLCproject“ • PLC 3.x Projektarchiv (*.tpzip): TwinCAT-3-PLC-Projektarchive mit der Erweiterung „.tpzip“ ◦ Siehe auch: Befehl Sichern <SPS-Projektname> als Archiv... [▶ 902] • PLC 3.x Bibliotheksimport (*.library): TwinCAT-3-PLC-Bibliotheken mit der Erweiterung „.library“, • PLC 2.x Projektdatei (*.pro): TwinCAT-2-PLC-Projekte mit der Erweiterung „.pro“ • PLC 2.x Bibliotheksimport (*.lib): TwinCAT-2-PLC-Bibliotheken mit der Erweiterung „.lib“ • PLC 3.x Projektimport (*.PLCproject): PLC-Projekte mit der Erweiterung „.project“
Öffnen	Die gewählte Projektdatei wird geöffnet bzw. konvertiert und dann geöffnet.

SPS-Projektarchiv *.tpzip

Inhalt vom *.tpzip	Der Archivordner *.tpzip enthält das SPS-Projekt, welches archiviert wird.
Befehl zum Erstellen	Ein tpzip-Archiv kann über den folgenden Befehl erstellt werden: Befehl Sichern <SPS-Projektname> als Archiv... [▶ 902]
Hinweis zu SPS-Projekten	Die Dateien und Ordner, die bezüglich des SPS-Projekts in dem Archivordner gespeichert werden, sind abhängig von den SPS-Projekteinstellungen dieses SPS-Projekts. Registerkarte Settings [▶ 969]

Mögliche Szenarien beim Öffnen eines SPS-Projekts

Folgende Szenarien sind beim Öffnen eines Projekts möglich:

1. Ein anderes Projekt ist noch geöffnet. [▶ 904]
2. Das Projekt wurde mit einer älteren TwinCAT-3-Version gespeichert. [▶ 904]
3. Das Projekt wurde nicht mit TwinCAT 3 gespeichert. [▶ 904]
4. Das Projekt wurde nicht regulär beendet und „Automatisch Speichern“ war aktiviert. [▶ 906]
5. Das Projekt ist schreibgeschützt. [▶ 906]
6. Es handelt sich um eine Bibliothek, die in einem Bibliotheks-Repository installiert ist und aus diesem aufgerufen wird. [▶ 906]

1. Ein anderes Projekt ist noch geöffnet.

Sie werden gefragt, ob das andere Projekt gespeichert und geschlossen werden soll.

2. Das Projekt wurde mit einer älteren TwinCAT-3-Version gespeichert.

Wenn sich das Speicherformat unterscheidet, weil das geöffnete Projekt mit einer älteren Version von TwinCAT 3 gespeichert wurde, gibt es zwei Fälle:

- Wenn das Projekt nicht im Speicherformat des aktuell verwendeten Programmiersystems speicherbar ist, müssen Sie es aktualisieren, um weiter am Projekt arbeiten zu können. Der an dieser Stelle auftretende Ausdruck **Die durchgeführten Änderungen...** bezieht sich auf interne Aktionen verschiedener Komponenten während des Ladens des Projekts.
- Wenn das Projekt im bisherigen Speicherformat weiterhin speicherbar ist, können Sie entscheiden, ob das Speicherformat beibehalten oder aktualisiert werden soll. Wenn das Speicherformat beibehalten werden soll, muss mit möglichen Datenverlusten gerechnet werden. Wenn das Speicherformat aktualisiert werden soll, kann das Projekt nicht mehr mit der alten Version des Programmiersystems geöffnet werden.

Neben dem Speicherformat können sich auch die Versionen der explizit eingefügten Bibliotheken, des Visualisierungsprofils und die Compiler-Version des öffnenden Projekts von denen unterscheiden, die mit dem aktuellen Programmiersystem installierten wurden.

Wenn auf dem aktuellen Programmiersystem neuere Versionen installiert sind, öffnet sich automatisch der Dialog **Projektumgebung**, in dem Sie die Versionen aktualisieren können. Wenn an dieser Stelle noch keine Aktualisierung vorgenommen wird, kann dies jederzeit im Dialog **Optionen > Projektumgebung** nachgeholt werden.

Compiler-Version beachten

Wenn ein Projekt geöffnet wird, das mit einer älteren Version des Programmiersystems erstellt wurde und für das in den Projekteinstellungen die neueste Compiler-Version eingestellt ist, während im neuen Programmiersystem die Projektumgebungseinstellung für die Compiler-Version **Nicht aktualisieren** ist, dann wird die zuletzt im alten Projekt verwendete Compiler-Version weiter verwendet (nicht die „Aktuelle“ in der neuen Umgebung).

3. Das Projekt wurde nicht mit TwinCAT 3 gespeichert.

Fall 1)

Wenn Sie beim Auswählen des zu öffnenden Projekts den Dateifilter gezielt setzen und ein entsprechender Konverter verfügbar ist, wird der Konverter automatisch verwendet und das Projekt in das aktuelle Format gebracht. Die Konvertierung läuft konverterspezifisch ab. In der Regel werden Sie aufgefordert, die Behandlung von eingebundenen Bibliotheken oder Geräte-Referenzen zu definieren.

TwinCAT-3-Konverter

Die Anpassung eines TwinCAT-PLC-Control-Projekts an die TwinCAT-3-Syntax kann beim Import nur gelingen, wenn der Konverter das Projekt fehlerfrei übersetzen kann.

Wenn Sie beim Auswählen des zu öffnenden Projekts den Dateityp **Alle Dateien** eingestellt haben, ist kein Konverter aktiviert und es öffnet sich der Dialog **Projekt konvertieren**. In dem Dialog müssen Sie die Konvertierung des Projekts durch Auswählen einer der Optionen explizit anstoßen.

- **In das aktuelle Format konvertieren:** Wählen Sie aus der Auswahlliste den Konverter, der verwendet werden soll (Anwendung zum Konvertieren). Nach dem Konvertieren kann das Projekt nicht mehr in der alten Version geöffnet werden.
- **Ein neues Projekt erzeugen und ein spezielles Gerät hinzufügen:** (Noch nicht implementiert)

TwinCAT 2.x PLC-Control-Projektoptionen

Der in den TwinCAT 2.x PLC-Control-Projektoptionen eingestellte Projektverzeichnispfad sowie die Projektinformationen werden in den Dialog **Projektinformationen** übernommen.

Fall 2)

Wenn im Projekt Bibliotheken eingebunden sind, für die noch kein „Konvertierungs-Mapping“ in den Bibliotheks-Optionen gespeichert ist, erscheint der Dialog **Konvertierung einer Bibliotheksreferenz**, in dem Sie definieren, wie diese Referenz konvertiert werden sollen:

- **Die Bibliothek ebenfalls konvertieren und installieren:** Wenn Sie diese Option aktivieren, wird die eingebundene Bibliothek in das neue Format übergeführt und bleibt im Projekt referenziert. Sie wird automatisch im Bibliotheks-Repository in der Kategorie **Sonstige** installiert und weiterhin verwendet. Wenn die Bibliothek nicht die für eine Installation nötigen Projektinformationen (Titel, Version) mitbringt, werden Sie aufgefordert, diese im Dialog **Projektinformationen eingeben** nachzutragen.
- **Die folgende bereits installierte Bibliothek verwenden:** Wenn Sie die Optionen aktivieren, wird die eingebundene Bibliothek durch eine andere ersetzt, die bereits auf dem lokalen System installiert ist. Mit der Schaltfläche **Auswählen** öffnen Sie den Dialog **Auswählen...**. Hier können Sie die gewünschte Version einer der installierten Bibliothek auswählen. Dies entspricht der Konfiguration des Versions-Handlings im Dialog **Bibliothekseigenschaften**. Ein Sternchen („*“) bedeutet, dass immer die neueste Version der Bibliothek, die auf dem System verfügbar ist, im Projekt verwendet wird. Die Liste der verfügbaren Bibliotheken ist genauso strukturiert, wie im Dialog **Bibliotheks-Repository**. Sie können die Auflistung nach Firma und Kategorie sortieren.
- **Die Bibliothek ignorieren. Die Referenz wird im konvertierten Projekt nicht erscheinen:** Wenn Sie diese Option aktivieren, wird die Bibliotheksreferenz entfernt. Die Bibliothek ist dann nicht mehr im konvertierten Projekt eingebunden.
- **Dieses Mapping auch zukünftig verwenden, wenn diese Bibliothek auftritt:** Wenn Sie diese Option aktivieren, werden die hier im Dialog vorgenommen Einstellungen auch für künftige Projekt-Konvertierungen angewendet, sobald die betreffende Bibliothek referenziert ist.

Im konvertierten Projekt sind die Bibliotheksreferenzen im globalen Bibliotheksverwalter im Projektmappen-Explorer definiert. Nach Abschluss der Konvertierung der Bibliotheks-Referenzen wird die Projekt-Konvertierung wie oben beschrieben mit Dialog **Projekt öffnen** weitergeführt.

Generelle Informationen zur Bibliotheksverwaltung finden Sie in der PLC-Dokumentation im Abschnitt „Bibliotheken verwenden“.

Fall 3)

Wenn Sie ein TwinCAT 2.x PLC-Control-Projekt öffnen, das ein Gerät (Zielsystem) referenziert, für das noch kein „Konvertierungs-Mapping“ in den TwinCAT 2.x PLC-Control-Konverter-Optionen definiert ist, öffnet der Dialog **Gerätekonvertierung**, in dem Sie festlegen können, ob und wie die alten Geräte-Referenzen durch aktuellere ersetzt werden sollen. Das ursprünglich verwendete Gerät wird angezeigt. Wählen Sie eine der folgenden Optionen:

- **Das folgende bereits installierte Gerät verwenden:** Öffnen Sie mit der Schaltfläche **Auswählen** den Dialog **Zielsystem auswählen**, in dem Sie eines der aktuell auf dem System installierten Geräte auswählen können. Dieses Gerät wird dann anstelle des alten im **Projektmappen-Explorer** des konvertierten Projekts eingefügt. Aktivieren Sie Option **Wählen Sie ein Zielsystem aus...**, um eines der aufgelisteten Geräte auswählen zu können. Die Liste der verfügbaren Geräte ist genauso strukturiert wie im Dialog **Device-Repository**. Sie können die Auflistung nach Hersteller und Kategorie sortieren.
- **Das Gerät ignorieren. Alle applikationsspezifischen Objekte werden nicht verfügbar sein:** Wenn Sie diese Option aktiveren, wird im **Projektmappen-Explorer** des neuen Projekts kein Eintrag für das Gerät angelegt, d.h. das Gerät wird bei der Konvertierung ignoriert und auch applikationsspezifische Objekte wie z.B. die Taskkonfiguration werden nicht übernommen.
- **Diese Zuordnung für die Zukunft speichern:** Wenn Sie diese Option aktivieren, werden alle Einstellungen des Dialogs, d.h. das dargestellte „Konvertierungs-Mapping“ für das Gerät, in den TwinCAT 2.x PLC-Control-Konverter-Optionen gespeichert und für künftige Konvertierungen angewendet.

4. Das Projekt wurde nicht regulär beendet und Automatisch Speichern war aktiviert.

Wenn die Funktion **Automatisch Speichern** in den Optionen **Laden und Speichern** aktiviert war und TwinCAT 3 PLC, nach der letzten Änderung des Projekts ohne Speichern, nicht regulär beendet worden war, öffnet der Dialog **Auto Save Backup** zur Handhabung der Sicherungskopie.

5. Das Projekt ist schreibgeschützt.

Ist das Projekt, das geöffnet werden soll, schreibgeschützt, so werden Sie gefragt, ob Sie das Projekt in schreibgeschütztem Modus öffnen oder ob den Schreibschutz aufheben wollen.

6. Es handelt sich um eine Bibliothek, die in einem Bibliotheks-Repository installiert ist und aus diesem aufgerufen wird.

Wenn Sie ein Bibliotheksprojekt öffnen möchten, das in einem Bibliotheks-Repository installiert ist, wird eine Fehlermeldung ausgegeben. Ein Bibliotheksprojekt können Sie über diesen Pfad nicht öffnen. Nach Schließen des Dialogs mit **OK** erscheint der Projektname in der Titelzeile der Benutzeroberfläche. Ein Sternchen („*“) hinter dem Namen bedeutet, dass das Projekt seit dem letzten Speichern verändert wurde.

Siehe auch:

- Dokumentation PLC: TwinCAT-3-SPS-Projekt öffnen [▶ 58]
- Dokumentation PLC: TwinCAT-2-SPS-Projekt öffnen [▶ 58]

17.1.2 Befehl Projekt... (Neues TwinCAT-Projekt anlegen)

Symbol: 

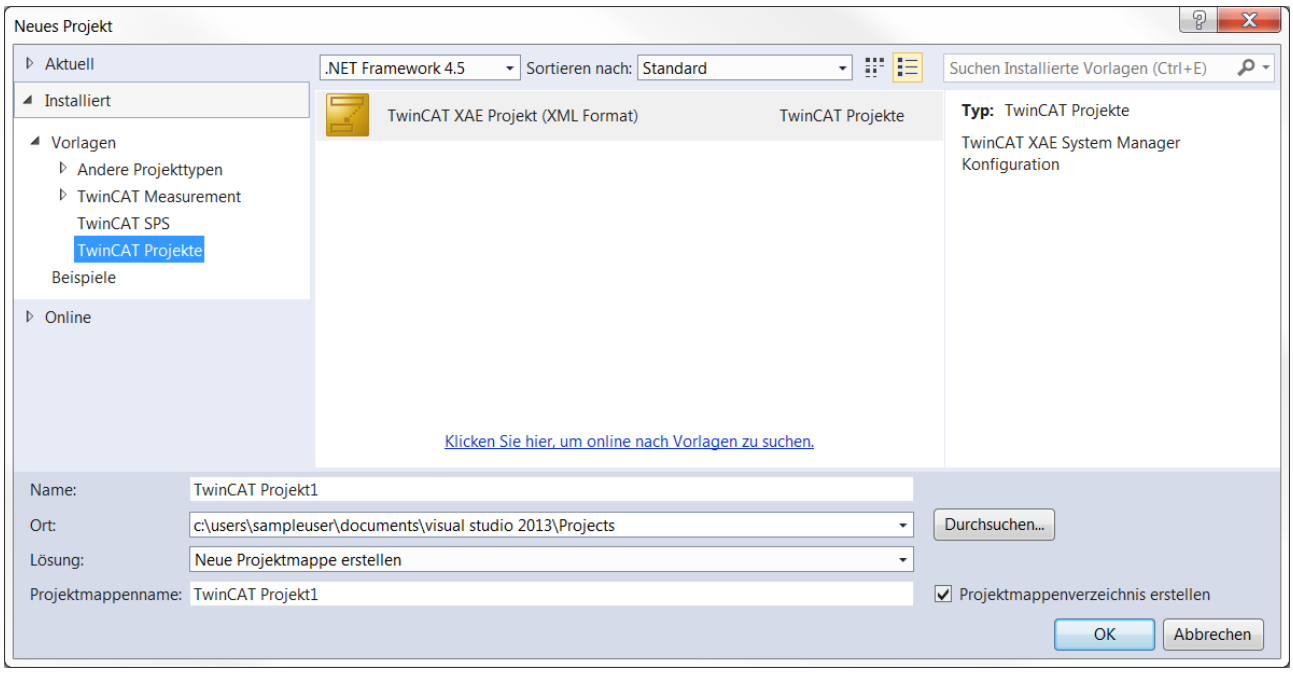
Tastaturkürzel: **[Strg] + [Umschalt] + [N]**

Funktion: Der Befehl öffnet den Dialog **Neues Projekt** zum Anlegen einer neuen TwinCAT-Projektdatei.

Aufruf: Menü **Datei > Neu**

Dialog Neues Projekt

Je nach gewählter Vorlage erhalten Sie ein Projekt, das automatisch mit einem bestimmten Umfang an Objekten ausgestattet ist.



Kategorien

Aktuell	Zeigt die zuletzt verwendete Projektvorlage an.
Installiert > Vorlagen	Zeigt die TwinCAT-Projektvorlagen an: • Andere Projekttypen • TwinCAT Measurement • TwinCAT SPS • TwinCAT Projekte
Online	Nicht relevant

Vorlagen

Kategorie Andere Projekttypen	
Visual Studio Projektmappen	Leere Visual Studio Projektmappe
Kategorie TwinCAT Measurement	
BodePlot	Bode Plot
Scope	Scope YT Project Scope YT NC Project Scope YT Project with Reporting Scope XY Project Scope XY Project with Reporting
Kategorie TwinCAT SPS	
	TwinCAT SPS Projekt
Kategorie TwinCAT Projekte	
	TwinCAT XAE Projekt

Name	Name des anzulegenden Projekts. Je nach Vorlage erscheint ein Standardname. Der numerische Zusatz stellt die Eindeutigkeit des Namens im Dateisystem sicher. Sie können den Dateinamen unter Berücksichtigung der Dateipfadkonventionen des Betriebssystems ändern. Punkte im Namen sind nicht erlaubt. TwinCAT fügt automatisch die der gewählten Vorlage entsprechende Dateierweiterung hinzu.
Ort	Speicherort für die neue Projektdatei. Die Schaltfläche **Durchsuchen…** öffnet einen Dialog zum Durchsuchen des Dateisystems. Das Kombinationsfeld zeigt die Historie früher eingegebener Pfade
Lösung	• Neue Projektmappe erstellen • Hinzufügen • In neuer Instanz erstellen
Projektmappenverzeichnis erstellen	☑ Projektmappenverzeichnis wird erstellt.
Projektmappenname	Name der Projektmappe. Standardmäßig wird der TwinCAT-Projektname automatisch übernommen.
OK	TwinCAT öffnet ein neues Projekt.

Siehe auch:

- Dokumentation PLC: Ihr erstes TwinCAT-3-SPS-Projekt [▶ 25]
- Dokumentation PLC: Standardprojekt anlegen [▶ 55]

17.1.3 Befehl Projekt/Projektmappe (Projekt/Projektmappe öffnen)

Symbol: 

Tastaturkürzel: **[Strg] + [Umschalt] + [O]**

Funktion: Der Befehl öffnet den Standarddialog zum Öffnen einer Datei. Hier können Sie das Dateisystem nach einer TwinCAT-Projektdatei durchsuchen und im Entwicklungssystem öffnen.

Aufruf: Menü **Datei > Öffnen**

Dialog Projekt öffnen

Dateityp	Auswahlliste zum Filtern des Dateityps • Dateien aller unterstützten Formate können geöffnet werden.
Optionen	• Hinzufügen (Verwenden Sie diese Option nur, um einer Projektmappe z. B. noch ein Measurement-Projekt hinzuzufügen. Verwenden Sie diese Option nicht, um einer Projektmappe mehrere TwinCAT-Projekte hinzuzufügen.) • Mappe schließen
Öffnen	TwinCAT öffnet die gewählte Projektdatei. Wenn erforderlich, wird sie zuvor konvertiert.

TwinCAT-Projektarchiv *.tszip

Inhalt vom *.tszip	Der Archivordner *.tszip enthält das TwinCAT-Projekt, welches archiviert wird.
Befehl zum Erstellen	Ein tszip-Archiv kann über den folgenden Befehl erstellt werden: Befehl Sichern <TwinCAT-Projektname> als Archiv... [▶ 1113]
Hinweis zu SPS-Projekten	Falls das TwinCAT-Projekt ein oder mehrere SPS-Projekte enthält, sind die Dateien und Ordner, die bezüglich dieser SPS-Projekte in dem Archivordner gespeichert werden, abhängig von den SPS-Projekteinstellungen des jeweiligen SPS-Projekts. Registerkarte Settings [▶ 969]

Siehe auch:

- Dokumentation PLC: Ihr erstes TwinCAT-3-SPS-Projekt [▶ 25]
- Dokumentation PLC: Standardprojekt anlegen [▶ 55]

17.1.4 Befehl Open Project from Target

Funktion: Der Befehl lädt ein Projekt vom Zielsystem.

Aufruf: Menü **Datei > Öffnen**

Voraussetzung: Der Netzwerkpfad zum Zielsystem muss konfiguriert sein.

Nachdem Sie den Befehl ausgeführt haben, öffnet sich eine Übersicht mit allen Geräten im Netzwerk. Aus dieser Übersicht wählen Sie das Zielsystem aus. Danach öffnet sich der Dialog **Wähle Verzeichnis für neue Arbeitsmappe**.

17.1.5 Befehl Neues Projekt... (Neues TwinCAT-Projekt hinzufügen)

Funktion: Der Befehl öffnet den Dialog **Neues Projekt** zum Anlegen einer weiteren TwinCAT-Projektdatei in der Projektmappe.

Aufruf: Menü **Datei > Hinzufügen**

Voraussetzung: Ein TwinCAT-Projekt ist geöffnet.

Verwenden Sie diesen Befehl nur, um einer Projektmappe z. B. noch ein Measurement-Projekt hinzuzufügen. Verwenden Sie diesen Befehl nicht, um einer Projektmappe mehrere TwinCAT-Projekte hinzuzufügen. Diese Funktion wird von TwinCAT aktuell noch nicht unterstützt.

17.1.6 Befehl Vorhandenes Projekt... (Vorhandenes TwinCAT-Projekt hinzufügen)

Funktion: Der Befehl öffnet den Dialog **Vorhandenes Projekt hinzufügen** zum Hinzufügen einer TwinCAT-Projektdatei in die Projektmappe.

Aufruf: Menü **Datei > Hinzufügen**

Voraussetzung: Ein TwinCAT-Projekt ist geöffnet.

Verwenden Sie diesen Befehl nur, um einer Projektmappe z. B. noch ein Measurement-Projekt hinzuzufügen. Verwenden Sie diesen Befehl nicht, um einer Projektmappe mehrere TwinCAT-Projekte hinzuzufügen. Diese Funktion wird von TwinCAT aktuell noch nicht unterstützt.

17.1.7 Befehl Zuletzt geöffnete Projekte und Projektmappen

Funktion: Der Befehl öffnet die Liste der zuletzt verwendeten Projekte, aus der Sie ein Projekt zum Öffnen auswählen können.

Aufruf: Menü **Datei**

Siehe auch:

- Dokumentation PLC: Projekt anlegen und konfigurieren [▶ 55]

17.1.8 Befehl Alles speichern

Symbol: 

Funktion: Der Befehl speichert alle Objekte des TwinCAT-Projekts.

Aufruf: Menü **Datei**, Standard Symbolleistenoptionen

17.1.9 Befehl Auswahl speichern/sichern

Symbol: 

Tastaturkürzel: **[Strg] + [S]**

Funktion: Der Befehl speichert die Projektmappe, das TwinCAT-Projekt, das TwinCAT-SPS-Projekt oder ein ausgewähltes SPS-Objekt (Main, GVL,…) unter dem aktuellen Namen.

Aufruf: Menü **Datei**, Standard Symbolleistenoptionen

Voraussetzung: Die Projektmappe, das TwinCAT-Projektobjekt, das SPS-Projektobjekt (<SPS-Projektname> Project) oder das zu speichernde SPS-Objekt ist im **Projektmappen-Explorer** ausgewählt.

Objekt speichern

Der Befehl speichert das Objekt unter dem aktuellen Namen. Wenn das Objekt seit dem letzten Speichern geändert wurde, ist das „Disketten“-Symbol am Objektsymbol rot und der Name in der Titelleiste des Editor des geöffneten Objekts mit einem Sternchen („*“) versehen.

Siehe auch:

- Befehl Sichern <TwinCAT-Projektname> als [▶ 910]

17.1.10 Befehl <Projektmappenname> speichern unter

Funktion: Der Befehl öffnet den Standarddialog zum Speichern einer Datei. Die Projektmappe kann unter dem gewünschten Speicherpfad ablegt werden. Standardmäßig ist der Dateityp UTF-8-Projektmappendatei (*.sln) ausgewählt.

Aufruf: Menü **Datei**

Voraussetzung: Die Projektmappe ist im **Projektmappen-Explorer** ausgewählt.

17.1.11 Befehl Sichern <TwinCAT-Projektname> als

Funktion: Der Befehl öffnet den Standarddialog zum Speichern einer Datei. Das Projekt kann unter dem gewünschten Speicherpfad und Dateityp ablegt werden. Standardmäßig ist der Dateityp TwinCAT XAE Project (*.tsproj) ausgewählt.

Aufruf: Menü **Datei**

Voraussetzung: Das TwinCAT-Projektobjekt ist im **Projektmappen-Explorer** ausgewählt.

Beachten Sie, dass bei diesem Speichervorgang beispielsweise nur die *.tsproj-Datei an einem anderen Speicherort erzeugt wird. Die referenzierten Projekte und die dort enthaltenen Objekte werden nicht an diesem neuen Speicherort gespeichert (z. B. ein SPS-Projekt, das in dem TwinCAT-3-Projekt eingebunden ist, und dessen Objekte).

Siehe auch:

- Befehl Auswahl speichern/sichern [▶ 910]
- Dokumentation PLC: Bibliothekserstellung

17.1.12 Befehl Sichern <SPS-Projektname> als

Funktion: Der Befehl öffnet einen Dialog, in dem ein Zielverzeichnis für die SPS-Projektdatei bestimmt werden kann. Die SPS-Projektobjekte und die .plcproj-Datei werden in dem ausgewählten Verzeichnis gespeichert.

Aufruf: Kontextmenü SPS-Projektobjekt

Voraussetzung: Das SPS-Projektobjekt (<SPS-Projektname>) ist im **Projektmappen-Explorer** ausgewählt.

17.1.13 Befehl Disassemblierungsdatei erzeugen

Funktion: Der Befehl erzeugt aus dem aktuellen Projekt eine Disassemblierungsdatei <project name>.asm und legt sie im Dateiverzeichnis im Projektordner ab.

Aufruf: Menü PLC > Disassemblierungsdatei erzeugen

17.1.14 Befehl Sende per E-Mail…/Send by E-Mail...

Symbol: 

Funktion: Der Befehl startet das aktuell im System eingestellte E-Mail-Programm und öffnet eine neue E-Mail. Diese enthält im Anhang die Archivdatei des ausgewählten Projekts.

Aufruf: Menü **Datei**, Kontextmenü

17.1.15 Befehl Projekt/Projektmappe schließen

Symbol: 

Funktion: Der Befehl schließt das gerade geöffnete Projekt. TwinCAT bleibt geöffnet.

Aufruf: Menü **Datei** oder implizit beim Öffnen eines neuen/anderen Projekts, während noch ein Projekt geöffnet ist.

Wenn das Projekt nicht-gespeicherte Änderungen enthält, erscheint eine Abfrage, ob das Projekt gespeichert werden soll.

Wenn Sie das Projekt noch nicht explizit gespeichert haben, erscheint eine Abfrage, ob Sie die Projektdateien löschen wollen.

Siehe auch:

- Dokumentation PLC: SPS-Projekt anlegen und konfigurieren [▶ 55]

17.1.16 Befehl Schließen

Funktion: Der Befehl schließt den geöffneten Editor.

Aufruf: Menü **Datei**

Voraussetzung: Der zu schließende Editor ist aktiv bzw. das Objekt im SPS-Projektbaum ist selektiert.

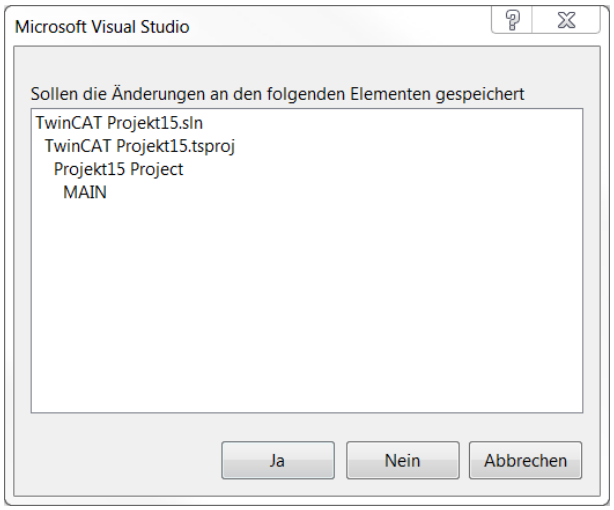
17.1.17 Befehl Beenden

Symbol: 

Tastaturkürzel: **[Alt] + [F4]**

Funktion: Der Befehl beendet das Programmiersystem. Wenn gerade ein Projekt geöffnet ist, das seit dem letzten Speichern verändert wurde, erscheint ein Dialog mit der Abfrage, ob das Projekt gespeichert werden soll.

Aufruf: Menü **Datei**



17.1.18 Befehl Seite einrichten…

Symbol: 

Funktion: Der Befehl öffnet den Dialog **Seiteneinstellungen** zur Konfiguration des Layouts für die Druckversion des Projektinhalts.

Aufruf: Menü **Datei**

Voraussetzung: Ein Editorfenster ist aktiv.

Siehe auch:

- Befehl Drucken [▶ 912]

17.1.19 Befehl Drucken

Symbol: 

Funktion: Der Befehl öffnet den Standarddialog von Windows zum Drucken von Dokumenten.

Aufruf: Menü **Datei**

Voraussetzung: Ein Editorfenster ist aktiv.

17.2 Bearbeiten

17.2.1 Standardbefehle

TwinCAT stellt Ihnen die folgenden Standardbefehle zur Verfügung:

- Rückgängig:

   , Tastaturkürzel: **[Strg] + [Z]**
- Wiederholen:

   , Tastaturkürzel: **[Strg] + [Y]**
- Ausschneiden:

   , Tastaturkürzel: **[Strg] + [X]**
- Kopieren:

   , Tastaturkürzel: **[Strg] + [C]**
- Einfügen:

   , Tastaturkürzel: **[Strg] + [V]**
- Löschen:

   , Tastaturkürzel: **[Entf]**
- Alles auswählen: Tastaturkürzel: **[Strg] + [A]**

Aufruf: Menü **Bearbeiten**, Kontextmenü SPS-Projektbaum, Kontextmenü Editorfenster

Der Befehl **Einfügen** wird nicht von allen Editoren unterstützt oder kann in manchen nur eingeschränkt verwendet werden. In grafischen Editoren wird der Befehl nur unterstützt, wenn durch das Einfügen ein korrektes Konstrukt entsteht.

Im SPS-Projektbaum bezieht sich der Befehl auf das gerade ausgewählte Objekt. Mehrfachauswahl ist möglich.

17.2.2 Befehl Entfernen

Tastaturkürzel: **[Entf]**

Funktion: Der Befehl entfernt das ausgewählte SPS-Objekt aus der Projektmappe. Im Projektverzeichnis bleibt das Objekt erhalten.

Aufruf: Kontextmenü SPS-Objekt

17.2.3 Befehl Alles auswählen

Tastaturkürzel: **[Strg+A]**

Funktion: Der Befehl wählt den gesamten Inhalt aus.

Aufruf: Menü **Bearbeiten**, Kontextmenü Editorfenster

17.2.4 Befehl Eingabehilfe

Symbol: 

Tastaturkürzel: **[F2]**

Funktion: Der Befehl öffnet den Dialog **Eingabehilfe**, der Sie bei der Auswahl eines an der aktuellen Cursorposition möglichen Programmierelements unterstützt.

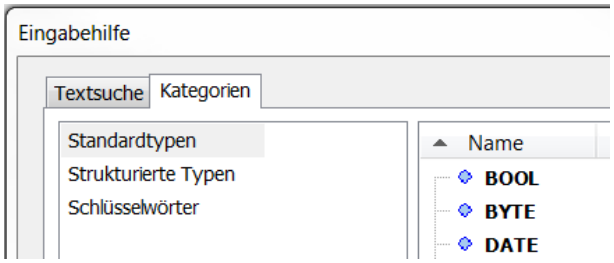
Aufruf: Menü **Bearbeiten**, Kontextmenü Editorfenster

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in einer Programmzeile.

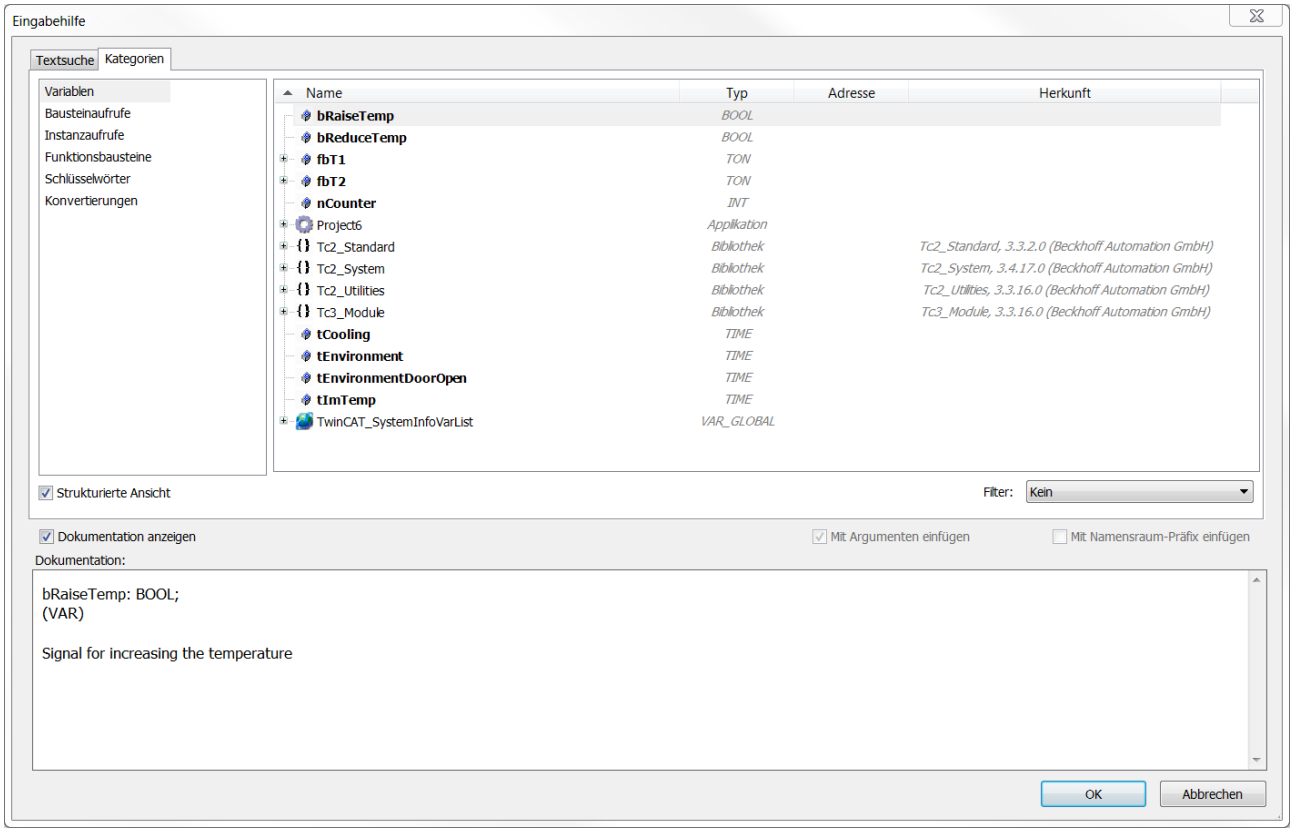
Dialog Eingabehilfe - Registerkarte Kategorien

Der Dialog bietet Ihnen alle Programmierelemente an, die Sie im Editor an der aktuellen Cursorposition einfügen können. Die Elemente sind nach Kategorien sortiert. Sie können in der Kategorie **Variablen** zusätzlich einen Filter für den Gültigkeitsbereich setzen, wie beispielsweise Lokale Variablen, Globale Variablen oder Konstanten.

Ausschnitt aus dem Dialog **Eingabehilfe** im Deklarationsteil des Editors:



Dialog **Eingabehilfe** im Implementierungsteil des Editors:



Strukturierte Ansicht	 : Die Elemente werden in einem Strukturbaum dargestellt. Sie können die Spalten Typ, Adresse und Herkunft mit einem Rechtsklick in den Spaltentitel in einem Untermenü aus- oder einblenden.  : Die Elemente werden in einer flachen Struktur dargestellt.
Filter	In dem Drop-down-Listenfeld können Sie einen zusätzlichen Filter für den Variablentyp setzen.
Dokumentation anzeigen	 : Eine Beschreibung des ausgewählten Elements wird angezeigt.
Mit Argumenten einfügen	 : TwinCAT fügt Elemente, die Argumente besitzen, wie zum Beispiel Funktionen, mit diesen Argumenten an der Cursorposition ein. Beispiel: Wenn Sie den Funktionsbaustein fb1, der eine Eingabevariable fb1_in und eine Ausgabevariable fb1_out enthält, “mit Argumenten“ einfügen, sieht dies im Editor folgendermaßen aus: `fb1(fb1_in:= , fb1_out=> )`
Mit Namensraum-Präfix einfügen	 : TwinCAT fügt das ausgewählte Element mit vorangestelltem Namensraum ein. Im Fall von Bibliotheksbausteinen können Sie das Auswahlkästchen nicht bedienen, wenn in den Eigenschaften der Bibliothek festgelegt ist, dass die Angabe des Namensraums zwingend ist.

Dialog Eingabehilfe - Registerkarte Textsuche

In der Registerkarte können Sie nach bestimmten Objekten suchen. Sobald Sie ein oder mehrere Zeichen in das Suchfeld eingeben, listet das Trefferfenster die Namen aller Objekte, deren Name diese Suchzeichenfolge enthält. Mit einem Doppelklick auf das gewünschte Objekt fügen Sie es an der aktuellen Cursorposition im Editor ein.

Filter	Einschränken der Suche auf eine bestimmte Variablenkategorie.

Siehe auch:

- Dokumentation PLC: Eingabeunterstützung nutzen [▶ 141]

17.2.5 Befehl Variable deklarieren

Funktion: Der Befehl öffnet den Dialog **Variable deklarieren**, der die Deklaration einer Variablen unterstützt.

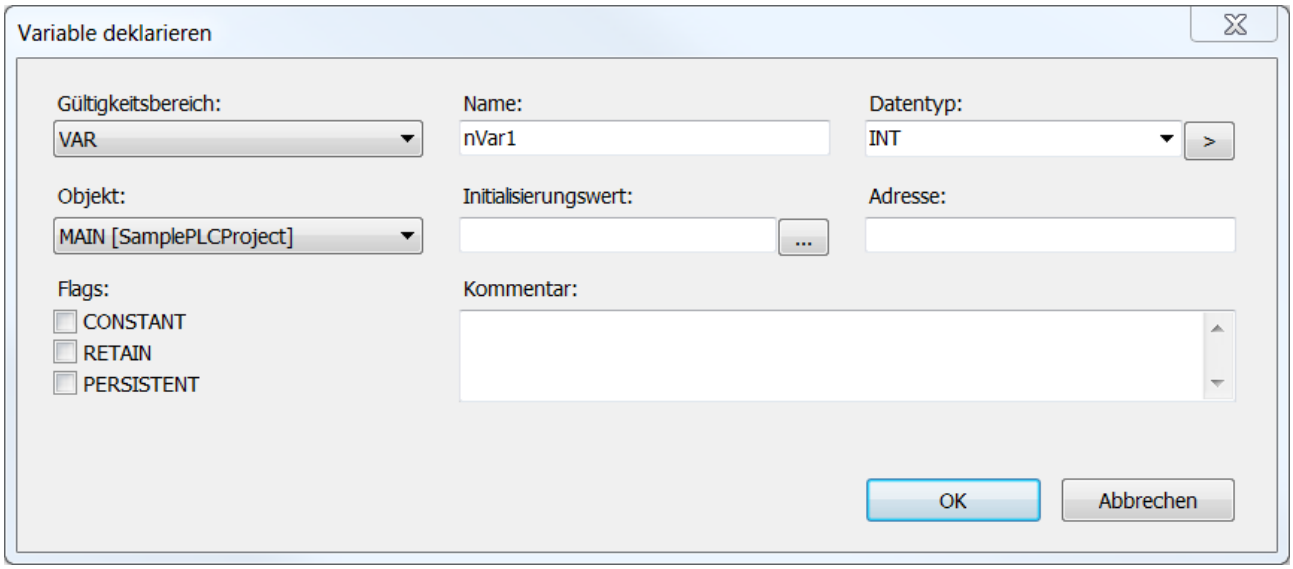
Aufruf: Menü **Bearbeiten**, Kontextmenü Editorfenster

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in einer Programmzeile.

Durch die Autodeklarationsfunktion erscheint der Dialog **Variable deklarieren** auch, wenn der Cursor im Implementierungsteil einer POU in einer Zeile steht, die den Namen einer nicht deklarierten Variablen enthält. Als Voraussetzung hierfür müssen Sie in den TwinCAT-Optionen die Option **Unbekannte Variablen automatisch deklarieren (AutoDeclare)** aktiviert haben
(**Extras > Optionen > TwinCAT > SPS Programmierumgebung > Intelligentes Kodieren**).

Durch die Smart-Tag-Funktion erscheint der Befehl **Variable deklarieren** auch, wenn Sie im Implementierungsteil des ST-Editors den Cursor auf eine Variable setzen, die nicht deklariert wurde, und anschließend auf  klicken (verfügbar ab Build 4026).

Dialog Variable deklarieren



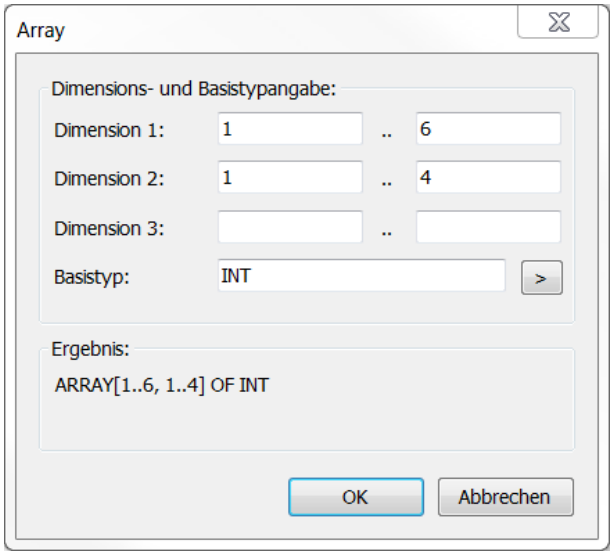
Gültigkeitsbereich	Gültigkeitsbereich der noch nicht deklarierten Variablen. Beispiel: `VAR` (Standardeinstellung bei lokaler Variable)
Name	Noch nicht deklarierter Variablenname Beispiel: `bVar`
Datentyp	: Listet die Standarddatentypen auf. : • Eingabehilfe: Öffnet den Dialog **Eingabehilfe** • Array-Assistent: Öffnet den Dialog **Array** Beispiel: `BOOL`
Objekt	Objekt, in dem die neue Variable deklariert wird. Standardmäßig steht hier das Objekt, das Sie gerade bearbeiten. : Listet die Objekte auf, in denen die Variable deklariert werden kann. Wenn für den ausgewählten Gültigkeitsbereich keine Objekte verfügbar sind, erscheint der Eintrag <Objekt anlegen>. Wenn Sie den Eintrag <Objekt anlegen> auswählen, öffnet sich der Dialog **Objekt hinzufügen** zur Erzeugung eines geeigneten Objekts.
Initialisierungswert	Wenn Sie keinen Initialisierungswert eingeben, wird die Variable automatisch initialisiert. : Öffnet den Dialog **Initialisierungswert**. Diese Vorgehensweise ist für die Initialisierung von strukturierten Variablen hilfreich. Beispiel: `FALSE`
Adresse	Speicheradresse (siehe Dokumentation PLC: Adressen [▶ 790]) Beispiel: %IX1.0
Flags	Attribut-Schlüsselwörter • CONSTANT: Schlüsselwort für eine Konstante • RETAIN: Schlüsselwort für eine remanente Variable • PERSISTENT: Schüsselwort für eine persistente Variable (strenger als RETAIN) Das ausgewählte Attribut-Schlüsselwort wird der Variablendeklaration hinzugefügt.
Kommentar	Im tabellarischen Deklarationseditor erfolgt die Anzeige des eingegebenen Kommentars in der Spalte Kommentar, im textuellen Deklarationseditor oberhalb der Variablendeklaration. Beispiel: `New variable`
Änderungen mit Hilfe von Refactoring anwenden	Die Option erscheint bei folgenden Gültigkeitsbereichen: • Eingabevariable (VAR_INPUT) • Ausgabevariable (VAR_OUTPUT) • Ein- und Ausgabevariable (VAR_IN_OUT) Die Option ist automatisch aktiviert, wenn in den TwinCAT-Optionen die Autodeklaration-Optionen **Beim Umbenennen von Variablen** und **Für das Hinzufügen oder Entfernen von Variablen, oder für das Ändern des Namensraums** aktiviert sind (**Extras > Optionen > TwinCAT > SPS Programmierumgebung > Refactoring**) (siehe Dialog Optionen - Refactoring [▶ 1024]). Wenn die Option aktiviert ist, wird die Variable beim Beenden des Dialogs noch nicht deklariert, sondern es öffnet sich zunächst der Dialog **Refactoring**, in dem Sie die Änderungen weiter bearbeiten können.

OK	Die Variable wird deklariert und erscheint in der Deklaration.
	Beispiel:
	`VAR`
	`    // New variable`
	`    bVar: BOOL := FALSE;`
	`END_VAR`

Siehe auch:

- Dokumentation PLC: Variablen deklarieren [▶ 68]

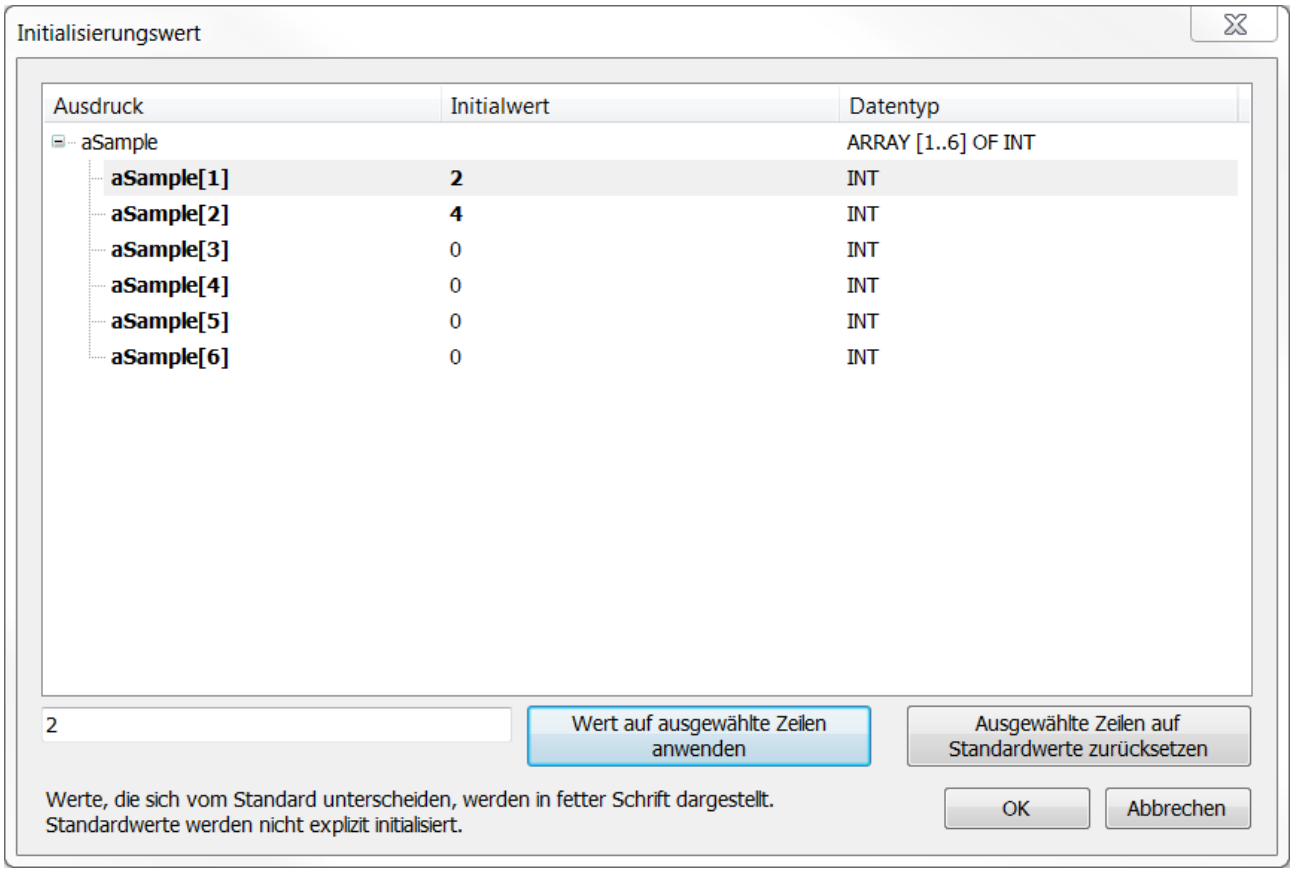
Dialog Array



Dimensions- und Basistyp-Angabe	Definition der Feldgrößen (Dimension) durch Eingeben der unteren und oberen Grenzen und des Basistyps des Arrays. Den Basistyp können Sie direkt eingeben oder mithilfe der Dialoge **Eingabehilfe** oder **Array**, wenn Sie die Schaltfläche > anklicken.
Ergebnis	Anzeige des definierten Arrays.

TwinCAT initialisiert Variablen nur neu, wenn Sie die Initialisierungswerte der Variablen geändert haben.

Dialog Initialisierungswert



Auflistung der Variablen mit Namen (Ausdruck), Initialisierungswert und Datentyp. Geänderte Initialisierungswerte werden fett dargestellt.	
Eingabefeld unterhalb der Liste	Eingabe eines Initialisierungswerts für die selektierte Variable(n).
Wert auf ausgewählte Zeilen anwenden	Änderung des Initialisierungswerts der selektierten Zeile(n) entsprechend dem Wert des Eingabefelds.
Ausgewählte Zeilen auf Standardwerte zurücksetzen	Herstellung der Standard-Initialisierungswerte.
OK	TwinCAT übernimmt die Initialisierungswerte in den Dialog **Variable deklarieren**.

Falls die über diesen Dialog zu initialisierende Variable eine Funktionsbaustein-Instanz mit erweiterter FB_Init-Methode ist, wird oberhalb der Tabelle Initialisierungswert eine weitere Tabelle angezeigt (siehe Dokumentation PLC: Methoden FB_init, FB_reinit und FB_exit [▶ 887]). In dieser Tabelle werden die zusätzlichen FB_Init-Parameter aufgelistet. Die Bedeutung und Bedienung entspricht im Wesentlichen der unteren Tabelle mit folgenden Unterschieden:

- Es müssen alle Variablen mit Initialisierungswerten belegt sein. Ansonsten ist OK nicht anwählbar.
- Bei komplexen Datentypen (Strukturen, Arrays) werden keine darin enthaltenen Komponenten angezeigt (Typ kann nicht aufgeklappt werden). In diesem Fall muss der komplexe Typ mit einer entsprechenden Variable initialisiert werden.

Bei so konfigurierten FB_Init-Parametern wird im Dialog **Variable deklarieren** ein entsprechendes Symbol hinter dem Initialisierungswert angezeigt.

Siehe auch:

- Dokumentation PLC: Eingabeunterstützung nutzen [▶ 141]

17.2.6 Befehl Zur Überwachungsliste hinzufügen

Symbol: 

Funktion: Der Befehl fügt die Variable, auf der gerade der Cursor steht, einer Überwachungsliste hinzu, die dem Online-Monitoring dient.

Aufruf: Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und der Cursor steht in einem Editor auf einer Variablen.

Der Befehl fügt die Variable in die gerade geöffnete Überwachungsliste ein. Wenn gerade keine Überwachungsliste geöffnet ist, fügt der Befehl die Variable in Überwachungsliste 1 ein und öffnet deren Ansicht.

Siehe auch:

- Dokumentation PLC: Überwachungslisten verwenden [▶ 239]
- Dokumentation PLC: Monitoring von Werten [▶ 234]

17.2.7 Befehl Aufrufbaum ausgeben

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Aufrufbaum**, die die Aufrufe des Bausteins sowie seine Aufrufer darstellt.

Aufruf: Kontextmenü Editorfenster

Voraussetzung: Ein Baustein ist im Editor geöffnet und der Cursor steht in einer Variablen.

17.2.8 Befehl Gehe zu

Funktion: Mit dem Befehl springt der Cursor zu einer definierten Zeile im Code.

Aufruf: Menü **Bearbeiten**

Voraussetzung: Ein Texteditor ist geöffnet und der Cursor steht in einer Programmzeile.

Der Befehl öffnet einen Dialog mit einem Eingabefeld **Zeilennummer**.

17.2.9 Befehl Gehe zur Definition

Symbol: 

Tastaturkürzel: **[F12]**

Funktion: Der Befehl zeigt die Definitionsstelle einer Variablen oder Funktion.

SPS-Editor

Aufruf: Kontextmenü Editorfenster

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht auf einer Variablen oder Funktion.

SPS-Prozessabbild

Aufruf: Kontextmenü Projektmappen-Explorer

Voraussetzung: Das Prozessabbild (Projektinstanz) ist ausgeklappt und der Cursor steht auf einer allokierten Variablen im Prozessabbild.

17.2.10 Befehl Gehe zur Instanz

Symbol: 

Funktion: Der Befehl öffnet die Instanz eines Funktionsbausteins in einem neuen Fenster.

Aufruf: Kontextmenü Editorfenster

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Eine POU ist im Editor geöffnet und der Cursor steht auf der Instanz eines Funktionsbausteins.

Der Befehl ist nicht verfügbar für temporäre Instanzen oder Instanzen aus übersetzten Bibliotheken.

17.2.11 Befehl Gehe zur Implementierung

Symbol: 

Funktion: Der Befehl öffnet die Online-Ansicht einer Methode in einem neuen Fenster.

Aufruf: Kontextmenü Editorfenster

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Eine POU ist im Editor geöffnet und der Cursor steht auf einem Methodenaufruf.

Verfügbar ab TC3.1 Build 4026

17.2.12 Befehl Gehe zum Verweis

Symbol: 

Funktion: Der Befehl öffnet im Onlinebetrieb die Deklarationsstelle der Variablen, die von dem gerade fokussierten Pointer referenziert wird.

Aufruf: Kontextmenü Editorfenster

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Eine POU ist im Editor geöffnet und der Cursor steht auf einem Pointer. Die referenzierte Variable liegt in einem statischen Speicher.

Wenn der Pointer nicht exakt auf den Beginn der Variablen zeigt, wird beim Wechseln zur Variablendeklaration eine entsprechende Meldung ausgegeben.

Verfügbar ab TC3.1 Build 4026

17.2.13 Befehl Verweise suchen

Tastaturkürzel: **[Umschalt+F12]**

Funktion: Der Befehl zeigt alle Verwendungsstellen einer Variablen in der Ansicht **Querverweisliste**.

Aufruf: Kontextmenü

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in einer Variablen oder die Ansicht **Querverweisliste** ist geöffnet und eine Variable ist im Feld **Name** angegeben.

Siehe auch:

- Dokumentation PLC: Verwendungsstellen mit der Querverweisliste finden [▶ 165]

17.2.14 Befehl Navigiere zu

Funktion: Der Befehl öffnet einen Dialog zur Auswahl eines Elements, das gezielt geöffnet werden soll.

Aufruf: Menü **Bearbeiten**

17.2.15 Befehl In Großbuchstaben umwandeln

Funktion: Der Befehl wandelt alle Kleinbuchstaben im selektierten Code in Großbuchstaben um.

Aufruf: Menü **Bearbeiten > Erweitert**

Voraussetzung: Eine POU ist im Editor geöffnet und Code ist selektiert.

17.2.16 Befehl In Kleinbuchstaben umwandeln

Funktion: Der Befehl wandelt alle Großbuchstaben im selektierten Code in Kleinbuchstaben um.

Aufruf: Menü **Bearbeiten > Erweitert**

Voraussetzung: Eine POU ist im Editor geöffnet und Code ist selektiert.

17.2.17 Befehl Leerstelle anzeigen

Symbol: 

Funktion: Der Befehl bewirkt, dass Steuerzeichen für Leerzeichen und Tabulatoren angezeigt werden.

Aufruf: Menü **Bearbeiten > Erweitert**

Voraussetzung: Eine POU ist im Editor geöffnet.

TwinCAT visualisiert Leerzeichen durch einen Punkt und Tabulatoren durch einen Pfeil.

17.2.18 Befehl Auswahl auskommentieren

Tastaturkürzel: [Ctrl+K] + [Ctrl+C]

Funktion: Der Befehl kommentiert einen selektierten Codebereich aus. Der Codebereich wird von der Kompilierung ausgeschlossen und hat keinen Einfluss auf die Programmausführung.

Aufruf: Menü **Bearbeiten > Erweitert**

Voraussetzung: Ein Baustein ist im Editor geöffnet und Code ist selektiert.

Sie verwenden den Befehl, um Kommentare für die Dokumentation eines Programms oder Programmabschnitts zu erstellen oder um einen Codebereich kurzfristig von der Kompilierung auszuschließen. Über den Befehl Auskommentierung der Auswahl aufheben [▶ 922] können Sie einen Kommentar aufheben und einen auskommentierten Codebereich wieder in die Programmausführung einbinden.

17.2.19 Befehl Auskommentierung der Auswahl aufheben

Tastaturkürzel: [Ctrl+K] + [Ctrl+U]

Funktion: Der Befehl hebt einen Kommentar auf und fügt einen auskommentierten Codebereich wieder in die Programmausführung ein.

Aufruf: Menü **Bearbeiten > Erweitert**

Voraussetzung: Ein Baustein ist im Editor geöffnet und Code ist selektiert, welcher zuvor über den Befehl Auswahl auskommentieren [▶ 922] auskommentiert wurde.

17.2.20 Befehl Schnellsuche (In Dateien suchen)

Symbol: 

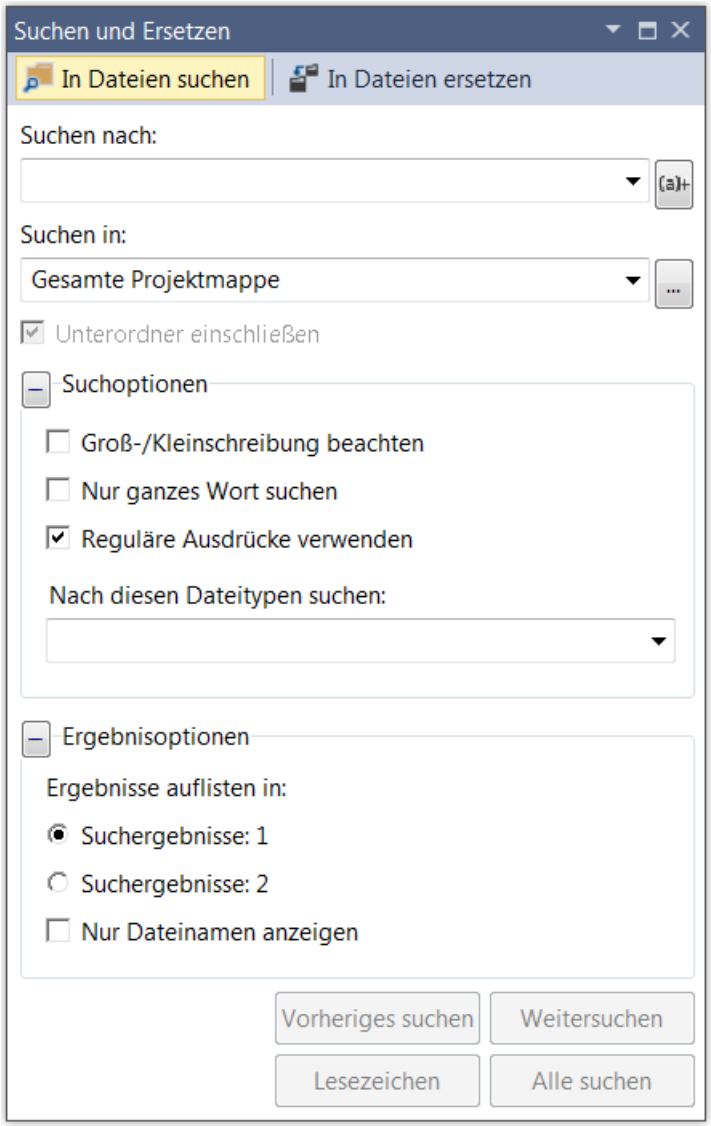
Tastaturkürzel: **[Strg] + [F]**

Funktion: Der Befehl durchsucht das Projekt oder Teile davon nach einer bestimmten Zeichenkette.

Aufruf: Menü **Bearbeiten > Suchen und Ersetzen**

Der Befehl öffnet den Dialog **Suchen und Ersetzen** (Schaltfläche **In Dateien suchen** ist aktiv), in dem die zu suchende Zeichenkette eingegeben und die Suchoptionen definiert werden.

Dialog Suchen und Ersetzen



In Dateien ersetzen	Wechselt zum Dialog **Suchen und Ersetzen** (Schaltfläche **In Dateien ersetzen** ist aktiv)
Suchen nach	Zeichenkette, nach der gesucht wird.
Suchen in	: Auswahlliste mit den Objekten, die durchsucht werden: Gesamte Projektmappe: Alle editierbaren Stellen in allen Objekten des Projekts werden durchsucht. Aktuelles Projekt: Alle geöffneten Dokumente: Alle Editoren, die gerade in einem Fenster geöffnet sind, werden durchsucht. Aktuelles Dokument: Nur der Editor, in dem gerade der Cursor steht, wird durchsucht. : Öffnet einen Dialog, in dem die zu durchsuchenden Objekte genauer definiert werden können.
Groß-/Kleinschreibung	: Die Suche beachtet Groß-/Kleinschreibung.
Nur ganzes Wort	: Nur Zeichenketten, die genau die gesuchte Zeichenkette wiedergeben, werden gefunden.
Nach diesen Dateitypen suchen	Drop-down-Liste zur Auswahl eines Dateityps
Reguläre Ausdrücke verwenden	Aktiviert die Schaltfläche , über die Sie Unterstützung bei der Eingabe von regulären Ausdrücken (Regular Expressions) erhalten. Diese Funktion wird für SPS-Editoren nicht unterstützt!
Nur Dateinamen anzeigen	: Nur Dateinamen werden angezeigt.
Weitersuchen	Starten der Suche. Das nächste Suchergebnis wird an dessen Position im betreffenden Editor angezeigt.
Alle Suchen	Der Befehl gibt alle Treffer der Suche im Meldungsfenster aus. Dabei werden das Objekt und die genaue Position des Treffers angezeigt. • (Dekl): Deklarationsteil des Objekts • (Impl): Implementierungsteil des Objekts Durch einen Doppelklick auf den Listeneintrag wird der Treffer im Editor angezeigt.

Siehe auch:

- Befehl Schnellersetzung (In Dateien ersetzen) [▶ 924]
- Dokumentation PLC: Projektweites Suchen und Ersetzen [▶ 168]

17.2.21 Befehl Schnellersetzung (In Dateien ersetzen)

Symbol:

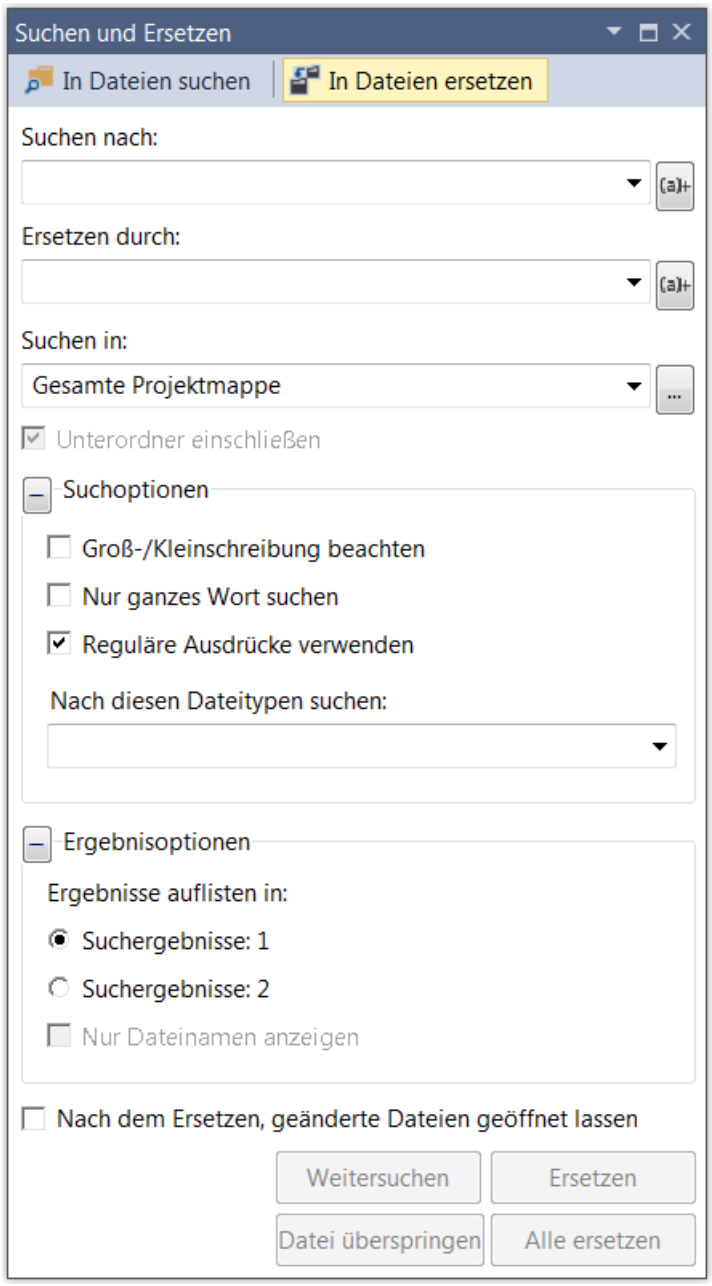
Tastaturkürzel: **[Strg] + [H]**

Funktion: Die Befehle durchsuchen das Projekt oder Teile davon nach einer bestimmten Zeichenkette und ersetzen diese.

Aufruf: Menü **Bearbeiten > Suchen und Ersetzen**

Der Befehl öffnet den Dialog **Suchen und Ersetzen** (Schaltfläche **In Dateien ersetzen** ist aktiv), in dem die zu ersetzende Zeichenkette und die neue Zeichenkette eingegeben und die Suchoptionen definiert werden.

Dialog Suchen und Ersetzen



Zusätzlich zu den Optionen des Dialogs „Suchen“ sind noch folgende Einstellungen möglich:

Ersetzen durch	Eingabefeld für die neue Zeichenkette.
Ersetzen	Damit wird jeweils die nächste gefundene Zeichenkette im Editor hervorgehoben und ersetzt (schrittweises Ersetzen).
Alle Ersetzen	Alle gefundenen Zeichenketten werden auf einmal ersetzt, ohne dass sie in den Editoren angezeigt werden.
Nach dem Ersetzen, geänderte Dateien geöffnet lassen	Die Editoren der gefundenen Objekte bleiben geöffnet.

Siehe auch:

- Befehl Schnellsuche (In Dateien suchen) [▶ 923]
- Dokumentation PLC: Projektweites Suchen und Ersetzen [▶ 168]

17.2.22 Befehl Schreibmodus umschalten

Tastaturkürzel: **[Einfg]**

Funktion: Dieser Befehl aktiviert den Überschreibmodus oder den Einfügemodus.

Aufruf: Doppelklick auf das Zeichen **[EINFG]** bzw. **[ÜB]** in der Status- und Informationsleiste

Voraussetzung: Ein Editorfenster ist aktiv.

Wenn der Überschreibmodus aktiviert ist, werden Zeichen vor dem Cursor bei der Eingabe neuer Zeichen überschrieben. Wenn der Einfügemodus aktiviert ist, werden Zeichen eingefügt und bestehende Zeichen vor dem Cursor bleiben erhalten.

17.2.23 Befehl Umbenennen

Funktion: Der Befehl ermöglicht das Umbenennen eines SPS-Objekts im **Projektmappen-Explorer**.

Aufruf: Kontextmenü SPS-Objekt

17.2.24 Befehl Objekt (offline) bearbeiten

Funktion: Der Befehl öffnet das Objekt offline in seinem Editor.

Aufruf: Menü **Projekt**, Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Ein Objekt im SPS-Projektbaum ist selektiert.

Somit können Sie auch im Onlinebetrieb das Objekt editieren. Die Änderung übertragen Sie anschließend mit dem Befehl **Online-Change** oder **Laden** auf die Steuerung.

Siehe auch:

- Befehl Online-Change [▶ 1000]
- Befehl Laden [▶ 999]

17.2.25 Befehl Refactoring - <Variable> umbenennen

Funktion: Der Befehl öffnet den Dialog **Umbenennen** zum projektweiten Umbenennen eines Objekts oder einer Variablen.

Aufruf: Kontextmenü SPS-Objekt, Kontextmenü Editorfenster > Refactoring

Voraussetzung: Im SPS-Projektbaum ist ein Objekt selektiert oder im Deklarationsteil eines Programmierobjekts ist der Cursor vor oder auf einem Variablenbezeichner positioniert.

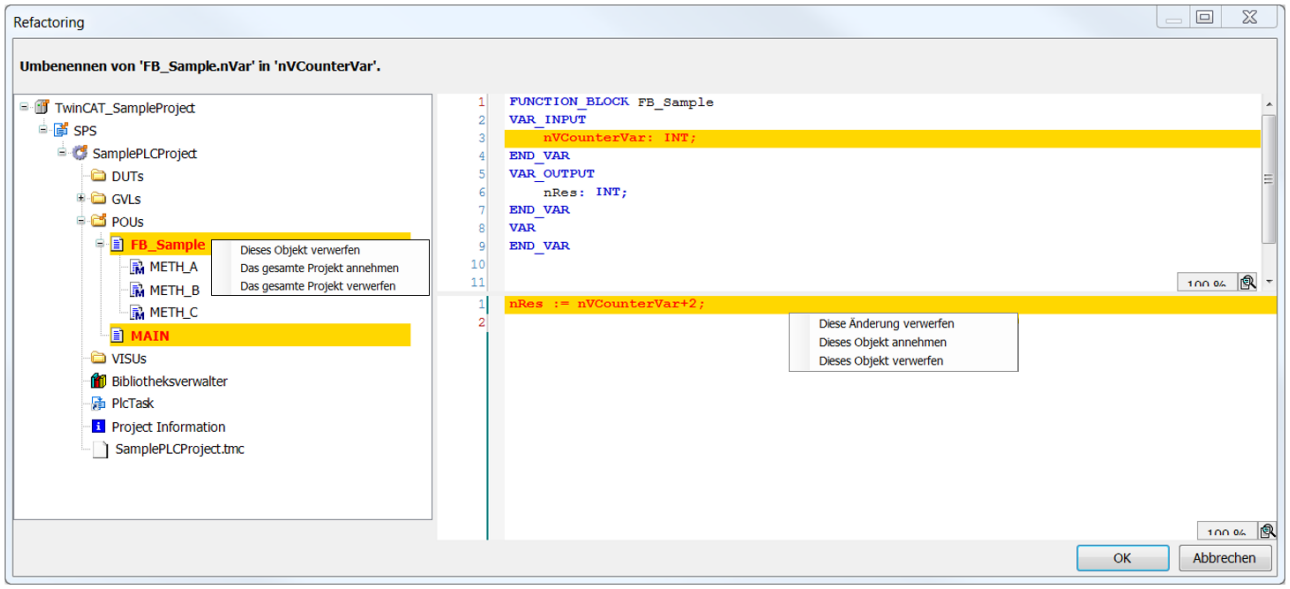
Sie können umbenennen:

- Variablen
- POUs
- GVLs
- Methoden
- Eigenschaften

Dialog Umbenennen

Aktueller Name	Name des Objekts oder der Variablen
Neuer Name	Eingabefeld für einen neuen Namen. Wenn der eingegebene Namen bereits existiert, meldet TwinCAT dies direkt unter diesem Eingabefeld.
OK	Aktivierbar, wenn Sie in **Neuer Name** einen gültigen Namen eingegeben haben. Öffnet den Dialog **Refactoring.** In beiden Fenstern sind die jeweiligen Objekte und Stellen farblich markiert. In beiden Fenstern können Sie für jede Verwendungsstelle festlegen, was gemacht werden soll. Dazu stehen Ihnen im Kontextmenü verschiedene Befehle zur Verfügung.

Dialog Refactoring



Der Dialog zeigt alle Verwendungsstellen innerhalb des Projekts. Die jeweiligen Objekte und Stellen sind farblich markiert.	
Linker Dialogteil	Navigationsbaum des Projekts mit dem jeweiligen Objekt.
Rechter Dialogteil	Anzeige der jeweiligen Stelle innerhalb eines Objekts, wo der aktuelle Name vorkommt.
Sie können in beiden Fenstern für jede Verwendungsstelle festlegen, was gemacht werden soll. Dazu stehen Ihnen im Kontextmenü die nachfolgend beschriebenen Befehle zur Verfügung.	
Diese Änderung verwerfen	Verwerfen der einzelnen Änderung im rechten Dialogteil.
Dieses Objekt annehmen	Annehmen aller Änderungen im betroffenen Objekt
Dieses Objekt verwerfen	Verwerfen aller Änderungen im betroffenen Objekt
Das gesamte Projekt annehmen	Annehmen aller Änderungen im Projekt
Das gesamte Projekt verwerfen	Verwerfen aller Änderungen im Projekt
TwinCAT stellt angenommene Änderungen mit gelbem Hintergrund und verworfene Änderungen mit grauem Hintergrund dar.	

Siehe auch:

- Dokumentation PLC: Refactoring [▶ 168]

17.2.26 Befehl Refactoring - Variable hinzufügen

Symbol: 

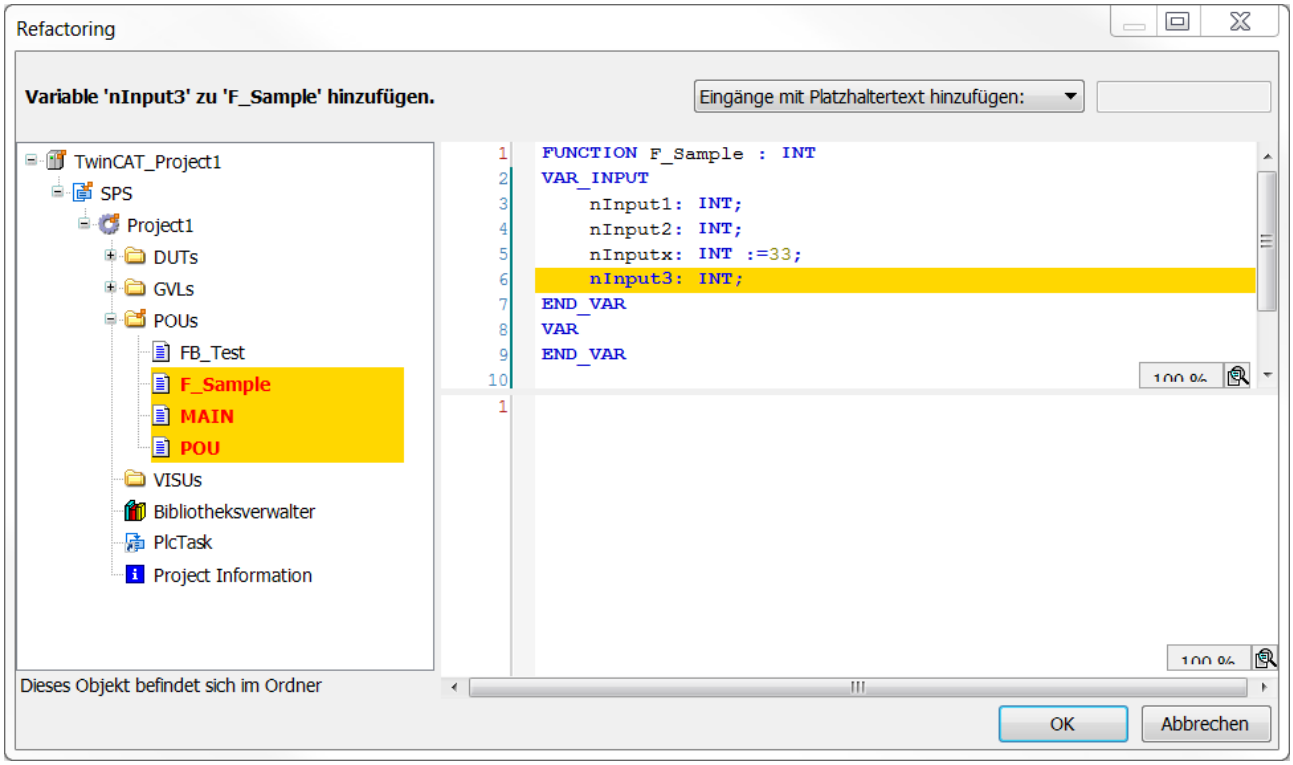
Funktion: Der Befehl ermöglicht das Deklarieren einer neuen Variablen in einer POU und die entsprechende automatische Aktualisierung an den Verwendungsstellen der POU.

Aufruf: Kontextmenü Editorfenster > Refactoring

Voraussetzungen: Der Fokus liegt im Deklarationsteil einer POU.

Der Befehl öffnet den Standarddialog zum Deklarieren der Variablen. Nach Schließen des Deklarationsdialogs mit **OK** erscheint der zweigeteilte Dialog **Refactoring**.

Dialog Refactoring



Linker Dialogteil	Navigationsbaum des Projekts. Farbliche Kennzeichnung der Bausteine, in denen die POU verwendet wird: rote Schrift und gelb hinterlegt. Nach einem Klick auf das POU-Objekt öffnet sich die Detailansicht im rechten Dialogteil.
Rechter Dialogteil	Deklarationsteil und Implementierung der POU, in deren Deklaration die Variable hinzugefügt wird. Farbliche Kennzeichnung der Änderungsstellen: Neu hinzugefügte Deklaration in blauer Schrift und gelb hinterlegt.

Bevor Sie entscheiden, welche Änderungen Sie an welchen Stellen übernehmen wollen, wählen Sie die gewünschte Option aus der Auswahlliste rechts oben:

Eingänge mit Platzhaltertext hinzufügen	Standardplatzhaltertext: _REFACTOR_; editierbar Der hier definierte Platzhaltertext erscheint an den Verwendungsstellen der neu hinzugefügten Variablen im Implementierungscode. Er dient der Suche nach den betroffenen Stellen.
Eingänge mit folgendem Wert hinzufügen	Initialisierungswert für die neue Variable

Im Kontextmenü der Änderungsstellen, sowohl im linken als auch rechten Teil des Dialogs gibt es Befehle zum Annehmen oder Ablehnen der Änderung(en). Sehen Sie hierzu auch die Beschreibung zu Befehl Refactoring - <Variable> umbenennen [▶ 926].

Beispiele:

1. Die Funktion F_Sample erhält über Refactoring eine neue Eingangsvariable „nInput3“ mit Initialisierungswert „1“. Die Änderung wirkt sich folgendermaßen aus:

Vorher:

```
F_Sample(nVarA + nVarB, 3, TRUE);
F_Sample(nInput1:= nVarA + nVarB, nInput2 :=3 , nInputx := TRUE);
```

Nachher:

```
F_Sample(nVarA + nVarB, 3, TRUE, nInput3 := 1);
F_Sample(nInput1:= nVarA + nVarB, nInput2 :=3 , nInputx := TRUE, nInput3 := 1);
```

2. Die Funktion F_Sample erhält über Refactoring eine neue Eingangsvariable „nInput3“ mit Platzhaltertext „_REFACTOR_“:

Vorher:

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInputx := TRUE);
F_Sample(nVarA + nVarB, 3, TRUE);
```

Nachher:

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInputx := TRUE, nInput3 := _REFACTOR_);
F_Sample(nVarA + nVarB, 3, TRUE, nInput3 := _REFACTOR_);
```

Siehe auch:

- Dokumentation PLC: Refactoring [▶ 168]
- Dokumentation PLC: Dialog Variable deklarieren [▶ 916]

17.2.27 Befehl Refactoring - <Variable> entfernen

Symbol: 

Funktion: Der Befehl entfernt eine Eingangs- oder Ausgangsvariable aus der POU und allen Verwendungsstellen der POU.

Aufruf: Kontextmenü Editorfenster > Refactoring

Voraussetzungen: Der Cursor steht im Bezeichner der zu entfernenden Variablen im Deklarationsteil der POU.

Der Befehl öffnet zunächst einen Dialog mit den Angaben zur gewünschten Entfernung. Nach dessen Bestätigung erscheint der Dialog **Refactoring**. Eine Beschreibung des Dialogs finden Sie im Abschnitt „Befehl Refactoring - Variable hinzufügen [▶ 928]“.

Wenn Sie die Änderungen in Dialog **Refactoring** akzeptieren, werden an den Verwendungsstellen der betroffenen POU die entsprechenden Eingangs- oder Ausgangsparameter gelöscht.

Im CFC wird nur die Verbindung des entfernten Eingangs oder Ausgangs zum Baustein entfernt. Der Eingang oder Ausgang selbst bleibt im Chart enthalten.

Beispiel in ST:

Sie entfernen in einer POU über **Refactoring** die Eingangsvariable „nInput4“. An den Verwendungsstellen erfolgt eine automatische Anpassung:

Vor der Entfernung:

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInput4 := 1, nInput5 := TRUE);
F_Sample(nVarA + nVarB, 3, 1, TRUE);
```

Nach der Entfernung:

```
F_Sample(nInput1 := nVarA + nVarB, nInput2 := 3, nInput5 := TRUE);
F_Sample(nVarA + nVarB, 3, TRUE);
```

Siehe auch:

- Dokumentation PLC: Refactoring [▶ 168]

17.2.28 Befehl Refactoring - Variablen neu ordnen

Symbol: 

Funktion: Der Befehl ermöglicht im Deklarationseditor das Ändern der Reihenfolge der Variablen im gerade fokussierten Gültigkeitsbereich VAR_INPUT, VAR_OUTPUT oder VAR_IN_OUT.

Aufruf: Kontextmenü des gerade fokussierten Gültigkeitsbereichs im Deklarationseditor

Voraussetzung: Der Fokus steht in der Deklaration eines der oben genannten Gültigkeitsbereiche und es ist mehr als eine Variable darin deklariert.

Der Befehl öffnet den Dialog **Neu ordnen** mit einer Liste aller Deklarationen des gerade fokussierten Gültigkeitsbereichs. Durch Ziehen mit der Maus können Sie eine selektierte Deklaration nach oben oder unten an eine andere Position ziehen.

Siehe auch:

- Dokumentation PLC: Variablen in der Deklaration neu anordnen [▶ 170]

17.3 Ansicht

17.3.1 Befehl Objekt öffnen

Symbol: 

Funktion: Der Befehl öffnet das Objekt in seinem Editor.

Aufruf: Menü **Ansicht**, Kontextmenü SPS-Objekt, Doppelklick auf das SPS-Objekt

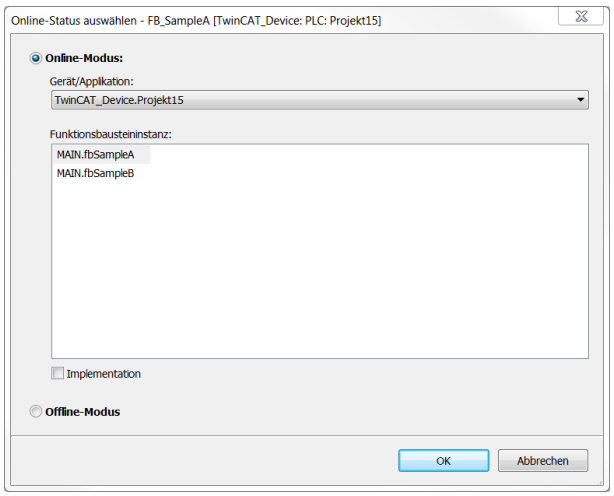
Voraussetzung: Ein Objekt im SPS-Projektbaum ist selektiert.

Im Onlinebetrieb öffnet der Dialog **Online Status auswählen**, in dem Sie auswählen können, in welcher Ansicht das Objekt geöffnet werden soll. Der Dialog öffnet nicht, wenn die die Auswahl des Objekts eindeutig ist. Dann wird das Objekt direkt im Online-Modus geöffnet.

Dialog Online Status auswählen

Funktion: Der Dialog legt fest, wie ein Objekt (Funktionsblock etc.) im Online-Modus geöffnet werden soll, das im Offline-Modus noch nicht geöffnet war**.** Sie können auswählen, ob eine Instanz oder die grundlegende Implementation des Objekts selbst geöffnet werden soll (und dies wahlweise im Online- oder Offline-Modus).

Voraussetzung: Im SPS-Projekt existieren mehrere Instanzen des ausgewählten Objekts.



Online-Modus	Aktivieren Sie die Auswahlfläche, um eine Ansicht im Online-Modus zu erhalten.
Gerät/Applikation	Zeigt die Anwendung (das Projekt), denen das Objekt zugeordnet ist.
Funktionsbausteininstanz	Wenn das Objekt ein Funktionsblock ist, erscheint eine Liste aller momentan in der Anwendung verwendeten Instanzen.
Implementation	Wählen Sie diese Option, um die Basisimplementation des Funktionsblocks unabhängig von der gewählten Instanz anzuzeigen. Diese Option hat keine Funktion für nicht-instanziierte Objekte.
Offline-Modus	Aktivieren Sie die Auswahlfläche, um eine Ansicht im Offline-Modus zu erhalten.

17.3.2 Befehl Textuelle Ansicht

Symbol: 

Funktion: Der Befehl öffnet den Deklarationseditor in textueller Ansicht.

Aufruf: Schaltfläche am rechten Rand des Editors

Siehe auch:

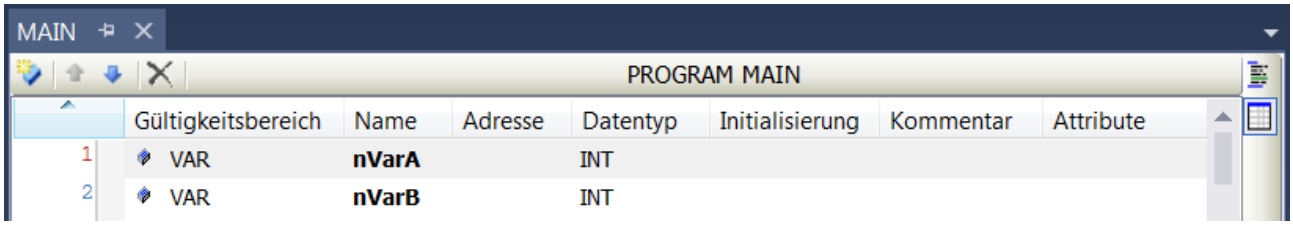
- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]

17.3.3 Befehl Tabellarische Ansicht

Symbol: 

Funktion: Der Befehl öffnet den Deklarationseditor in tabellarischer Ansicht.

Aufruf: Schaltfläche am rechten Rand des Editors



Siehe auch:

- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]

17.3.4 Befehl Ganzer Bildschirm

Symbol: 

Tastaturkürzel: **[Strg] + [Umschalt] + [F12]**

Funktion: Der Befehl schaltet die Anzeige von TwinCAT in den Vollbild-Modus.

Aufruf: Menü **Ansicht**

Wenn Sie den Befehl aktivieren, wird das Hauptfenster der TwinCAT-Benutzeroberfläche im Vollbildmodus angezeigt. Sie können zur vorher eingestellten Größe zurückkehren, wenn Sie den Befehl wieder deaktivieren.

17.3.5 Befehl Symbolleisten

Funktion: Der Befehl öffnet ein Menü zur Auswahl der angezeigten Symbolleisten.

Aufruf: Menü **Ansicht**, Kontextmenü Symbolleistenbereich

Markieren Sie in dem sich öffnenden Menü die Symbolleisten, die Sie ein- bzw. ausblenden möchten. Der Befehl funktioniert als Option, d. h. bei einer eingeblendeten Symbolleiste erscheint diese mit einem davor gesetzten Haken im Menü.

Siehe auch:

- Dokumentation TC3 User Interface: Symbolleisten anpassen

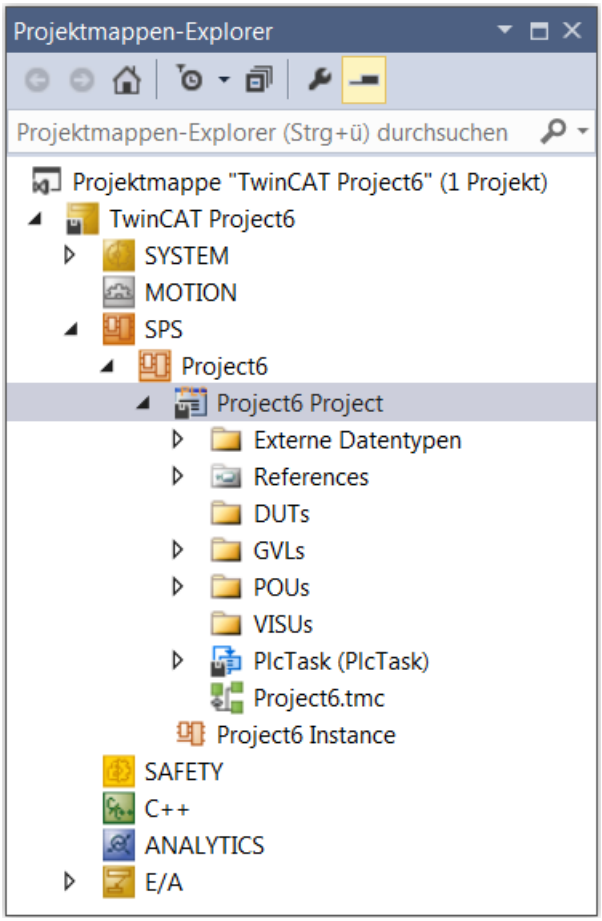
17.3.6 Befehl Projektmappen-Explorer

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Projektmappen-Explorer**.

Ansicht Projektmappen-Explorer

Die Ansicht **Projektmappen-Explorer** zeigt das TwinCAT-3-Projekt mit den dazugehörigen Projektelementen in strukturierter Form. Sie können in dieser Ansicht Objekte zum Bearbeiten und Konfigurieren öffnen.



17.3.7 Befehl Eigenschaftenfenster

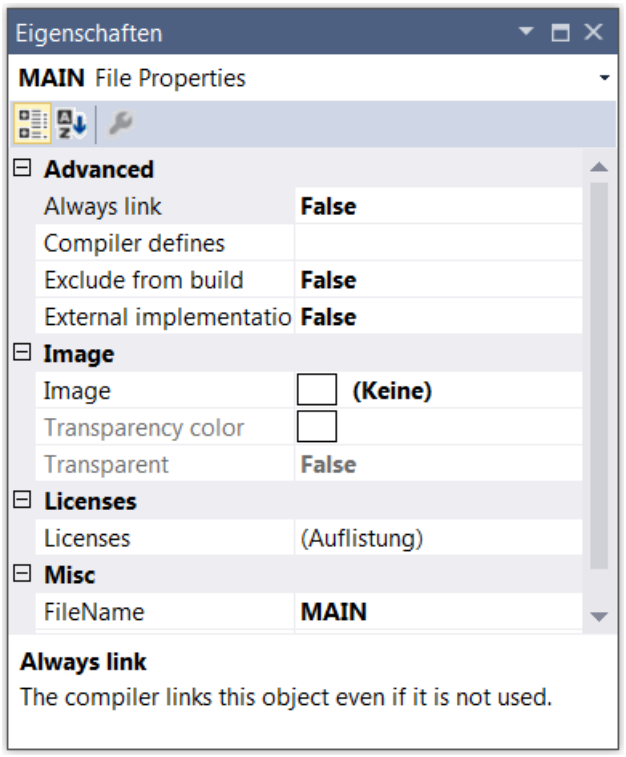
Symbol:

Funktion: Der Befehl öffnet die Ansicht **Eigenschaften**.

Aufruf: Menü **Ansicht**

Ansicht Eigenschaften

Die Ansicht **Eigenschaften** zeigt die Eigenschaften für das gerade im Projektmappen-Explorer selektierte Objekt. Standardmäßig werden die Elementeigenschaften nach Kategorien sortiert in einer Tabelle dargestellt. Durch einen Mausklick auf die Plus- bzw. Minuszeichen vor der Kategorie können Sie die zugehörigen Parameter einblenden oder ausblenden. Durch einen Mausklick auf das Wertefeld eines Parameters gelangen Sie in den Eingabemodus und können den Wert bzw. die Eigenschaft bearbeiten. Sie können die Ansicht der Eigenschaften filtern oder sortieren.



Sie können das Eigenschaftenfenster auch über das Kontextmenü eines Objekts im SPS-Projektbaum aufrufen. Eine Beschreibung des Befehls sowie verschiedener Objekteigenschaften finden Sie im Abschnitt Befehl Eigenschaften (Objekt) [▶ 942].

17.3.8 Befehl Werkzeugkasten

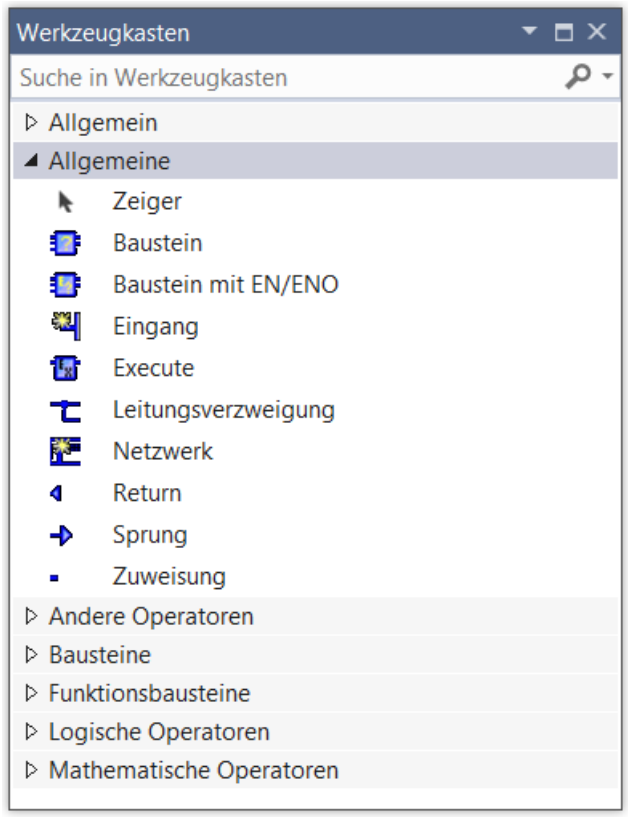
Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Werkzeugkasten**.

Aufruf: Menü **Ansicht**

Ansicht Werkzeugkasten

Die Ansicht **Werkzeugkasten** zeigt die vorhandenen „Werkzeuge“ für den gerade aktiven Editor. Diese Ansicht steht Ihnen standardmäßig bei einem grafischen Editor oder einer Visualisierung zur Verfügung. Sie enthält die grafischen Programmierelemente, die Sie über drag-and-drop in das Editorfenster ziehen können.



17.3.9 Befehl Fehlerliste

Symbol:

Funktion: Der Befehl öffnet die Ansicht **Fehlerliste**.

Aufruf: Menü **Ansicht**

Ansicht Fehlerliste

Die Ansicht **Fehlerliste** zeigt Fehler, Warnungen und Meldungen bezüglich Syntaxprüfung, Kompiliervorgang (Übersetzungsfehler, Code-Größe), Importprozessen oder Bibliotheksverwalter. Die Meldungen werden in Tabellenform ausgegeben.

Filter	Drop-down-Liste zur Auswahl des zu verwendenden Satz von Codedateien: Geöffnete Dokumente Aktuelles Projekt Aktuelles Dokument
Meldungskategorien • : Fehler • : Warnung • : Information	Klicken Sie auf das Symbol der Meldungskategorie, um Meldungen ein- oder auszublenden. Neben jedem Symbol zeigt TwinCAT die Anzahl der aufgetretenen Meldungen an.
Clear	Löscht die Meldungsanzeige
	Löscht die Filtereinstellungen der Fehlerliste
Schweregrad (Meldungskategorie)	Meldungstext mit dem verursachenden Objekt und der Position innerhalb des Objekts. Doppelklicken Sie auf einen Meldungseintrag in der Tabelle, um zur Quelltextposition zu gelangen.
Code	
Beschreibung	
Projekt	
Datei	
Zeile	

Befehle im Kontextmenü

Clear	Löscht die Meldungsanzeige
Spalten einblenden	Hinzufügen von weiteren Spalten, welche einen Fehler näher beschreiben können
Kopieren	Kopieren der ausgewählten Fehlermeldung
Nächster Fehler	Selektiert die nächste Meldung. Die Quelltextposition des nächsten Fehlers wird angezeigt.
Vorheriger Fehler	Selektiert die vorherige Meldung. Die Quelltextposition des vorherigen Fehlers wird angezeigt.

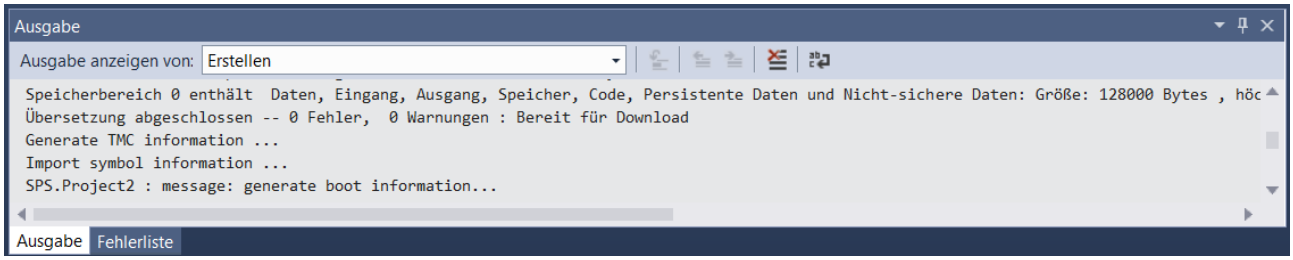
17.3.10 Befehl Ausgabe

Symbol:

Funktion: Der Befehl öffnet die Ansicht **Ausgabe**.

Aufruf: Menü **Ansicht**

Ansicht Ausgabe



Meldungskategorie	Die Meldungen sind nach Komponente oder Funktionalität kategorisiert und in einem Auswahldialog verfügbar. Filtern Sie die Anzeige der Meldungen durch die Wahl einer Kategorie.
 Meldung im Code suchen	Die Quelltextposition der Meldung wird angezeigt. Voraussetzung: Eine Meldung ist markiert.
 Gehe zur vorherigen Meldung	Die vorherige Meldung wird ausgewählt.
 Gehe zur nächsten Meldung	Die nächste Meldung wird ausgewählt.
 Alle löschen	Löscht alle Meldungen
 Zeilenumbruch umschalten	Zeilenumbruch wird ein bzw. ausgeschalten

Befehle im Kontextmenü

 Kopieren	Meldungstext wird kopiert
 Alle Löschen	Löscht alle Meldungen
 Gehe zu Speicherort	Die Quelltextposition der Meldung wird angezeigt. Voraussetzung: Eine Meldung ist markiert.
 Gehe zu nächster Position	Die nächste Meldung wird ausgewählt.
 Gehe zu vorheriger Position	Die vorherige Meldung wird ausgewählt.

17.4 Projekt

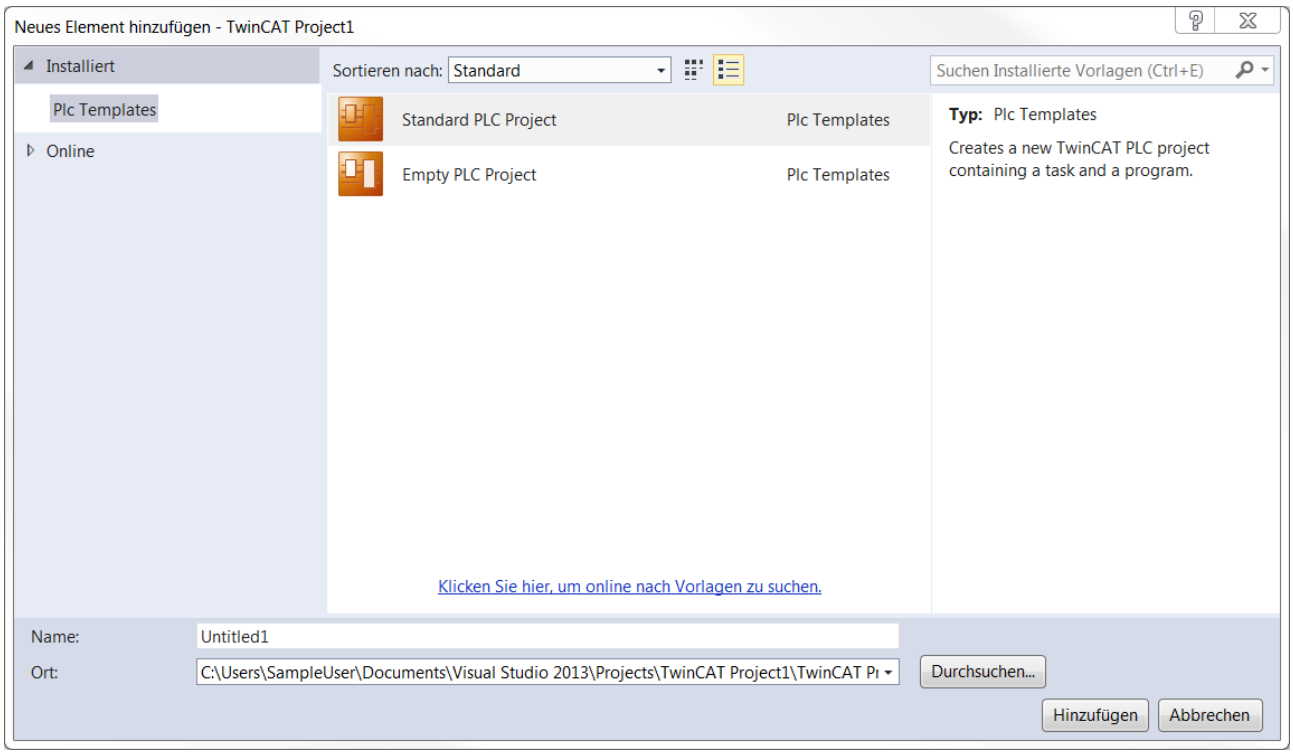
17.4.1 Befehl Neues Element hinzufügen (Projekt)

Symbol: 

Funktion: Der Befehl öffnet den Dialog **Neues Element hinzufügen**, über den eine neue SPS-Projektdatei angelegt werden kann. (Der Befehl steht nur bei ausgewähltem SPS-Knoten zur Verfügung.)

Aufruf: Menü **Projekt** oder Kontextmenü SPS-Objekt im **Projektmappen-Explorer**

Voraussetzung: Der SPS-Knoten ist im TwinCAT-Projektbaum selektiert.



Plc Templates	Wählen Sie eine der aufgelisteten Vorlagen aus. Die Vorlage bestimmt die Basiskonfiguration einer SPS-Projektdatei. Folgende Vorlagen stehen standardmäßig zur Verfügung: • Standard PLC Project: Erstellt ein neues TwinCAT-SPS-Projekt (*.project). Das Projekt wird von einem Wizard unterstützt und enthält einen Bibliotheksverwalter, ein Programm POU „MAIN“ und eine referenzierte Task. • Empty PLC Project: Erstellt ein „leeres“ TwinCAT-SPS-Projekt für Bibliotheksprojekte (*.library). Das Projekt enthält keine Objekte oder Geräte.
Name	Definieren Sie hier den Namen des neuen Projekts. Der Default-Name ist von der gewählten Vorlage (üblicherweise „Unbenannt<n>“) abhängig und enthält eine fortlaufende Nummer, um die Eindeutigkeit des Projektnamens im Dateisystem zu gewährleisten. Den vorgegebenen Namen können Sie unter Berücksichtigung der Dateipfad-Konventionen des lokalen Betriebssystems ändern. Eine Dateierweiterung (z. B. .project) kann hinzugefügt werden. Standardmäßig wird durch die gewählte Vorlage automatisch die passende Erweiterung ergänzt.
Ort	Legen Sie hier den Speicherort für die neue Projektdatei fest. Der standardmäßig eingestellte Speicherpfad ist von der gewählten Vorlage abhängig. Sie können entweder über die Schaltfläche **Durchsuchen**… den Standard-Browser öffnen und einen Speicherpfad bestimmen oder in der zugehörigen Drop-down-Liste aus der Historie bereits eingegebener Speicherpfade einen auswählen.
Hinzufügen	Wenn Sie auf **Hinzufügen** klicken, wird ein neues Projekt entsprechend der vorgenommenen Einstellungen angelegt. Wenn der Cursor auf ein Fehlersymbol gesetzt wird, gibt ein Tooltip Hinweise, wie Sie weiter vorgehen müssen. Wenn bereist ein anders SPS-Projekt geöffnet ist, öffnet sich ein Dialog, indem Sie gefragt werden, ob das Projekt gespeichert und geschlossen werden soll, bevor das neue Projekt geöffnet wird. Der Name des neuen Projekts wird anschließend in der Titelleiste des TwinCAT-XAE-Rahmenfensters angezeigt. Ein Sternchen („*“) hinter dem Namen bedeutet, dass das Projekt seit dem letzten Speichern verändert wurde.

Siehe auch:

- Dokumentation PLC: Ihr erstes TwinCAT-3-SPS-Projekt [▶ 25]
- Dokumentation PLC: SPS-Projekt anlegen und konfigurieren [▶ 55]

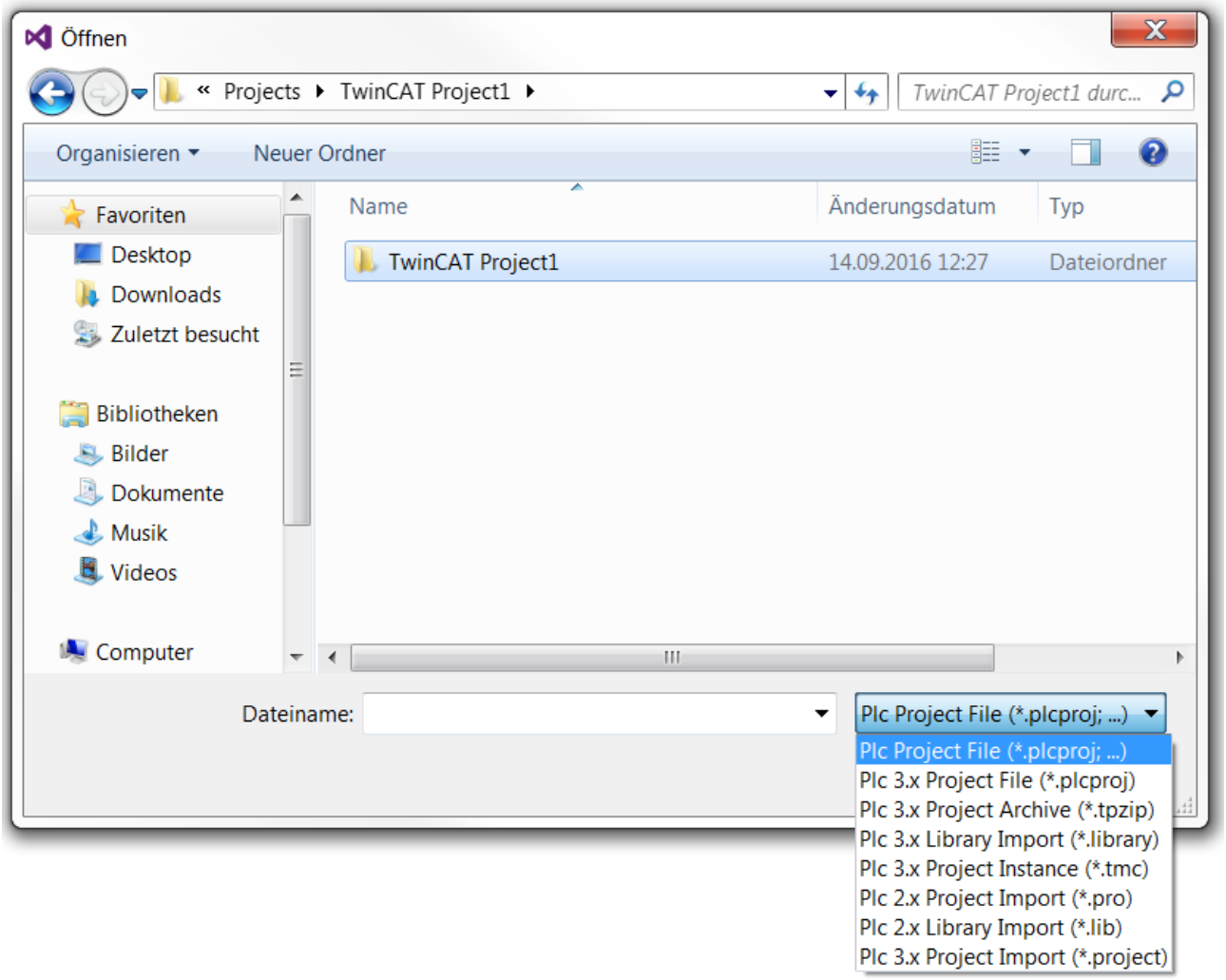
17.4.2 Befehl Vorhandenes Element hinzufügen (Projekt)

Symbol: 

Funktion: Der Befehl öffnet den Standard-Browserdialog, über den eine SPS-Projektdatei gesucht und im Programmiersystem geöffnet werden kann. Wenn ein entsprechender Konverter installiert ist, können auch Projekte in einem anderen Format geöffnet werden.

Aufruf: Menü **Projekt** oder Kontextmenü SPS-Objekt im **Projektmappen-Explorer**

Voraussetzung: Der SPS-Knoten ist im TwinCAT-Projektbaum selektiert.



Dateityp	Standardmäßig können Sie den Filter auf einen der folgenden Dateitypen setzen: • PLC 3.x Projektdatei (*.PLCproject): TwinCAT-3-PLC-Projekte mit der Erweiterung „.PLCproject“ • PLC 3.x Projektarchiv (*.tpzip): TwinCAT-3-PLC-Projektarchive mit der Erweiterung „.tpzip“ ◦ Siehe auch: Befehl Sichern <SPS-Projektname> als Archiv... [▶ 902] • PLC 3.x Bibliotheksimport (*.library): TwinCAT-3-PLC-Bibliotheken mit der Erweiterung „.library“, • PLC 2.x Projektdatei (*.pro): TwinCAT-2-PLC-Projekte mit der Erweiterung „.pro“ • PLC 2.x Bibliotheksimport (*.lib): TwinCAT-2-PLC-Bibliotheken mit der Erweiterung „.lib“ • PLC 3.x Projektimport (*.PLCproject): PLC-Projekte mit der Erweiterung „.project“
Öffnen	Die gewählte Projektdatei wird geöffnet bzw. konvertiert und dann geöffnet.

SPS-Projektarchiv *.tpzip

Inhalt vom *.tpzip	Der Archivordner *.tpzip enthält das SPS-Projekt, welches archiviert wird.
Befehl zum Erstellen	Ein tpzip-Archiv kann über den folgenden Befehl erstellt werden: Befehl Sichern <SPS-Projektname> als Archiv... [▶ 902]
Hinweis zu SPS-Projekten	Die Dateien und Ordner, die bezüglich des SPS-Projekts in dem Archivordner gespeichert werden, sind abhängig von den SPS-Projekteinstellungen dieses SPS-Projekts. Registerkarte Settings [▶ 969]

Mögliche Szenarien beim Öffnen eines SPS-Projekts

Folgende Szenarien sind beim Öffnen eines Projekts möglich:

1. Ein anderes Projekt ist noch geöffnet. [▶ 940]
2. Das Projekt wurde mit einer älteren TwinCAT-3-Version gespeichert. [▶ 940]
3. Das Projekt wurde nicht mit TwinCAT 3 gespeichert. [▶ 940]
4. Das Projekt wurde nicht regulär beendet und „Automatisch Speichern" war aktiviert. [▶ 942]
5. Das Projekt ist schreibgeschützt. [▶ 942]
6. Es handelt sich um eine Bibliothek, die in einem Bibliotheks-Repository installiert ist und aus diesem aufgerufen wird. [▶ 942]

1. Ein anderes Projekt ist noch geöffnet.

Sie werden gefragt, ob das andere Projekt gespeichert und geschlossen werden soll.

2. Das Projekt wurde mit einer älteren TwinCAT-3-Version gespeichert.

Wenn sich das Speicherformat unterscheidet, weil das geöffnete Projekt mit einer älteren Version von TwinCAT 3 gespeichert wurde, gibt es zwei Fälle:

- Wenn das Projekt nicht im Speicherformat des aktuell verwendeten Programmiersystems speicherbar ist, müssen Sie es aktualisieren, um weiter am Projekt arbeiten zu können. Der an dieser Stelle auftretende Ausdruck **Die durchgeführten Änderungen...** bezieht sich auf interne Aktionen verschiedener Komponenten während des Ladens des Projekts.
- Wenn das Projekt im bisherigen Speicherformat weiterhin speicherbar ist, können Sie entscheiden, ob das Speicherformat beibehalten oder aktualisiert werden soll. Wenn das Speicherformat beibehalten werden soll, muss mit möglichen Datenverlusten gerechnet werden. Wenn das Speicherformat aktualisiert werden soll, kann das Projekt nicht mehr mit der alten Version des Programmiersystems geöffnet werden.

Neben dem Speicherformat können sich auch die Versionen der explizit eingefügten Bibliotheken, des Visualisierungsprofils und die Compiler-Version des öffnenden Projekts von denen unterscheiden, die mit dem aktuellen Programmiersystem installierten wurden.

Wenn auf dem aktuellen Programmiersystem neuere Versionen installiert sind, öffnet sich automatisch der Dialog **Projektumgebung**, in dem Sie die Versionen aktualisieren können. Wenn an dieser Stelle noch keine Aktualisierung vorgenommen wird, kann dies jederzeit im Dialog **Optionen > Projektumgebung** nachgeholt werden.

Compiler-Version beachten

Wenn ein Projekt geöffnet wird, das mit einer älteren Version des Programmiersystems erstellt wurde und für das in den Projekteinstellungen die neueste Compiler-Version eingestellt ist, während im neuen Programmiersystem die Projektumgebungseinstellung für die Compiler-Version **Nicht aktualisieren** ist, dann wird die zuletzt im alten Projekt verwendete Compiler-Version weiter verwendet (nicht die „Aktuelle" in der neuen Umgebung).

3. Das Projekt wurde nicht mit TwinCAT 3 gespeichert.

Fall 1)

Wenn Sie beim Auswählen des zu öffnenden Projekts den Dateifilter gezielt setzen und ein entsprechender Konverter verfügbar ist, wird der Konverter automatisch verwendet und das Projekt in das aktuelle Format gebracht. Die Konvertierung läuft konverterspezifisch ab. In der Regel werden Sie aufgefordert, die Behandlung von eingebundenen Bibliotheken oder Geräte-Referenzen zu definieren.

TwinCAT-3-Konverter

Die Anpassung eines TwinCAT-PLC-Control-Projekts an die TwinCAT-3-Syntax kann beim Import nur gelingen, wenn der Konverter das Projekt fehlerfrei übersetzen kann.

Wenn Sie beim Auswählen des zu öffnenden Projekts den Dateityp **Alle Dateien** eingestellt haben, ist kein Konverter aktiviert und es öffnet sich der Dialog **Projekt konvertieren**. In dem Dialog müssen Sie die Konvertierung des Projekts durch Auswählen einer der Optionen explizit anstoßen.

- **In das aktuelle Format konvertieren:** Wählen Sie aus der Auswahlliste den Konverter, der verwendet werden soll (Anwendung zum Konvertieren). Nach dem Konvertieren kann das Projekt nicht mehr in der alten Version geöffnet werden.
- **Ein neues Projekt erzeugen und ein spezielles Gerät hinzufügen:** (Noch nicht implementiert)

TwinCAT 2.x PLC-Control-Projektoptionen

Der in den TwinCAT 2.x PLC-Control-Projektoptionen eingestellte Projektverzeichnispfad sowie die Projektinformationen werden in den Dialog **Projektinformationen** übernommen.

Fall 2)

Wenn im Projekt Bibliotheken eingebunden sind, für die noch kein „Konvertierungs-Mapping“ in den Bibliotheks-Optionen gespeichert ist, erscheint der Dialog **Konvertierung einer Bibliotheksreferenz**, in dem Sie definieren, wie diese Referenz konvertiert werden sollen:

- **Die Bibliothek ebenfalls konvertieren und installieren:** Wenn Sie diese Option aktivieren, wird die eingebundene Bibliothek in das neue Format übergeführt und bleibt im Projekt referenziert. Sie wird automatisch im Bibliotheks-Repository in der Kategorie **Sonstige** installiert und weiterhin verwendet. Wenn die Bibliothek nicht die für eine Installation nötigen Projektinformationen (Titel, Version) mitbringt, werden Sie aufgefordert, diese im Dialog **Projektinformationen eingeben** nachzutragen.
- **Die folgende bereits installierte Bibliothek verwenden:** Wenn Sie die Optionen aktivieren, wird die eingebundene Bibliothek durch eine andere ersetzt, die bereits auf dem lokalen System installiert ist. Mit der Schaltfläche **Auswählen** öffnen Sie den Dialog **Auswählen...**. Hier können Sie die gewünschte Version einer der installierten Bibliothek auswählen. Dies entspricht der Konfiguration des Versions-Handlings im Dialog **Bibliothekseigenschaften**. Ein Sternchen („*“) bedeutet, dass immer die neueste Version der Bibliothek, die auf dem System verfügbar ist, im Projekt verwendet wird. Die Liste der verfügbaren Bibliotheken ist genauso strukturiert, wie im Dialog **Bibliotheks-Repository**. Sie können die Auflistung nach Firma und Kategorie sortieren.
- **Die Bibliothek ignorieren. Die Referenz wird im konvertierten Projekt nicht erscheinen:** Wenn Sie diese Option aktivieren, wird die Bibliotheksreferenz entfernt. Die Bibliothek ist dann nicht mehr im konvertierten Projekt eingebunden.
- **Dieses Mapping auch zukünftig verwenden, wenn diese Bibliothek auftritt:** Wenn Sie diese Option aktivieren, werden die hier im Dialog vorgenommen Einstellungen auch für künftige Projekt-Konvertierungen angewendet, sobald die betreffende Bibliothek referenziert ist.

Im konvertierten Projekt sind die Bibliotheksreferenzen im globalen Bibliotheksverwalter im Projektmappen-Explorer definiert. Nach Abschluss der Konvertierung der Bibliotheks-Referenzen wird die Projekt-Konvertierung wie oben beschrieben mit Dialog **Projekt öffnen** weitergeführt.

Generelle Informationen zur Bibliotheksverwaltung finden Sie in der PLC-Dokumentation im Abschnitt „Bibliotheken verwenden“.

Fall 3)

Wenn Sie ein TwinCAT 2.x PLC-Control-Projekt öffnen, das ein Gerät (Zielsystem) referenziert, für das noch kein „Konvertierungs-Mapping“ in den TwinCAT 2.x PLC-Control-Konverter-Optionen definiert ist, öffnet der Dialog **Gerätekonvertierung**, in dem Sie festlegen können, ob und wie die alten Geräte-Referenzen durch aktuellere ersetzt werden sollen. Das ursprünglich verwendete Gerät wird angezeigt. Wählen Sie eine der folgenden Optionen:

- **Das folgende bereits installierte Gerät verwenden:** Öffnen Sie mit der Schaltfläche **Auswählen** den Dialog **Zielsystem auswählen**, in dem Sie eines der aktuell auf dem System installierten Geräte auswählen können. Dieses Gerät wird dann anstelle des alten im **Projektmappen-Explorer** des konvertierten Projekts eingefügt. Aktivieren Sie Option **Wählen Sie ein Zielsystem aus...**, um eines der aufgelisteten Geräte auswählen zu können. Die Liste der verfügbaren Geräte ist genauso strukturiert wie im Dialog **Device-Repository**. Sie können die Auflistung nach Hersteller und Kategorie sortieren.
- **Das Gerät ignorieren. Alle applikationsspezifischen Objekte werden nicht verfügbar sein:** Wenn Sie diese Option aktiveren, wird im **Projektmappen-Explorer** des neuen Projekts kein Eintrag für das Gerät angelegt, d.h. das Gerät wird bei der Konvertierung ignoriert und auch applikationsspezifische Objekte wie z.B. die Taskkonfiguration werden nicht übernommen.
- **Diese Zuordnung für die Zukunft speichern:** Wenn Sie diese Option aktivieren, werden alle Einstellungen des Dialogs, d.h. das dargestellte „Konvertierungs-Mapping“ für das Gerät, in den TwinCAT 2.x PLC-Control-Konverter-Optionen gespeichert und für künftige Konvertierungen angewendet.

4. Das Projekt wurde nicht regulär beendet und Automatisch Speichern war aktiviert.

Wenn die Funktion **Automatisch Speichern** in den Optionen **Laden und Speichern** aktiviert war und TwinCAT 3 PLC, nach der letzten Änderung des Projekts ohne Speichern, nicht regulär beendet worden war, öffnet der Dialog **Auto Save Backup** zur Handhabung der Sicherungskopie.

5. Das Projekt ist schreibgeschützt.

Ist das Projekt, das geöffnet werden soll, schreibgeschützt, so werden Sie gefragt, ob Sie das Projekt in schreibgeschütztem Modus öffnen oder ob den Schreibschutz aufheben wollen.

6. Es handelt sich um eine Bibliothek, die in einem Bibliotheks-Repository installiert ist und aus diesem aufgerufen wird.

Wenn Sie ein Bibliotheksprojekt öffnen möchten, das in einem Bibliotheks-Repository installiert ist, wird eine Fehlermeldung ausgegeben. Ein Bibliotheksprojekt können Sie über diesen Pfad nicht öffnen. Nach Schließen des Dialogs mit **OK** erscheint der Projektname in der Titelzeile der Benutzeroberfläche. Ein Sternchen („*“) hinter dem Namen bedeutet, dass das Projekt seit dem letzten Speichern verändert wurde.

Siehe auch:

- Dokumentation PLC: TwinCAT-3-SPS-Projekt öffnen [▶ 58]
- Dokumentation PLC: TwinCAT-2-SPS-Projekt öffnen [▶ 58]

17.4.3 Befehl Eigenschaften (Objekt)

Funktion: Der Befehl aktiviert die Ansicht **Eigenschaften**, die allgemeine Informationen zum gerade selektierten Objekt anzeigt.

Aufruf: Kontextmenü SPS-Objekt

Voraussetzung: Ein Objekt im SPS-Projektbaum ist selektiert.

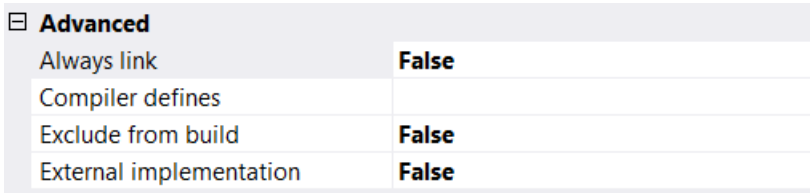
In Abhängigkeit vom gerade selektierten Objekt, werden folgende Eigenschaftsbereiche angezeigt:

- Erweitert [▶ 943] (Übersetzungseinstellungen)
- Image [▶ 943]
- Licenses [▶ 943]
- Allgemein [▶ 943] (Objektname, Objektpfad)
- AS-Einstellungen [▶ 946] (Flags für Ablaufsprache)
- Befehl Eigenschaften (Objekt) [▶ 946] (Ausführungsreihenfolge-Modus)

Die speziellen Visualisierungseigenschaften sind unter Visualisierungsobjekt [▶ 422] und die speziellen Bibliotheks- und Platzhaltereigenschaften sind unter Befehl Eigenschaften [▶ 377] dokumentiert.

Erweitert

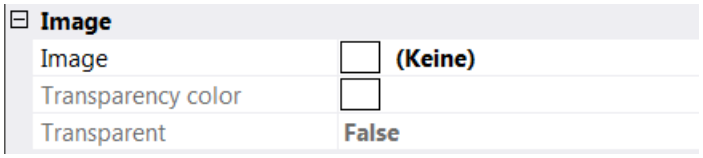
Dieser Bereich zeigt die Einstellungen bezüglich der Kompilation (Übersetzen) des Objekts.



Immer binden	True: Das Objekt ist beim Compiler markiert und damit immer in der Compile-Information enthalten. Es wird somit immer kompiliert und auf die SPS geladen. Diese Option wird dann relevant, wenn das Objekt unterhalb einer Applikation liegt oder über Bibliotheken referenziert wird, die ebenfalls unterhalb einer Applikation liegen. Die Compile-Information wird ebenfalls als Basis für die auswählbaren Variablen der Symbolkonfiguration verwendet. Alternativ kann mittels des Pragmas {attribute 'linkalways'} [▶ 846] der Compiler angewiesen werden, ein Objekt immer einzubinden.
Compilerdefinitionen	Die hier eingetragenen Compilerdefinitionen werden nicht ausgewertet. Wenn Sie Compilerdefinitionen verwenden möchten, tragen Sie diese in den SPS-Projekteigenschaften ein. Siehe: • Befehl Eigenschaften (SPS-Projekt) > Kategorie Übersetzen [▶ 951] • Dokumentation PLC: Referenz Programmierung > Pragmas > Bedingte Pragmas [▶ 875]
Vom Übersetzen ausschließen	True: Das Objekt wird beim nächsten Übersetzungslauf nicht berücksichtigt.
Externe Implementierung	(Spätes Binden im Laufzeitsystem) Die Verwendung dieser Funktionalität ist lediglich in speziellen Konstellationen möglich. In aller Regel können Sie diese Option ignorieren. True: Beim Übersetzen des Projekts wird für dieses Objekt kein Code generiert. Das Objekt wird erst gelinkt, wenn das Projekt auf das Zielsystem geladen wird, vorausgesetzt, es ist dort vorhanden (im SPS-Laufzeitsystem oder in einem anderen Echtzeitmodul).

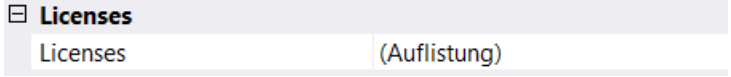
Image

In diesem Bereich können Sie dem Objekt ein Bild zuweisen, mit dem es in der grafischen Ansicht des Bibliotheksverwalters und im Werkzeugkasten des FUP/KOP/IL-Editors dargestellt wird. Transparenz der Abbildung kann durch Auswahl einer Farbe erreicht werden, die dann transparent dargestellt wird. Wenn Sie die Option **Transparenzfarbe** aktivieren, können Sie über die rechteckige Schaltfläche rechts davon den Standarddialog zur Auswahl einer Farbe öffnen.



Licenses

In diesem Bereich finden Sie eine Auflistung der Lizenzen zum Objekt.



Allgemein

In diesem Bereich finden Sie allgemeine Informationen zum selektierten Objekt.

FileName	Datei-/Objektname
FullPath	Speicherpfad/-ort des Objekts (an dieser Stelle nicht editierbar)
Version	Dateiversion, Werte: • 1.1.0.1: Diese Dateiversion wird verwendet, wenn das Objekt im Speicherformat XML gespeichert wird. • 1.2.0.0: Diese Dateiversion wird verwendet, wenn das Objekt im Speicherformat Base64 gespeichert wird. Bitte beachten Sie, dass Objekte mit der Dateiversion 1.2.0.0 (oder größer) nicht mit Engineering-Versionen < TC3.1.4024 geladen werden können! (an dieser Stelle nicht editierbar, indirekt konfigurierbar über das Speicherformat, siehe Eigenschaft **Format**)

Engineering-Inkompatibilität von Dateiversion 1.2.0.0 (oder größer) mit TwinCAT 3.1 < Build 4024

Beachten Sie, dass Objekte, die mit der Dateiversion 1.2.0.0 (oder größer) gespeichert werden, nicht mit Engineering-Versionen < TwinCAT 3.1.4024 geladen werden können!

Da ein Objekt bei Verwendung des optionalen Speicherformats „Base64“ automatisch mit der Dateiversion 1.2.0.0 gespeichert wird, können Objekte mit Base64-Speicherformat folglich nicht mit Engineering-Versionen < TwinCAT 3.1.4024 geladen werden.

Falls ein SPS-Projekt sowohl Objekte mit der Dateiversion 1.1.0.1 als auch Objekte mit der Dateiversion 1.2.0.0 enthält, werden die 1.1.0.1-Objekte mit einer Engineering-Version < TwinCAT 3.1.4024 weiterhin geladen. Lediglich die Objekte mit der Dateiversion 1.2.0.0 werden nicht geladen.

Die Dateiversion einer Datei, die mit der Dateiversion 1.2.0.0 gespeichert wurde, kann mit XAE-Versionen >= TwinCAT 3.1.4024 nachträglich wieder auf 1.1.0.1 gesetzt werden.

Optionen

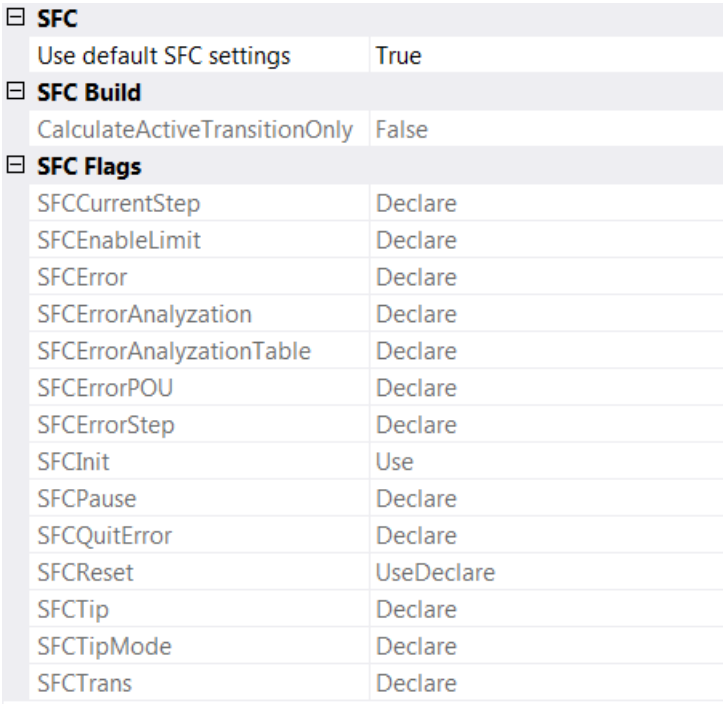
In diesem Bereich finden Sie einige Optionen, die für SPS-Objekte konfiguriert werden können.

Format	**Individuelle Einstellungsmöglichkeit des Speicherformats:** Das Speicherformat eines Objekts kann für die unten genannten Objekttypen an dieser Stelle individuell konfiguriert werden. Speicherformat, Werte: • XML: Das Objekt wird im XML-Format gespeichert. ◦ Objekte mit diesem Speicherformat werden in der Dateiversion 1.1.0.1 gespeichert, siehe Eigenschaft **Version**. • Base64: Das Objekt wird im Base64-Format gespeichert. ◦ Objekte mit diesem Speicherformat werden in der Dateiversion 1.2.0.0 gespeichert, siehe Eigenschaft **Version**. Bitte beachten Sie, dass Objekte mit der Dateiversion 1.2.0.0 (oder größer) nicht mit Engineering-Versionen < TC3.1.4024 geladen werden können! **Vorteile von Base64 gegenüber XML:** Im Vergleich zu XML ergibt sich mit Base64 eine komprimierte Speicherung. Als Folge dessen kann bei Dateizugriffen auf diese Objekte eine verbesserte Performance erreicht werden, welche beispielsweise beim Projektladen oder beim Verschieben/Kopieren der Objekte zum Tragen kommt. **Verfügbarkeit von Base64:** Das Speicherformat Base64 steht ab Build 4024 für die folgenden SPS-Objekte optional zur Verfügung: • POUs, bei denen der POU-Rumpf in einer grafischen Implementierungssprache programmiert ist ◦ AS (Ablaufsprache) ◦ FUP/KOP/AWL (Funktionsplan/Kontaktplan/Anweisungsliste) ◦ CFC (Continuous Function Chart und Seitenorientierter CFC) ◦ UML Klassendiagramm und Zustandsdiagramm • POUs, die über ein Unterelement (z. B. Aktion, Methode) verfügen, das in einer grafischen Implementierungssprache programmiert ist (grafische Sprachen siehe erster Stichpunkt) • Visualisierungen • Visualisierungsmanager • Textlisten • Rezeptmanager • Bildersammlungen **Einstellungsmöglichkeit des Standard-Speicherformats:** Über die Einstellung „Objektinhalt schreiben als“ in den SPS-Projekteigenschaften (Kategorie Advanced [▶ 961]) können Sie für ein SPS-Projekt definieren, welches das standardmäßige Speicherformat für die oben genannten Objekttypen ist.
Separate LineIds	Wert: True oder False • True: Die Line-IDs dieser POU werden in einer separaten Datei (LineIDs.dbg) gespeichert. • False: Die Line-IDs dieser POU werden in der POU selbst gespeichert. (an dieser Stelle nicht editierbar, konfigurierbar in den Write Options [▶ 1039])
Sort	Wert: Name oder GUID Gibt die Art und Weise an, in welcher Reihenfolge die Unterobjekte (z.B. Methoden) in dem übergeordneten Objekt gespeichert werden: entweder sortiert nach dem Namen oder nach der GUID. (an dieser Stelle nicht editierbar, konfigurierbar in den Write Options [▶ 1039])

Write ProductVersion	Wert: True oder False (an dieser Stelle nicht editierbar, konfigurierbar über die Einstellung „Produktversion in Dateien schreiben“ in der Kategorie Advanced [▶ 961] der SPS-Projekteigenschaften)

AS-Einstellungen

In diesem Bereich werden für das gerade ausgewählte AS-Objekt die aktuellen Einstellungen bezüglich des Übersetzens (Kompilieren) und der Handhabung der impliziten Variablen angezeigt.

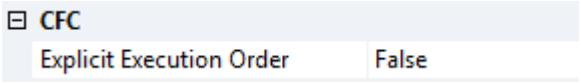


Voreingestellte SFC-Einstellungen nutzen	True (default): Mit dieser Option können die in den SPS-Projekteigenschaften definierten Standardwerte auf das gerade ausgewählte Objekt angewendet und in der Ansicht **Eigenschaften** des Objekts dargestellt werden. False: Mit dieser Option können speziell für dieses AS-Objekt gültige AS-Einstellungen konfiguriert werden.
Übersetzen („Build“) und Variablen („Flags“)	Die Bedeutung dieser Punkte entspricht den Standardeinstellungen für AS-Objekte, die in den SPS-Projekteigenschaften konfiguriert werden.

CFC-Einstellungen

Verfügbar ab TC3.1 Build 4026

In diesem Bereich wird der Ausführungsreihenfolge-Modus für das ausgewählte CFC-Objekt festgelegt. Im CFC-Editor positionieren Sie die Elemente und damit die Netzwerke frei. Um zu vermeiden, dass die Ausführungsreihenfolge im CFC-Programmierbausteins unbestimmt ist, stehen zwei Modi zur Verfügung.



Explicit Execution Order	False (default): Automatischer Datenfluss-Modus True: Expliziter Ausführungsreihenfolge-Modus

Automatischer Datenfluss-Modus

In diesem Modus wird die Ausführungsreihenfolge automatisch nach Datenfluss und bei Mehrdeutigkeit nach Netzwerktopologie festgelegt. Die Programmierbausteine und die Ausgänge werden intern nummeriert. Dabei werden die oberen vor den unteren und die linken vor den rechten Netzwerken ausgeführt.

Vorteil: Die automatische festgelegte Ausführungsreihenfolge ist zeit- und zyklusoptimiert. Sie benötigen auch während des Entwicklungsprozesses keine Information über die intern verwaltete Ausführungsreihenfolge.

Die Elemente im CFC-Editor werden ohne Marken und ohne Nummerierung angezeigt. Ein manuelles Ändern der Ausführungsreihenfolge ist nicht möglich. Bei Netzwerken mit Rückkopplungen können Sie zusätzlich einen Startpunkt setzen.

Im Menü **CFC > Ausführungsreihenfolge** stehen in diesem Modus folgende Befehle zur Verfügung:

- Befehl Ausführungsreihenfolge anzeigen [▶ 1061]
- Befehl Startpunkt der Rückkopplung setzen [▶ 1061]

Expliziter Ausführungsreihenfolge-Modus

In diesem Modus können Sie die Ausführungsreihenfolge explizit festlegen. Dafür werden die Elemente im CFC-Editor mit Marken und Nummerierung angezeigt und Menübefehle bereitgestellt, mit denen Sie die Reihenfolge bestimmen können.

Im Menü **CFC > Ausführungsreihenfolge** stehen folgende Befehle zur Verfügung:

- Befehl An den Anfang [▶ 1061]
- Befehl Ans Ende [▶ 1062]
- Befehl Eins vor [▶ 1062]
- Befehl Eins zurück [▶ 1063]
- Befehl Ausführungsreihenfolge setzen [▶ 1063]
- Befehl Nach Datenfluss anordnen [▶ 1063]
- Befehl Topologisch anordnen [▶ 1064]

Bis Build 4026 war dieses Verhalten das übliche Verhalten von CFC-Programmierbausteinen. Beachten Sie, dass Sie die Ausführungsreihenfolge in Eigenverantwortung anpassen und die Konsequenzen und Effekte selbst beurteilen müssen. Die Ausführungsreihenfolge wird dafür ständig eingeblendet.

17.4.4 Befehl Eigenschaften (SPS-Projekt)

Symbol: 

Funktion: Dieser Befehl öffnet ein Editorfenster, in dem die Eigenschaften des Projekts und zusätzliche projektbezogene Informationen angezeigt und definiert werden können.

Aufruf: Kontextmenü des SPS-Projektobjekts (<SPS-Projektname> Project) oder Menü **Projekt**

Voraussetzung: Ein Projekt ist geöffnet.

TwinCAT speichert die SPS-Projekteigenschaften direkt im SPS-Projekt.

Geltungsbereich von SPS-Projekteigenschaften

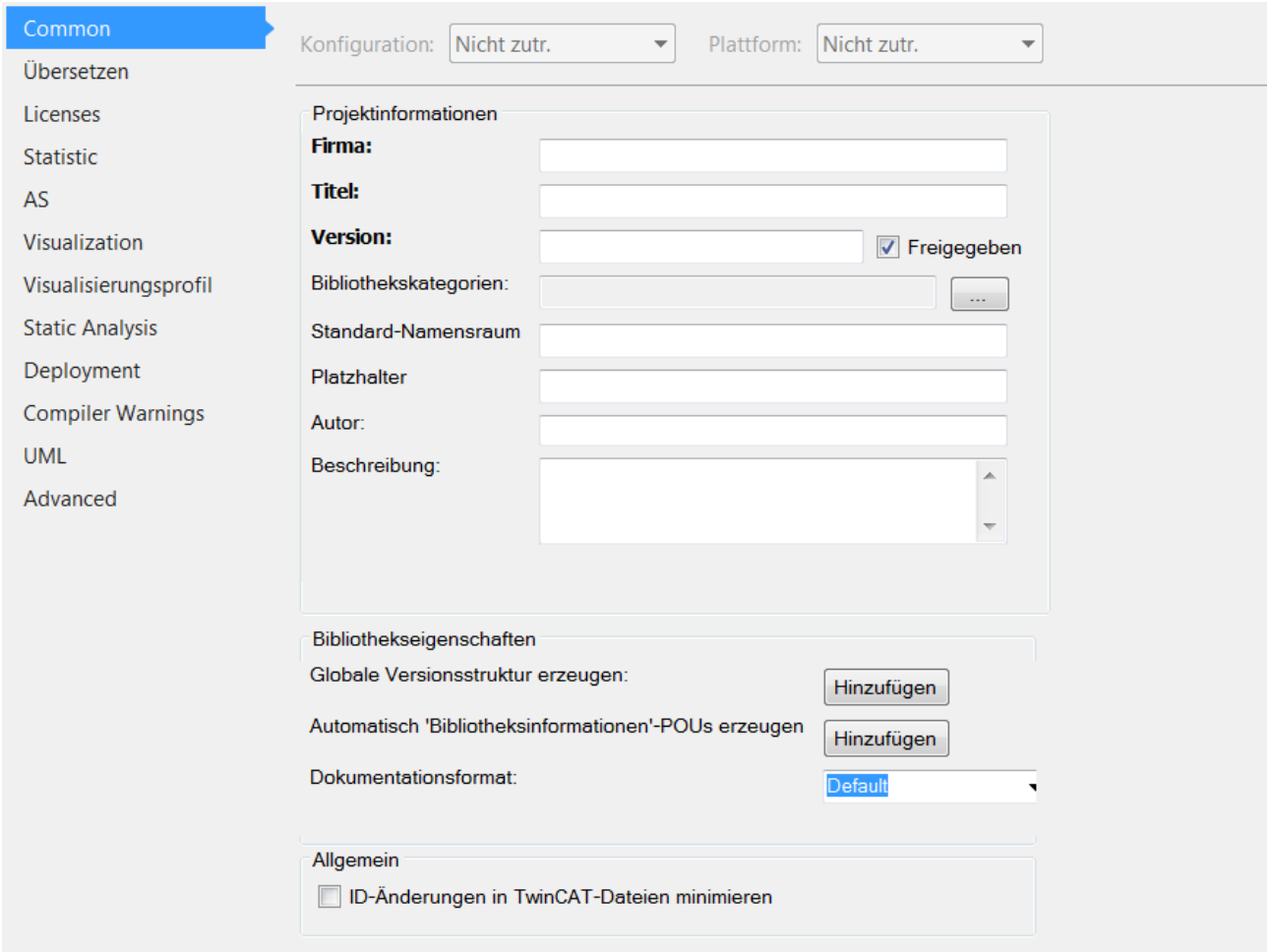
Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Siehe auch:

- Dokumentation PLC: SPS-Projekt konfigurieren [▶ 62]

17.4.4.1 Kategorie Common

Die Kategorie **Common** enthält allgemeine Informationen und Metainformationen der Projektdatei. TwinCAT erstellt aus diesen Informationen Schlüssel in der Registerkarte **Eigenschaften**. Wenn zum Beispiel im Textfeld **Firma** der Name „Company_A“ eingetragen ist, ist in der Registerkarte **Eigenschaften** der Schlüssel **Company** mit dem Wert „Company_A“ vorhanden.



Projektinformation

Bei einem Bibliotheksprojekt muss eine Firma, ein Titel und eine Version eingetragen sein, um die Bibliothek installieren zu können.	
Firma	Name der Firma, in der dieses Projekt (Applikation oder Bibliothek) erstellt wurde. Er dient neben der Bibliothekskategorie zur Sortierung im Bibliotheks-Repository
Titel	Titel des Projekts
Version	Version des Projekts, zum Beispiel „0.0.0.1“
Freigegeben	☑ : Schutz gegen Änderung aktiviert. Folge: Wenn Sie nun das Projekt editieren, erscheint eine Eingabeaufforderung, ob das Projekt wirklich geändert werden soll. Wenn Sie diese Abfrage einmalig mit Ja beantworten, erscheinen bei weiteren Editieraktionen keine Abfrage mehr.
Bibliothekskategorien	Kategorien des Bibliotheksprojekts, nach denen Sie im Dialog Bibliotheks-Repository sortieren können. Wenn keine Kategorie angegeben ist, wird der Bibliothek die Kategorie „Sonstige“ zugewiesen. Soll sie einer anderen Kategorie angehören, muss eine solche definiert sein. Die Definition von Bibliothekskategorien erfolgt in einer oder mehreren externen Beschreibungsdateien im XML-Format. Für die Zuordnung der Bibliothek kann entweder eine solche Datei aufgerufen werden oder aber eine andere Bibliotheksdatei, die bereits selbst die Informationen über die Kategorien aus einer Beschreibungsdatei aufgenommen hat. Voraussetzung: Das Projekt ist ein Bibliotheksprojekt.
	Der Dialog **Bibliothekskategorien** öffnet sich, in dem Sie Bibliothekskategorien hinzufügen können.
Standard-Namensraum	Die Standardeinstellung für den Namensraum einer Bibliothek entspricht dem Bibliothekstitel. Für eine Bibliothek kann auch explizit ein davon abweichender Namensraum definiert werden: entweder allgemein für die Bibliothek bei der Bibliothekserstellung an dieser Stelle in den Projektinformationen oder für den lokalen Gebrauch der Bibliothek in einem Projekt im **Eigenschaften**-Dialog der Bibliotheksreferenz. Der Namensraum der Bibliothek muss als Präfix des Bezeichners verwendet werden, damit ein eindeutiger Zugriff auf ein Modul möglich ist, das mehrfach im Projekt vorhanden ist, oder wenn der Gebrauch dieses Präfixes durch die Bibliothekseigenschaft LanguageModelAttribute „qualified-access-only“ („Eindeutiger Zugriff auf Bibliotheksmodule oder -variablen“) erzwungen wird. Wenn Sie hier keinen Standardnamensraum definieren, gilt automatisch der Name der Bibliotheksdatei als Namensraum.
Platzhalter	An dieser Stelle kann ein Standardname des Platzhalters festgelegt werden, der diese Bibliothek repräsentiert bzw. referenziert. Falls ein dieser Stelle nicht explizit ein Platzhalter festgelegt wird, entspricht die Standardeinstellung für den Platzhalternamen einer Bibliothek dem Bibliothekstitel.
Autor	Autor des Projekts
Beschreibung	Kurze Beschreibung des Projekts (z. B. Inhalt, Funktionalitäten, allgemeine Hinweise wie nur für den internen Gebrauch etc.)

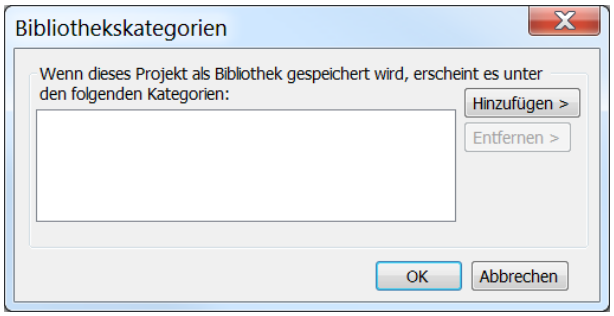
Bibliothekseigenschaften

Globale Versionsstruktur erzeugen	Erzeugt im SPS-Projekt eine globale Variablenliste, welche die Versionsinformationen enthält.
Automatisch Bibliotheksinformationen-POU erzeugen	Schaltfläche **Hinzufügen**: POU-Objekte vom Typ „Funktion“ werden automatisch im Projektbaum angelegt, die verwendet werden können, um im Applikationsprogramm auf die Projekteigenschaften zuzugreifen. In diesem Fall werden spezielle Funktionen für die Eigenschaften **Firma**, **Title** und **Version** erzeugt (F_GetCompany, F_GetTitle, F_GetVersion). Falls diese Funktionen über **Hinzufügen** zum Projekt hinzugefügt wurden, können sie über **Entfernen** aus dem Projekt entfernt werden**.**
Dokumentationsformat	Optionen: • Bis TC3.1 Build 4024: **reStructuredText**: • Ab TC3.1 Build 4026: **TcDocGen** Bei der Bibliothekserstellung werden Kommentare, die einem bestimmten Format entsprechen, neustrukturiert und in dieser angepassten Darstellung in der Registerkarte **Dokumentation** im Bibliotheksverwalter angezeigt. Dadurch ergeben sich zusätzliche Möglichkeiten der Bibliotheksdokumentation.
Implizite Prüfungen für Compiled Libraries erlauben	Verfügbar ab TC3.1 Build 4026 ☑: TwinCAT führt implizite Prüfungen auch für Bausteine aus geschützten Bibliotheken (*.compiled-libraries) aus. Voraussetzung: Die Compiler-Definition „checks_in_libs“ ist in den **SPS-Projekteigenschaften** im Feld **Compilerdefinitionen** eingetragen (Kategorie **Übersetzen**). Siehe auch: Bausteine für implizite Prüfung verwenden [▶ 172]
Qualified_only für Bibliothekszugriff erzwingen	Verfügbar ab TC3.1 Build 4026 ☑: Objekte aus dieser Bibliothek dürfen nur mit der Angabe des Namensraums der Bibliothek verwendet werden. Siehe auch: Attribut 'qualified_only' [▶ 859]
Referenzierung als Bibliothek erlauben	Verfügbar ab TC3.1 Build 4026 ☑: Sie können das SPS-Projekt in einem anderen SPS-Projekt als Bibliothek referenzieren. Siehe auch: SPS-Projekt als referenzierte Bibliothek verwenden

Allgemein

ID-Änderungen in TwinCAT-Dateien minimieren	☑: Die GUIDs der SPS-Objekte (z. B. POUs) werden mit denen des SPS-Projekts verknüpft (mittels XOR). Damit werden Änderungen der GUIDs der SPS-Objekte vermieden, wenn diese mehrmals in verschiedenen Projekten verwendet werden.

Dialog Bibliothekskategorien



Liste von Kategorien	Liste der Kategorien, die dem Bibliotheksprojekt zugewiesen sind. Sie können aus mehrere Quellen stammen. Wenn Sie alle gewünschten Kategorien eingetragen haben, bestätigen Sie den Dialog mit **OK**.
Hinzufügen	Die Befehle **Aus Beschreibungsdatei…** und **Aus anderer Bibliothek…** erscheinen.
Entfernen	TwinCAT entfernt die selektierte Kategorie.
Aus Beschreibungsdatei…	Dialog **Beschreibungsdatei auswählen** erscheint, in dem Sie eine Beschreibungsdatei mit Erweiterung *.libcat.xml auswählen. Die Datei enthält Befehlskategorien. Wenn Sie den Dialog mit **Öffnen** beenden, übernimmt TwinCAT diese Kategorien.
Aus anderer Bibliothek…	Dialog **Bibliothek auswählen** erscheint, in dem Sie eine Bibliothek (*.library) auswählen, deren Befehlskategorien übernommen werden sollen. Wenn Sie den Dialog mit **Öffnen** beenden, übernimmt TwinCAT die Kategorien.
OK	TwinCAT stellt die Kategorien als Projektinformation zur Verfügung und zeigt sie im Feld **Bibliothekskategorien** an.
Abbrechen	Schließt den Dialog. Der Vorgang wird abgebrochen.

Siehe auch:

- Dokumentation PLC: SPS-Projekt konfigurieren [▶ 62]
- Dokumentation PLC: Bibliotheken verwenden [▶ 278]

17.4.4.2 Kategorie Übersetzen

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Die Kategorie **Übersetzen** dient der Konfiguration der Compileroptionen.

Einstellungen

Compilerdefinitionen	Hier können Compilerdefinitionen/„defines“ (siehe {define}-Anweisungen) und Bedingungen für die Kompilation der Applikation eingetragen werden (bedingte Kompilierung). Eine Beschreibung der verfügbaren bedingten Pragmas finden Sie im Abschnitt Bedingte Pragmas [▶ 875]. Der Ausdruck expr, der in solchen Pragmas verwendet wird, kann auch hier eingegeben werden. Mehrere Einträge sind in Form einer komma-separierten Liste möglich.
System-Compilerdefinitionen	Verfügbar ab TwinCAT 3.1 Build 4024 Hier werden automatisch die Compilerdefinitionen übernommen, die auf System Manager-Ebene in den SPS-Projekteinstellungen unter Compilerdefinitionen [▶ 967] gesetzt worden sind.
Download Applikationsinfo	Verfügbar ab TwinCAT 3.1 Build 4024 Situation: Sie sind dabei, ein SPS-Projekt auf die Steuerung zu laden, das sich von dem bereits dort liegenden unterscheidet. In diesem Fall erscheint ein Meldungsfenster, das einen **Details**-Button enthält. Über diesen Button können Sie das **Applikationsinformation**-Fenster öffnen, das eine Prüfung der Unterschiede zwischen dem aktuellen SPS-Projekt und dem SPS-Projekt auf der Steuerung ermöglicht. Dabei geht es um den Vergleich der Anzahl von Bausteinen, der Daten und der Speicherorte. Das **Applikationsinformation**-Fenster enthält eine grobe Beschreibung der Unterschiede, beispielsweise: • Deklaration von MAIN geändert • Variable fbMyNewInstance in MAIN eingefügt • Anzahl der Methoden/Aktionen von FB_Sample geändert ☑ (Voreinstellung): Wenn diese Einstellung aktiviert ist, wird die Information zum Inhalt des SPS-Projekts mit auf die SPS geladen. Dies ermöglicht eine erweiterte Prüfung der Unterschiede zwischen dem aktuellen SPS-Projekt und dem SPS-Projekt auf der Steuerung. Die erweiterte Prüfungsmöglichkeit besteht darin, dass das **Applikationsinformation**-Fenster den zusätzlichen Reiter **Onlinevergleich** enthält, welcher eine Baumvergleichsansicht zeigt. Anhand dieser können Sie erkennen, welche POUs geändert, gelöscht oder hinzugefügt wurden. Der zusätzliche Reiter erscheint, wenn Sie den blau unterstrichenen Befehl im unteren Bereich des **Applikationsinformation**-Fensters ausführen („Applikation nicht aktuell. Code jetzt generieren, um den Onlinevergleich anzuzeigen?“).

Generiere tpy-Datei	Verfügbar ab TwinCAT 3.1 Build 4024 Die tpy-Datei enthält u. a. Projekt-, Routing-, Compiler- und Zielsystem-Informationen und ist das Format zur Beschreibung eines TwinCAT 2 SPS-Projekts. Aus Kompatibilitätsgründen mit bestehenden Anwendungen kann diese Datei bei Bedarf für ein TwinCAT 3 SPS-Projekt erstellt werden. ☐ (Voreinstellung): Beim Erstellen des SPS-Projekts wird die zum Projekt gehörende tpy-Datei nicht erstellt. ☑: Beim Erstellen des SPS-Projekts wird die zum Projekt gehörende tpy-Datei erstellt und im Projektordner abgelegt. Beachten Sie, dass der Wert und die Konfigurationsverfügbarkeit dieser Option davon abhängt, ob die TPY-Datei als Target-Datei konfiguriert ist oder nicht (siehe Registerkarte Settings [▶ 969]). • Wenn die TPY-Datei als Target-Datei aktiviert wird, passiert folgendes: ◦ TwinCAT merkt sich den aktuellen Status der Option „Generiere tpy-Datei“ (= „ursprünglicher Wert“, s.u.). ◦ Falls dies nicht bereits der Fall ist, wird die Option „Generiere tpy-Datei“ beim nächsten Erstellen des Projekts automatisch aktiviert. ◦ Außerdem wird die Option „Generiere tpy-Datei“ ausgegraut, sodass sie vom Anwender nicht deaktiviert werden kann, solange die TPY-Datei als Target-Datei konfiguriert ist. • Wenn die TPY-Datei anschließend als Target-Datei wieder deaktiviert wird, passiert folgendes: ◦ Beim nächsten Erstellen des Projekts erhält die Option „Generiere tpy-Datei“ ihren „ursprünglichen Wert“ (s.o.). ◦ Außerdem ist die Option nicht mehr ausgegraut, sodass sie wieder für Konfigurationen durch den Anwender verfügbar ist.
Kommentare über POU und DUT Deklarationen zur TMC Datei hinzufügen	Verfügbar ab TwinCAT 3.1 Build 4026 ☑ (Voreinstellung): Kommentare zu POUs und DUTs, die oberhalb von deren Deklaration stehen, werden mit dem Datentyp in der TMC Datei gespeichert. ☐: Kommentare oberhalb der Deklaration von POUs und DUTs werden nicht mit in der TMC Datei gespeichert. Dadurch wird die Größe der TMC Datei verringert, wenn die Kommentare ausführliche Beschreibungen zu den POUs und DUTs enthalten.

Solution Options

Compilerversion	Definiert die Compilerversion, die TwinCAT beim Übersetzen und während des Ladens zum Übersetzen verwendet. Beachten Sie, dass diese Einstellung nicht den Remote Manager ersetzt. Für die Handhabung verschiedener Engineering-Versionen sollte, wenn es sich bei dem SPS-Projekt um ein Applikationsprojekt handelt, immer der Remote Manager verwendet werden. Die Compilerversion sollte in diesem Fall immer auf „neueste“ stehen. Die Einstellung der Compilerversion kommt nur dann zum Tragen, wenn es sich bei dem SPS-Projekt, welches versionsverwaltet werden soll, um ein Bibliotheksprojekt handelt. Es wird empfohlen, die Bibliothek mit der ältesten Version zu speichern, mit der sie letztlich verwendet werden soll. Dazu muss die Compilerversion auf die entsprechende fixe Version gesetzt werden (z. B. „3.1.4018.0“).
Maximale Anzahl an Warnungen	Bezieht sich auf die Warnungen, die TwinCAT in der Ansicht **Fehlerliste** maximal ausgibt. Die Auswahl der angezeigten Compilerwarnungen legen Sie im Dialog **Projekteinstellungen** in der Kategorie **Compilerwarnungen** fest.
Konstanten ersetzen	☑ : Für jede Konstante skalaren Typs, also nicht für STRING, ARRAY oder Strukturen, lädt TwinCAT direkt den Wert. Im Onlinebetrieb kennzeichnet TwinCAT die Konstanten im Deklarationseditor oder Monitoring-Fenster durch ein dem Wert vorangestelltes Symbol. In diesem Fall ist ein Zugriff, zum Beispiel über einen ADR-Operator, Forcen und Schreiben, nicht möglich. ☐ (Voreinstellung): Der Zugriff auf Konstanten ist möglich. Die Rechenzeit verlängert sich minimal.

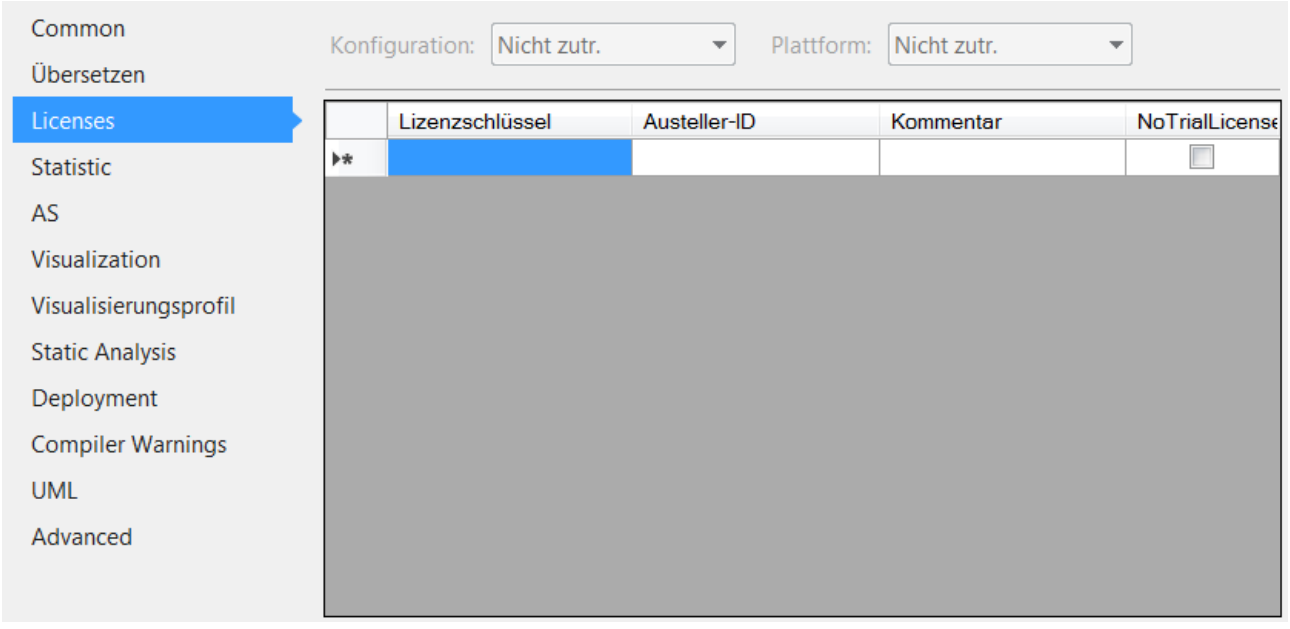
Siehe auch:

- Kategorie Compiler Warnings [▶ 960]

17.4.4.3 Kategorie Lizenzen

Über die Kategorie **Lizenzen** soll es zukünftig ermöglicht werden, einer eigenen Bibliothek eine TwinCAT 3 OEM-Lizenz zuzuordnen. Dieses Feature ist aber noch nicht implementiert.

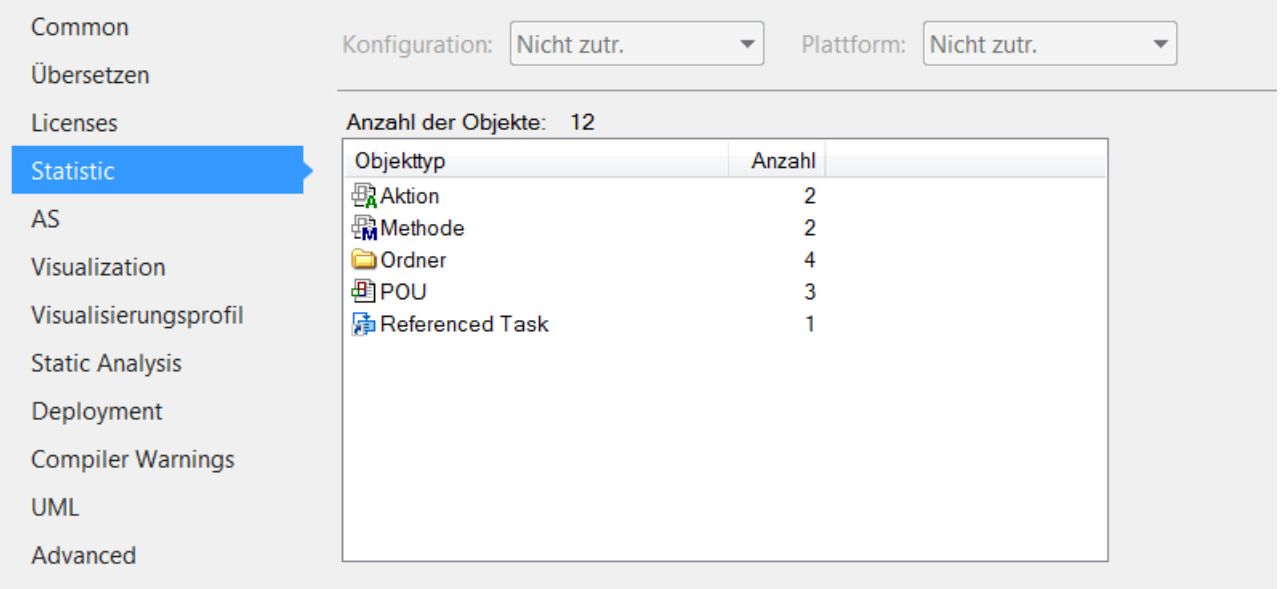
Diese Kategorie wird daher in der aktuellen TwinCAT-Version noch nicht unterstützt (Reservierung für zukünftige Nutzung!).



Die Abfrage einer OEM-Lizenz für eine eigene Bibliothek muss aktuell im Code der Bibliothek durch den Anwender erfolgen. Siehe Abfrage einer OEM-Lizenz in der PLC Applikation.

17.4.4.4 Kategorie Statistic

Die Kategorie **Satistic** gibt eine statistische Auskunft, wie viele Objekte der einzelnen Typen im Projekt verwendet sind.



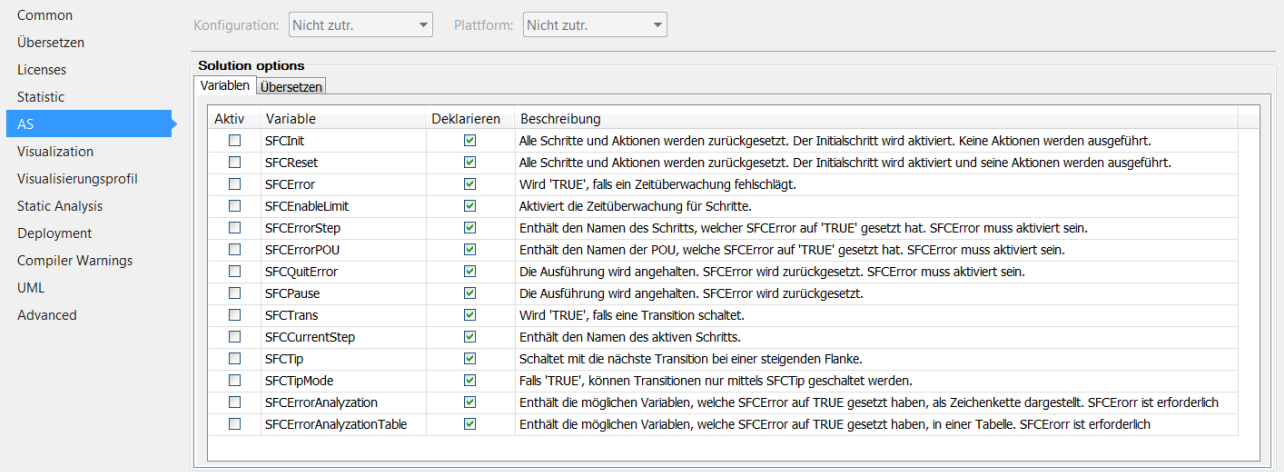
17.4.4.5 Kategorie AS

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Die Kategorie **AS** dient der Konfiguration der Einstellungen für AS-Objekte. Jedes neue AS-Objekt hat die konfigurierten Einstellungen automatisch in seinen Eigenschaften.

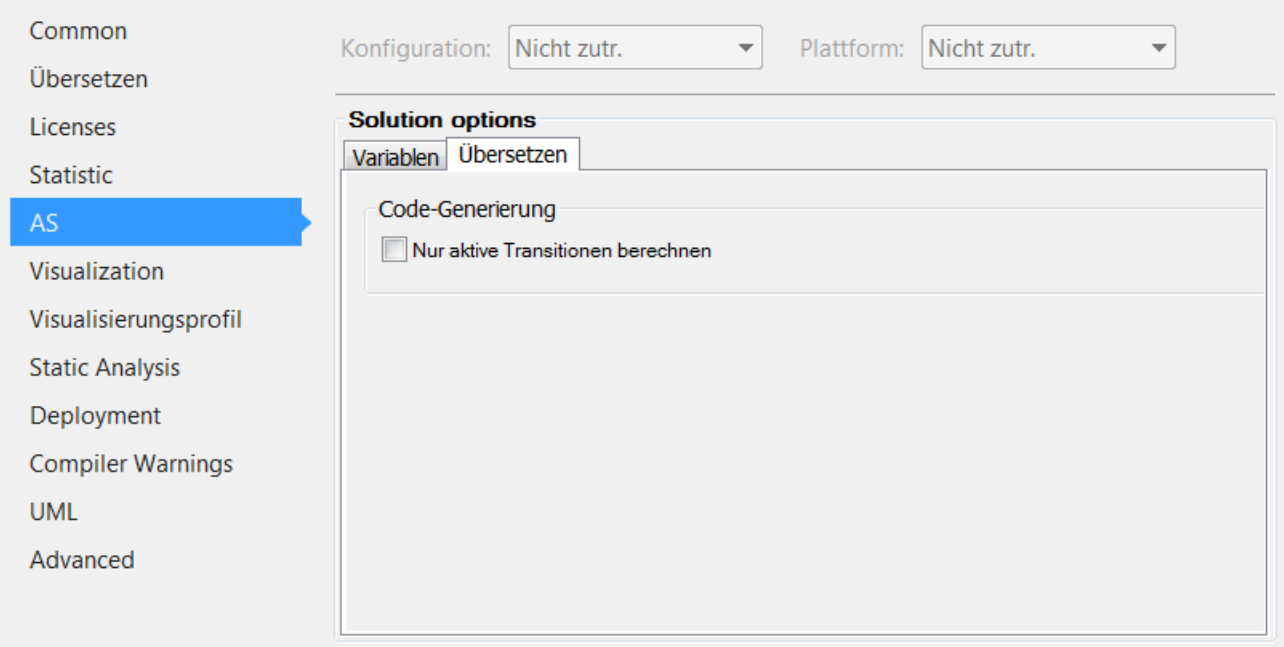
Registerkarte Variablen



Implizit erzeugte Variablen (Flags) zur Kontrolle und zum Monitoring der Abarbeitung in einem AS-Diagramm.	
Aktiv	☑ : Die entsprechende Variable wird verwendet.
Deklarieren	☑ : Die entsprechende Variable wird automatisch angelegt. Ansonsten, falls die Verwendung beabsichtigt ist (Aktiv ist gesetzt), muss der Anwender sie selbst deklarieren.

Eine automatisch deklarierte Flag-Variable ist nur im Onlinebetrieb im Deklarationsteil des AS-Editors sichtbar.

Registerkarte Übersetzen



Codegenerierung

Nur aktive Transitionen berechnen	☑ : TwinCAT erzeugt nur für gerade aktive Transitionen Code.

Siehe auch:

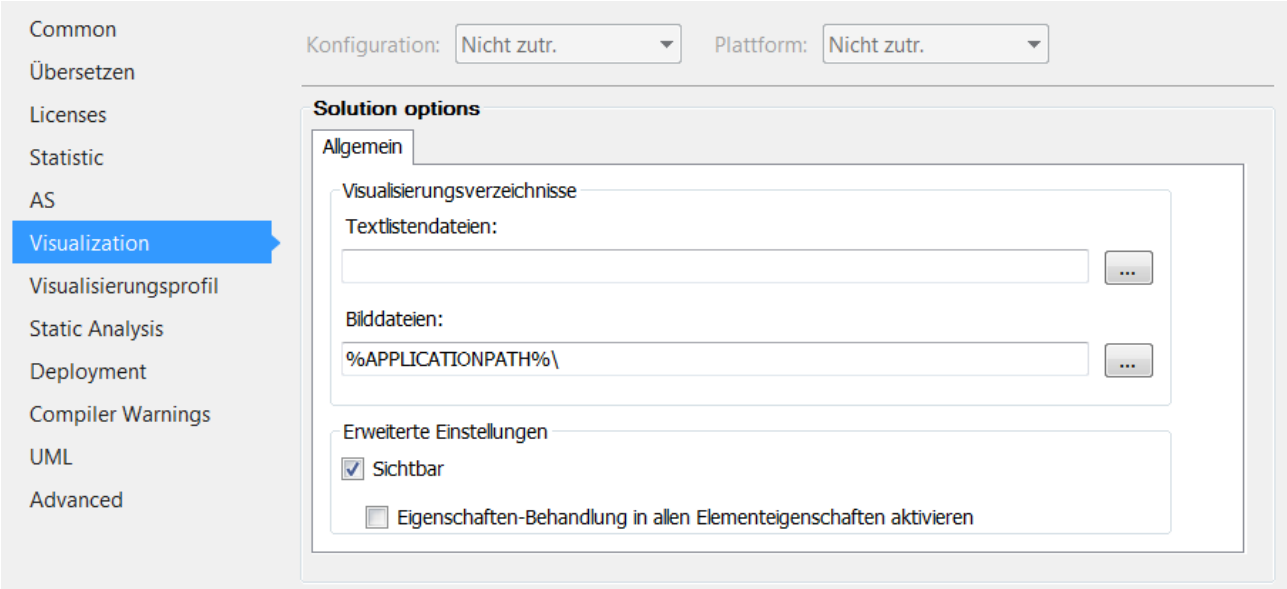
- AS-Flags [▶ 672]

17.4.4.6 Kategorie Visualization

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Die Kategorie **Visualization** dient der Konfiguration der projektweiten Einstellungen für Objekte des Typs Visualisierung.



Registerkarte Allgemein

Visualisierungsverzeichnisse

Textlistendateien	Verzeichnis, das Textlisten enthält, die im Projekt zur Verfügung stehen, um Texte für verschiedene Sprachen zu konfigurieren. TwinCAT verwendet das Verzeichnis zum Beispiel beim Exportieren oder Importieren von Textlisten. Mit Klick auf ... erscheint der Dialog **Ordner suchen**, der das Auswählen eines Verzeichnisses im Dateisystem ermöglicht.
Bilddateien	Verzeichnis, das Bilddateien enthält, die im Projekt zur Verfügung stehen. Mehrere Ordner sind mit einem Strichpunkt voneinander getrennt. TwinCAT verwendet das Verzeichnis zum Beispiel beim Exportieren oder Importieren von Bilddateien. Mit Klick auf ... erscheint der Dialog **Ordner suchen**, der das Auswählen eines Verzeichnisses im Dateisystem ermöglicht.

Erweiterte Einstellungen

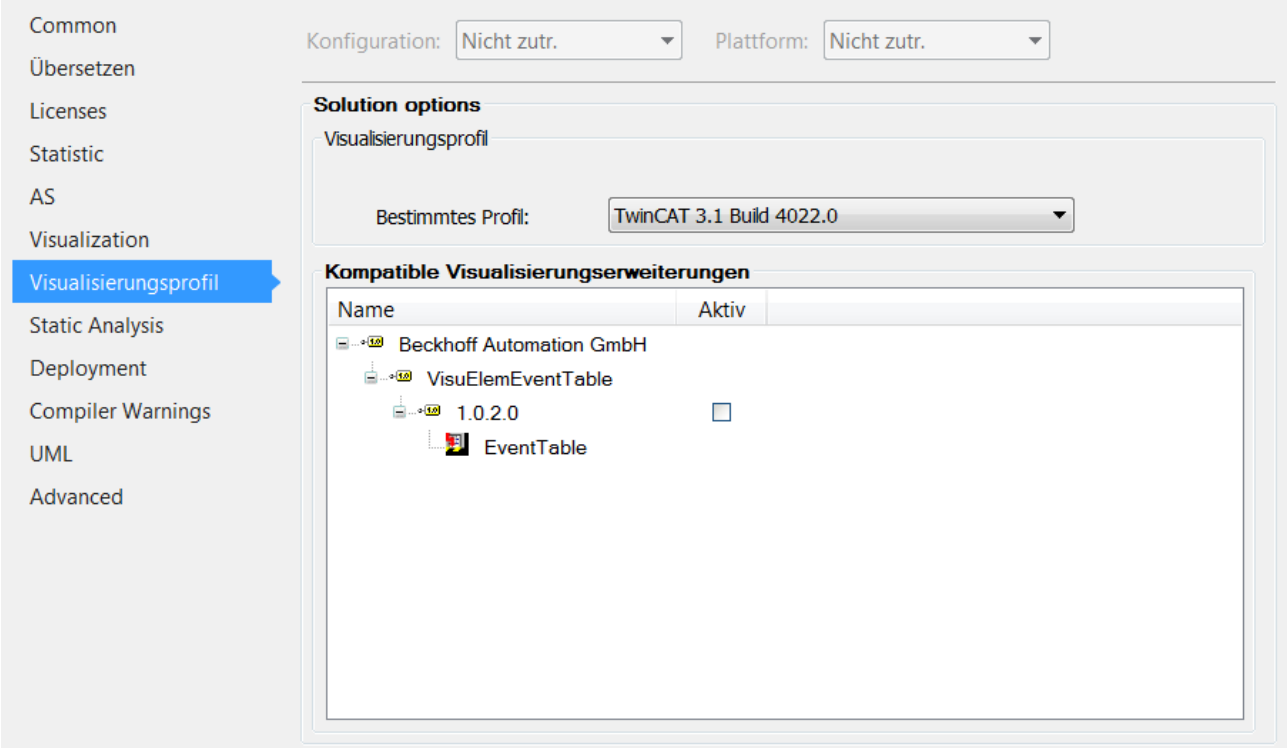
Eigenschaften-Behandlung in allen Elementeigenschaften aktivieren	: Sie können ein Visualisierungselement in denjenigen seiner Eigenschaften, in denen Sie eine IEC-Variable auswählen, auch mit einer Eigenschaft konfigurieren. TwinCAT erzeugt dann beim Übersetzen einer Visualisierung zusätzlichen Code für die Eigenschaften-Behandlung. Voraussetzung: Ihr IEC-Code enthält mindestens ein Objekt des Typs Schnittstelleneigenschaft, also eine Eigenschaft . MAIN (PRG) Property_A Get Set Voraussetzung: **Sichtbar** ist aktiviert.

17.4.4.7 Kategorie Visualisierungsprofil

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Die Kategorie **Visualisierungsprofil** ermöglicht die Einstellung des Visualisierungsprofils.



Visualisierungsprofil

Bestimmtes Profil	Profil, das TwinCAT im Projekt verwendet und das die Visualisierungselemente bestimmt, die im Projekt zur Verfügung stehen. Die Auswahlliste enthält alle bisher installierten Profile.

17.4.4.8 Kategorie Static Analysis

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Die Kategorie **Static Analysis** definiert die Prüfungen, die bei der statischen Codeanalyse berücksichtigt werden.

Static Analysis Light:

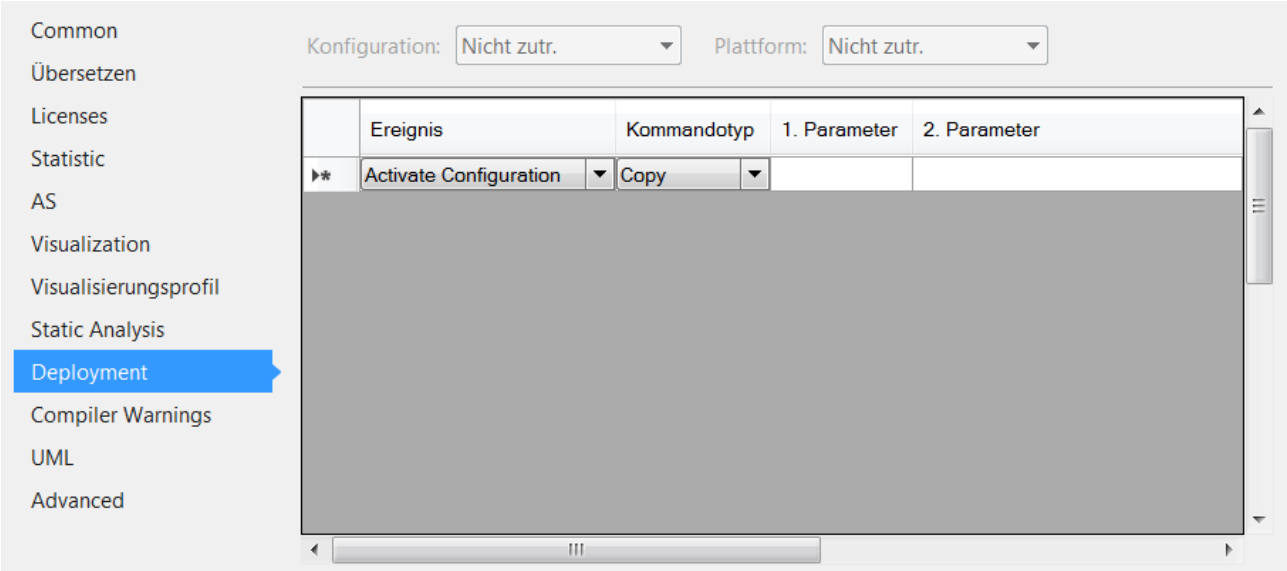
- Wenn Sie die zusätzliche Engineering-Lizenz TE1200 nicht aktiviert haben, können Sie die lizenzfreie Variante des Static Analysis (Static Analysis Light) nutzen, welche einige wenige Kodierregeln beinhaltet. Anhand der kostenfreien Light-Variante können Sie sich – auf Basis eines stark reduzierten Funktionsumfangs – beispielsweise mit dem prinzipiellen Handling des Produkts vertraut machen.
- Weitere Informationen zu Static Analysis Light finden Sie unter:
  Dokumentation PLC: SPS-Projekt programmieren > Syntax prüfen und Code analysieren >
  Codeanalyse (Static Analysis) [▶ 155]

Static Analysis Full:

- Wenn Sie die zusätzliche Engineering-Lizenz TE1200 aktiviert haben, steht Ihnen der volle Funktionsumfang des Static Analysis zur Verfügung (Speichern und Laden von Einstellungen, mehr als 100 Kodierregeln, Namenskonventionen, Metriken, unzulässige Symbole).
- Weitere Informationen zu Static Analysis Full finden Sie unter:
  TE1200 Static Analysis.

17.4.4.9 Kategorie Deployment

Die Kategorie **Deployment** dient der Einstellung von Befehlen, welche während des Aufspielen und Startens einer Anwendung ausgeführt werden sollen.



Folgende Ereignisse stehen zur Verfügung, nach welchen die in der Liste aufgeführten Kommandos aufgerufen werden können:

Activate Configuration	Das gewünschte Kommando wird nach einem Aktivieren der Konfiguration aufgerufen.
Plc Download	Das gewünschte Kommando wird nach einem erfolgten Download der SPS-Applikation auf das Zielsystem aufgerufen.
Plc Online Change	Das gewünschte Kommando wird nach einem erfolgten Online Change aufgerufen.
Plc After Compile	Das gewünschte Kommando wird nach einem Kompiliervorgang der SPS-Applikation aufgerufen.

Folgende Kommandos können ausgeführt werden:

Copy	Kopiert Dateien von Parameter 1 (Quellpfad) nach einem in Parameter 2 (Zielpfad) angegebenen Ort.
Execute	Führt die unter Parameter 1 aufgeführte Applikation bzw. das aufgeführte Skript aus.

Quell- und Zielpfade können virtuelle Umgebungsvariablen beinhalten, welche von TwinCAT entsprechend aufgelöst werden.

Die nachfolgenden Umgebungsvariablen werden u. a. unterstützt:

Virtuelle Umgebungsvariable	Registrierungswert	Defaultwert
%TC_INSTALLPATH%	InstallDir	C:\TwinCAT\3.**x** \
%TC_TARGETPATH%	TargetDir	C:\TwinCAT\3.**x** \Target\
%TC_BOOTPRJPATH%	BootDir	C:\TwinCAT\3.**x** \Boot\
%TC_RESOURCEPATH%	ResourceDir	C:\TwinCAT\3.**x** \Target\Resource\
%SOLUTIONPATH%	-	Speicherort der Solution-Datei

Die Registrierungswerte werden unter folgendem Schlüssel in der Regisitry abgelegt: *\HKLM\Software\Beckhoff\TwinCAT3*.

Beispiel:

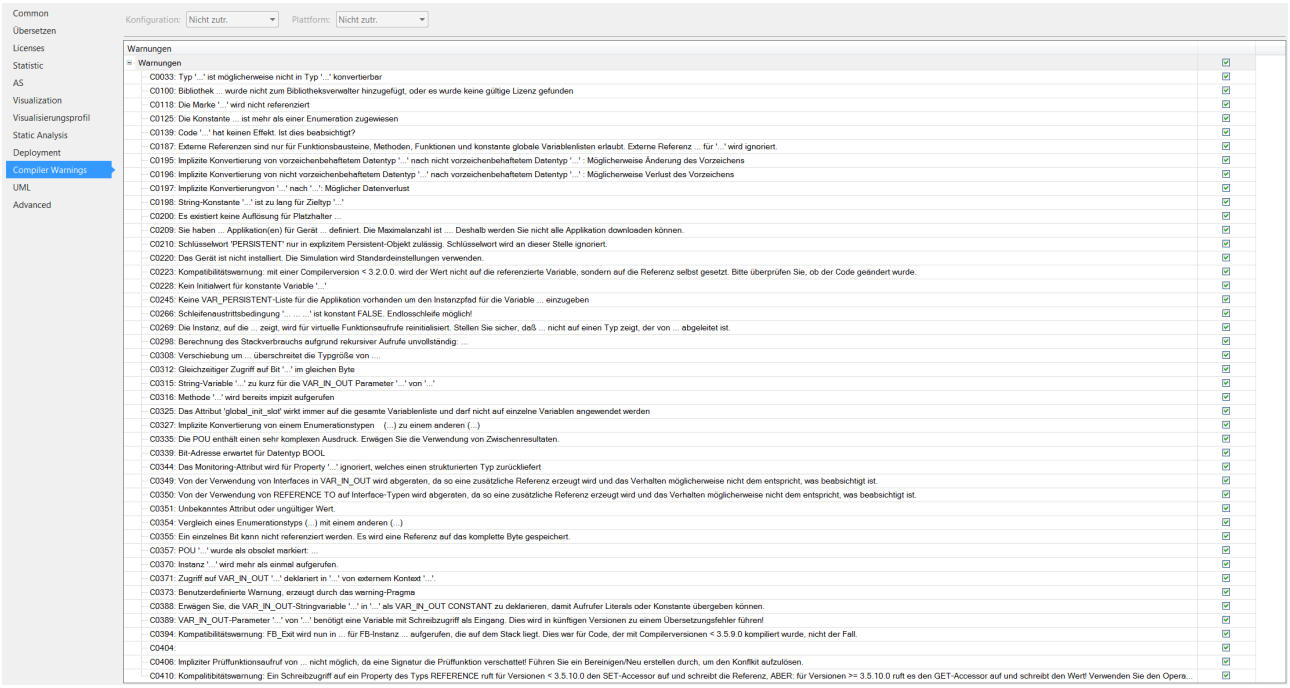
Im nachfolgenden Beispiel wird die Datei *SampleFile.xml* aus dem Projektunterordner Config der Solution in den Ordner *C:\plc\config* auf dem Zielsystem kopiert.

Ereignis	Kommandotyp	1. Parameter	2. Parameter
Activate Configuration	Copy	%SOLUTIONPATH%Config\SampleFile.xml	C:\plc\Config\SampleFile.xml

17.4.4.10 Kategorie Compiler Warnings

Die Kategorie **Compiler Warnings** dient der Auswahl der Compilerwarnungen, die TwinCAT bei einem Kompilierungslauf im Meldungsfenster anzeigt.

Die maximale Anzahl aufgelisteter Warnungen legen Sie in der Kategorie **Compile** fest.



Siehe auch:

- Befehl SPS-Projekt erstellen [▶ 973]
- Kategorie Übersetzen [▶ 951]

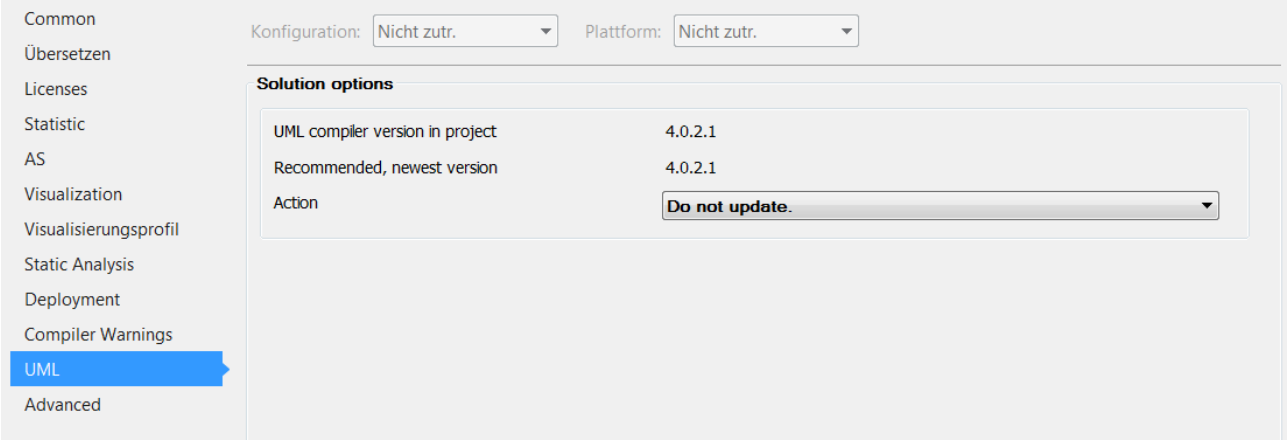
17.4.4.11 Kategorie UML

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

In der Kategorie **UML** können Sie die UML-Compiler-Version ändern. Diese Einstellung ist nur bei Verwendung des UML-Zustandsdiagramms relevant.

Weitere Informationen zu den Konfigurationsmöglichkeiten finden Sie im Abschnitt „UML Compiler-Version“ der Dokumentation TF1910 TC3 UML.



17.4.4.12 Kategorie Advanced

Geltungsbereich von SPS-Projekteigenschaften

Beachten Sie, dass sich der Geltungsbereich zwischen verschiedenen Projekteigenschaften unterscheidet!
Einige Eigenschaften betreffen nur das SPS-Projekt, dessen Eigenschaften Sie aktuell konfigurieren.
Andere Eigenschaften wirken sich hingegen auf alle SPS-Projekte aus, die sich in der Entwicklungsumgebung befinden. Solche Eigenschaften, die Sie zwar in den Projekteigenschaften eines SPS-Projekts ändern, die aber auch alle anderen SPS-Projekte betreffen, sind mit der Überschrift **Solution options** betitelt.

Die Kategorie **Advanced** dient der Konfiguration von erweiterten Eigenschaften.

Write options

Engineering-Inkompatibilität von Dateiversion 1.2.0.0 (oder größer) mit TwinCAT 3.1 < Build 4024

Beachten Sie, dass Objekte, die mit der Dateiversion 1.2.0.0 (oder größer) gespeichert werden, nicht mit Engineering-Versionen < TwinCAT 3.1.4024 geladen werden können!

Da ein Objekt bei Verwendung des optionalen Speicherformats „Base64“ automatisch mit der Dateiversion 1.2.0.0 gespeichert wird, können Objekte mit Base64-Speicherformat folglich nicht mit Engineering-Versionen < TwinCAT 3.1.4024 geladen werden.

Falls ein SPS-Projekt sowohl Objekte mit der Dateiversion 1.1.0.1 als auch Objekte mit der Dateiversion 1.2.0.0 enthält, werden die 1.1.0.1-Objekte mit einer Engineering-Version < TwinCAT 3.1.4024 weiterhin geladen. Lediglich die Objekte mit der Dateiversion 1.2.0.0 werden nicht geladen.

Die Dateiversion einer Datei, die mit der Dateiversion 1.2.0.0 gespeichert wurde, kann mit XAE-Versionen >= TwinCAT 3.1.4024 nachträglich wieder auf 1.1.0.1 gesetzt werden.

Objektinhalt schreiben als („Write object content as“)	Verfügbar ab TwinCAT 3.1 Build 4024 **Hintergrundinformationen:** Ab Build 4024 wird mit **Base64** ein neues Speicherformat eingeführt, das für die folgenden SPS-Objekte optional zur Verfügung steht: • POUs, bei denen der POU-Rumpf in einer grafischen Implementierungssprache programmiert ist ◦ AS (Ablaufsprache) ◦ FUP/KOP/AWL (Funktionsplan/Kontaktplan/Anweisungsliste) ◦ CFC (Continuous Function Chart und Seitenorientierter CFC) ◦ UML Klassendiagramm und Zustandsdiagramm • POUs, die über ein Unterelement (z. B. Aktion, Methode) verfügen, das in einer grafischen Implementierungssprache programmiert ist (grafische Sprachen siehe erster Stichpunkt) • Visualisierungen • Visualisierungsmanager • Textlisten • Rezeptmanager • Bildersammlungen Bislang wurden diese Objekte standardmäßig als XML gespeichert. Ob diese Objekttypen ab Build 4024 als XML oder als Base64 gespeichert werden sollen, ist konfigurierbar. **Vorteile von Base64 gegenüber XML:** Im Vergleich zu XML ergibt sich mit Base64 eine komprimierte Speicherung. Als Folge dessen kann bei Dateizugriffen auf diese Objekte eine verbesserte Performance erreicht werden, welche beispielsweise beim Projektladen oder beim Verschieben/Kopieren der Objekte zum Tragen kommt. **Einstellungsmöglichkeit des Standard-Speicherformats:** Über die Einstellung „Objektinhalt schreiben als“ in den SPS-Projekteigenschaften können Sie für ein SPS-Projekt definieren, welches das standardmäßige Speicherformat für die oben genannten Objekttypen ist. Das ausgewählte Standard-Speicherformat wird nur bei neu hinzugefügten Objekten verwendet (Ausnahme: nicht bei neu hinzugefügten POU-Unterobjekten. Beispiel: Eine POU ist als XML gespeichert und das Standard-Speicherformat ist als Base64 konfiguriert. Wenn zu der POU dann ein grafisches Unterobjekt hinzugefügt wird, bleibt das Speicherformat der POU und damit auch für das Unterobjekt als XML bestehen). Das Speicherformat eines existierenden Objekts mit einem vom Standard abweichenden Speicherformat wird nicht automatisch geändert, wenn das Objekt geändert und gespeichert wird. Das Speicherformat eines existierenden Objekts kann entweder individuell über das Eigenschaften-Fenster geändert werden (siehe unten) oder alternativ besteht beim Umstellen des Standard-Speicherformats die Möglichkeit, das neu ausgewählte Speicherformat auch für alle bereits bestehenden Objekte zu übernehmen. Wenn Sie an dieser Stelle das Speicherformat umstellen, erscheint ein entsprechendes Abfragefenster. Für die Einstellung „Objektinhalt schreiben als“ stehen die folgenden Auswahlmöglichkeiten zur Verfügung: • **XML** (Voreinstellung): Die oben genannten SPS-Objekte werden standardmäßig im XML-Format gespeichert. ◦ Objekte mit diesem Speicherformat werden in der Dateiversion 1.1.0.1 gespeichert. • **Base64**: Die oben genannten SPS-Objekte werden standardmäßig im Base64-Format gespeichert. ◦ Objekte mit diesem Speicherformat werden in der Dateiversion 1.2.0.0 gespeichert. Bitte beachten Sie, dass Objekte mit der Dateiversion 1.2.0.0 (oder größer) nicht mit Engineering-Versionen < TwinCAT 3.1.4024 geladen werden können! **Individuelle Einstellungsmöglichkeit des Speicherformats:**

Produktversion in Dateien schreiben („Write product version in files“)	Verfügbar ab TwinCAT 3.1 Build 4024 Die Produktversion repräsentiert, mit welcher Plugin-Version eine SPS-Datei (z. B. ein Funktionsbaustein) gespeichert wurde. Die Einstellung dieser Checkbox ist projektweit gültig und ist die Standard-Einstellung für alle geänderten oder neu hinzugefügten SPS-Objekte, die sich in diesem SPS-Projekt befinden. ☑ (Voreinstellung): Die Produkt- bzw. Plugin-Version wird mit in die Datei geschrieben (im XAE ist die Version nicht sichtbar, bei Analyse der Datei auf Dateiebene ist sie erkennbar). • Wenn Sie die Einstellung von deaktiviert nach aktiviert ändern, erscheint ein Abfragefenster, in dem Sie auswählen können, ob die Produktversion zu allen bestehenden Dateien hinzugefügt werden soll. • Anwendungsfall für die aktivierte Option: Diese Einstellung kann verwendet werden, um die Dateiversion beispielsweise zu Debug- oder Nachverfolgungszwecken mit in die Datei zu schreiben. • Bitte beachten Sie folgendes: Wenn die Datei mit einer anderen Produktversion gespeichert wird, führt dies zu einer Änderung dieser Datei, welche bei Verwendung von Sourcecode-Verwaltungssystemen als Dateiunterschied erkenntlich ist. ☐: Die Produkt- bzw. Plugin-Version wird nicht mit in die Datei geschrieben. • Wenn Sie die Einstellung von aktiviert nach deaktiviert ändern, erscheint ein Abfragefenster, in dem Sie auswählen können, ob die Produktversion aus allen bestehenden Dateien entfernt werden soll. • Anwendungsfall für die deaktivierte Option: Wenn die Produktversion nicht von Interesse ist, kann diese Einstellung verwendet werden. Dadurch werden Änderungen von Dateien hinsichtlich Sourcecode-Verwaltungssystemen minimiert.
Objektinhalt mit Profil schreiben: (“Write object content with Profile”)	Das Profil definiert das Format, in dem Objekte abgespeichert werden. Mit dem Build 4024 wurden beispielsweise neue Funktionalitäten für das PLC HMI hinzugefügt. Aus diesem Grund können Visualisierungsdateien, die mit dem Build 4024 abgespeichert worden sind, nicht direkt mit älteren Builds geöffnet werden. Wenn Sie hier ein 4022 Profil einstellen, dann werden die Visualisierungsdateien in dem passenden Format abgespeichert und können mit dem Build 4022 geöffnet werden. **Voraussetzung:** Damit zum Beispiel das 4022 Profil im Drop-Down-Menü verfügbar ist, muss entweder eine 4022 Remote Manager Installation durchgeführt worden sein oder die aktuelle 4024 XAE Installation über eine zuvor vorhandene 4022 XAE Installation installiert worden sein.
Write Bookmarks to File	Verfügbar ab TwinCAT 3.1 Build 4026 ☑: Die abgelegten Lesezeichen werden aus der Benutzeroptionsdatei vom Visual Studio Projekt (.suo) zusätzlich in eine separate Datei geschrieben. Diese Datei endet mit .bookmarks und liegt mit im Projektverzeichnis. Damit ist sie dann ebenfalls Bestandteil der bekannten Archivmöglichkeiten. ☐ (Voreinstellung): Die Lesezeichen werden nur in der .suo-Datei vom Visual Studio gespeichert. Eine bereits erstellte .bookmarks-Datei wird aus dem Projektverzeichnis gelöscht. Die globale Standardeinstellung für neue SPS-Projekte, ob Lesezeichen in einer separaten Datei abgelegt werden sollen, finden Sie unter Dialog Optionen - Write Options [▶ 1039]. Der Wert wird bei der Erstellung eines neuen SPS-Projekts einmalig in diese lokale Projekteinstellung übernommen.

Multiuser options

Multiuser nutzen („Use Multiuser“)	Verfügbar ab TwinCAT 3.1 Build 4024 ☐ (Voreinstellung): Die Multiuser-Funktionalität des SPS-Projekts ist nicht aktiviert. ☑ : Die Multiuser-Funktionalität des SPS-Projekts ist aktiviert. Bitte beachten Sie auch die weiterführenden Informationen hierzu in der Multiuser-Dokumentation.

Solution options

Sicherer Onlinebetrieb („Secure Online Mode“)	☐ : (Voreinstellung): Aus Sicherheitsgründen wird der Benutzer beim Aufruf der folgenden Befehle grundsätzlich dazu aufgefordert, die Ausführung nochmals zu bestätigen. • Konfiguration aktivieren • Restart TwinCAT System in Config/Run Mode • Reset Kalt • Reset Ursprung ☑ : Zusätzlich zu den oben genannten Befehlen, bei denen standardmäßig eine Bestätigungsaufforderung erscheint, werden Sie bei den folgenden Befehlen ebenfalls zu einer Bestätigung aufgefordert. • Start • Stopp • Einzelzyklus
Autoupdate Visu Profile	Mit dieser Option können Sie das automatische Update-Verhalten des Visualisierungsprofils konfigurieren. Wenn Sie ein SPS-Projekt öffnen, in dem ein veraltetes Visualisierungsprofil verwendet wird, erscheint im Meldungsfenster eine entsprechende Warnung („New Version found for Visualization profile“). ☑ : In einem solchen Fall wird die Visualisierungsprofil-Version automatisch auf die neueste Version gesetzt, falls die Option **Autoupdate Visu Profile** aktiviert ist. Bei einem solchen automatischen Update der Visualisierungsprofil-Version wird im Meldungsfenster eine entsprechende Warnung angezeigt (z. B. „Visualization profile set from ‚TwinCAT 3.1 Build 4020.10‘ to ‚TwinCAT 3.1 Build 4022.0‘“). ☐ (Voreinstellung): Wenn die Option **Autoupdate Visu Profile** deaktiviert ist, wird die Visualisierungsprofil-Version nicht automatisch geändert. Per Doppelklick auf die Warnung „New Version found for Visualization profile“ können Sie den ProfileUpdate-Dialog öffnen, in welchem Sie die Visualisierungsprofil-Version manuell ändern können.
Autoupdate Uml Profile	Mit dieser Option können Sie das automatische Update-Verhalten der UML-Compiler-Version konfigurieren. Wenn Sie ein SPS-Projekt öffnen, in dem eine veraltete UML-Compiler-Version verwendet wird, erscheint im Meldungsfenster eine entsprechende Warnung („neue Version für UML gefunden“). ☑ : In einem solchen Fall wird die UML-Compiler-Version automatisch auf die neueste Version gesetzt, falls die Option **Autoupdate Uml Profile** aktiviert ist. Bei einem solchen automatischen Update der UML-Compiler-Version wird im Meldungsfenster eine entsprechende Warnung angezeigt (z. B. „UML set from '4.0.2.0' to '4.0.2.1'“). ☐ (Voreinstellung): Wenn die Option **Autoupdate Uml Profile** deaktiviert ist, wird die UML-Compiler-Version nicht automatisch geändert. Per Doppelklick auf die Warnung „neue Version für UML gefunden“ können Sie den ProfileUpdate-Dialog öffnen, in welchem Sie die UML Compiler-Version manuell ändern können. Weitere Informationen hierzu finden Sie unter UML Compiler-Version.

Write Line IDs	Verfügbar ab TwinCAT 3.1 Build 4026 ☑ : Für die POUs des Projektes werden Line-IDs erzeugt und gespeichert (Standardverhalten bis TwinCAT 3.1 Build 4024). Über die Line-IDs können Code-Zeilen zu Maschinencode-Anweisungen zugeordnet werden, was u. a. für das Breakpoint-Handling benötigt wird. ☐ (Voreinstellung): Keine separaten Line-IDs werden erzeugt. Für die Zuordnung der Maschinencode-Anweisungen und das Breakpoint-Handling wird in diesem Fall die Zeilennummer genutzt. Bei Änderungen wie Leerzeichen oder Kommentaren ist daher ein Online Change erforderlich. Die globale Standardeinstellung für neue SPS Projekte bezüglich der Write Line IDs finden Sie unter Dialog Optionen - Write Options [▶ 1039]. Der Wert wird bei der Erstellung eines neuen SPS-Projekts einmalig in diese lokale Projekteinstellung übernommen.

Compatibility

Convert PLC Project to previous TwinCAT version	Verfügbar ab TwinCAT 3.1 Build 4026 In dem sich öffnenden Dialog können Sie eine TwinCAT-Version auswählen, in die das SPS Projekt konvertiert werden soll (TwinCAT 3.1 Build 4022 oder 4024). Nach Bestätigung der Konvertierung mit „Convert“ wird das Projekt geschlossen und kompatibel zu der ausgewählten Version abgespeichert. Beachten Sie, dass die Projektdaten bei der Konvertierung verändert werden und Einstellungen und Eigenschaften späterer Versionen verloren gehen. Die Konvertierung ist daher nicht für einen mehrfachen Wechsel zwischen verschiedenen Versionen geeignet.

17.4.5 SPS-Projekteinstellungen

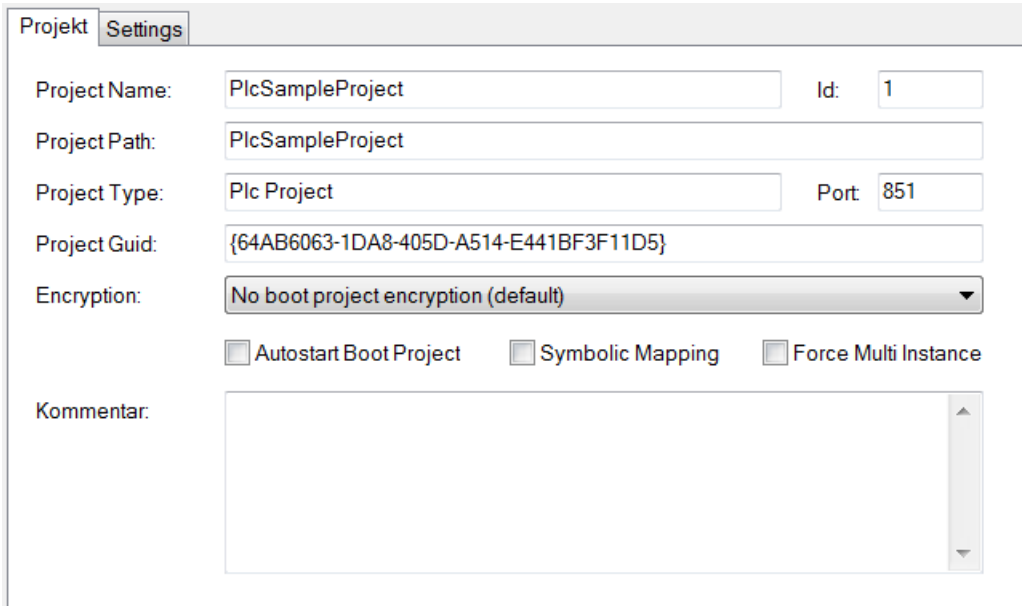
Funktion: Dieser Befehl öffnet einen Editor, in dem Projekteinstellungen definiert werden können.

Aufruf: Doppelklick auf das SPS-Projekt im **Projektmappen-Explorer**

Siehe auch:

- Dokumentation PLC: SPS-Projekt konfigurieren [▶ 62]

17.4.5.1 Registerkarte Projekt

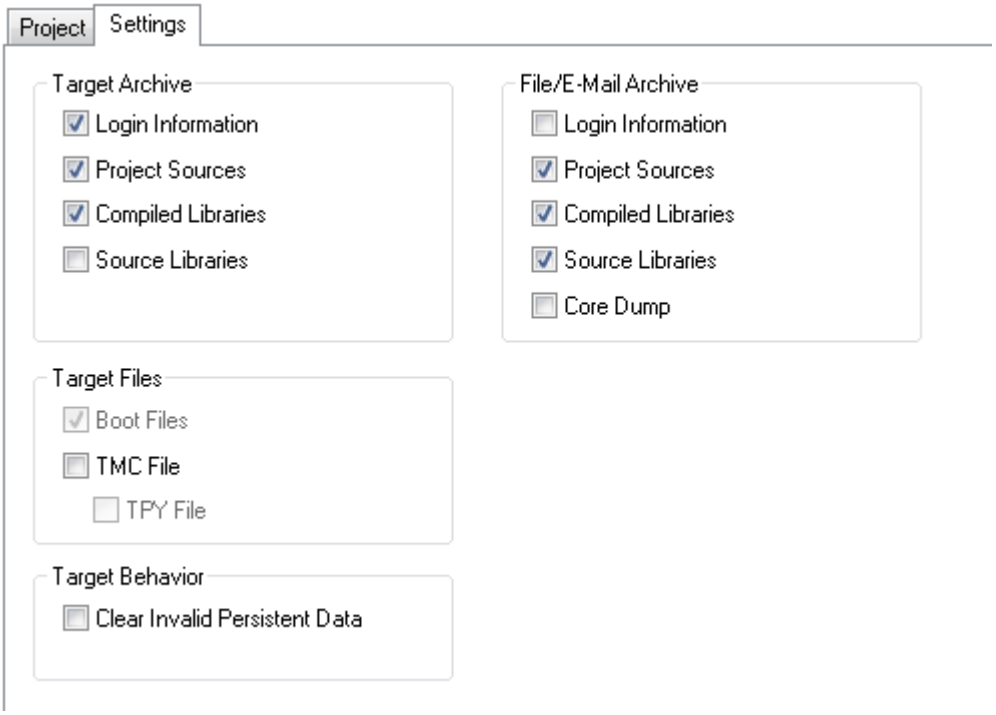


Project Name	Name des SPS-Projekts
Id	ID des SPS-Projekts
Project Path	Pfad zum Ablageort des SPS-Projekts
Project Type	Projekttyp
Port	AMS-Portnummer des Laufzeitsystems
Project Guid	GUID des SPS-Projets
Encryption	Verschlüsselung des Bootprojekts • No boot project encryption (default) • Encrypt boot project
Autostart Boot Project	☑ Nach dem Start der TwinCAT-Laufzeitumgebung, wird das SPS-Bootprojekt automatisch geladen und gestartet. Die Einstellung wird direkt auf das gerade ausgewählte Zielsystem übertragen. Die Einstellung wird nicht im TwinCAT-Projekt gespeichert. Die Option entspricht dem Befehl **Autostart Boot Project** im Kontextmenü des SPS-Projektknotens im Projektmappen-Explorer.
Symbolic Mapping	☑ Symbolisches Mapping ist aktiviert.
Force Multi Instance	☑ Möglichkeit zum Einloggen von mehrfachen Instanzen des SPS-Projekts aktiviert.
Kommentar	Kommentarfeld
Compiler Defines (Verfügbar ab TC3.1 Build 4024)	
Manual	Hier können eigene Compilerdefinitionen auf System Manager-Ebene festgelegt werden, die an das SPS-Projekt weitergegeben werden. Die Definitionen sind in den SPS-Projekteigenschaften unter der Kategorie Übersetzen als System-Compilerdefinitionen [▶ 951] eingetragen.
Implicit	☑ Die Namen der ausgewählten Projektvariante, sowie aller Gruppen, zu denen die Projektvariante gehört, werden automatisch als Compilerdefinition gesetzt und an das SPS-Projekt weitergegeben. Die Definitionen sind in den SPS-Projekteigenschaften unter der Kategorie Übersetzen als System-Compilerdefinitionen [▶ 951] eingetragen. **Hinweis:** Um diese Checkbox zu aktivieren, müssen die Defines für das Variantenmanagement freigegeben werden.

Siehe auch:

- Variantenmanagement: Konzept: Integration ins SPS-Projekt
- Befehl Eigenschaften (SPS-Projekt): Kategorie Übersetzen: System-Compilerdefinitionen [▶ 951]

17.4.5.2 Registerkarte Settings



Target Archive

Im Gruppenfeld **Target Archive** können Sie einstellen, welche Informationen beim Anlegen eines Bootprojekts zusammen mit anderen Daten auf das Zielsystem übertragen werden.

Login Information	COMPILEINFO-Datei, die die Compiler-Informationen des SPS-Projekts enthält.
Project Sources	Quellcode-Dateien des SPS-Projekts in lesbarer Quellcode-Form.
Compiled Libraries	Bibliotheken, die in kompilierter Form im SPS-Projekt verwendet werden.
Source Libraries	Bibliotheken, die in lesbarer Quellcode-Form im SPS-Projekt verwendet werden.

File/E-Mail Archive

Im Gruppenfeld **File/E-Mail Archive** können Sie einstellen, welche Informationen beim Archivieren eines SPS-Projekts [▶ 902], eines TwinCAT-Projekts [▶ 1113] oder einer Projektmappe [▶ 900] gespeichert werden. Wenn Sie die entsprechende Checkbox aktivieren, werden die in der folgenden Tabelle beschriebenen Dateien im Projektarchiv gespeichert.

Login Information	COMPILEINFO-Datei, die die Compiler-Informationen des SPS-Projekts enthält.
Project Sources	Quellcode-Dateien des SPS-Projekts in lesbarer Quellcode-Form.
Compiled Libraries	Bibliotheken, die in kompilierter Form im SPS-Projekt verwendet werden.
Source Libraries	Bibliotheken, die in lesbarer Quellcode-Form im SPS-Projekt verwendet werden.
Core Dump	Core Dump-Datei, die sich im Projektverzeichnis des SPS-Projekts befindet sowie die Compile-Info-Dateien, die sich im „_CompileInfo“-Ordner des Projektverzeichnisses befinden. Hinweis: Die Compile-Info-Dateien werden bei aktivierter „Core Dump [▶ 995]“-Einstellung ebenfalls im Archiv gespeichert, da diese Dateien benötigt werden, um den Core Dump nutzen zu können.

Weitergabe von Quellcode

Wenn Sie bei den Einstellungen für das Target- oder das File/E-Mail-Archiv konfiguriert haben, dass in einem dieser Archive die Projekt-Sourcen und/oder Source-Bibliotheken enthalten sein sollen, beachten Sie bei der Weitergabe/Auslieferung des Zielsystems oder bei Weitergabe des File/E-Mail-Archivs, dass die Projekt-Sourcen und/oder die im Projekt verwendeten Source-Bibliotheken (*.library) in lesbarer Quellcode-Form im ZIP-Archiv enthalten sind.

Bitte beachten Sie dies bei der Konfiguration der oben beschriebenen Einstellungen sowie bei der Speicherung und Referenzierung von Bibliotheken (*.library vs. *.compiled-library).

Weitere Informationen zum Thema Bibliotheksverwaltung finden Sie im Abschnitt Bibliotheken verwenden [▶ 278].

Informationen zu dem Thema Sourcecode-Verschlüsselung finden Sie in der Dokumentation Security Management.

Target Files

Im Gruppenfeld **Target Files** können Sie einstellen, welche Informationen beim Anlegen eines Bootprojekts auf das Zielsystem in den Ordner \Boot\Plc übertragen werden.

TMC File	TMC-Datei (TwinCAT Module Class) eines SPS-Projekts.
TPY File	TPY-Datei (enthält u. a. Projektinformationen, Routing-Informationen, Compiler-Informationen, Zielsystem-Informationen).

Target Behavior

Clear Invalid Persistent Data	Das Backup der gespeicherten persistenten Daten wird ignoriert. So wird sichergestellt, dass möglicherweise ungültige Daten nicht übernommen und verworfen werden. Siehe: Backup von persistenten Daten [▶ 970]

Backup von persistenten Daten

Persistente Daten werden beim TwinCAT-System-Stopp/Shutdown regulär in einer .bootdata-Datei im TwinCAT\Boot-Ordner gespeichert. Beim nächsten Systemstart (TwinCAT Run Mode) wird diese Datei eingelesen und die persistenten Variablen werden im Laufzeitsystem mit den Werten aus der Datei initialisiert. Die .bootdata-Datei wird vom System zu .bootdata-old umbenannt.

Diese Backup-Datei (.bootdata-old) der persistenten Daten wird beim Systemstart eingelesen, wenn die Datei (.bootdata) der persistenten Daten nicht vorhanden ist. Dieser Fall ist eine Ausnahme, kann aber z. B. dann eintreten, wenn ein IPC ohne USV einen Stromausfall erfährt und TwinCAT somit nicht ordnungsgemäß herunterfahren konnte.

- Wenn vorhersehbar ist, dass der Inhalt der Backup-Datei bei einem neuen Systemstart nicht nutzbar ist, können Sie die Option **Clear Invalid Persistent Data** aktivieren, um die Backup-Datei zu ignorieren. Dies kann z. B. der Fall sein, wenn Chargen-Informationen oder Werkzeugdaten in einer Produktion abgespeichert werden und aktuell sein müssen.
- Wenn die Struktur der persistenten Daten (deren Datentypen oder Symbolpfade im Programmcode) aufgrund von Online-Changes verändert wird, dann macht es keinen Sinn, später eine veraltete Datei der persistenten Daten zu laden. In diesem Fall sollten Sie die Option **Clear Invalid Persistent Data** vorausschauend aktivieren.

In beiden Fällen sollten Sie zusätzlich dafür sorgen, dass eine aktuelle Datei der persistenten Daten verfügbar ist. Hierfür gibt es einerseits Funktionsbausteine wie FB_WritePersistentData (PLC Lib Tc2_Utilities) und andererseits die USV-Absicherung für plötzliche Stromausfälle.

Grundsätzlich sollten bei der Verwendung von persistenten Daten die entsprechenden Flags (BootDataLoaded und OldBootData) aus der globalen Struktur PlcAppSystemInfo ausgewertet werden (siehe Dokumentation System > Globale Datentypen).

Wenn weder die reguläre Datei noch die Backup-Datei geladen werden können oder nicht vorhanden sind, werden die als PERSISTENT markierten Variablen ebenso wie andere „normale“ Variablen neu initialisiert, entweder mit ihren explizit vorgegebenen Initialwerten oder mit den Standardinitialisierungen.

Siehe auch:

- Dokumentation PLC: Remanente Variablen - PERSISTENT, RETAIN [▶ 726]

17.5 Erstellen

17.5.1 Befehl Projektmappe erstellen

Symbol: 

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codegenerierung für alle in der Projektmappe enthaltenen Projekte.

Aufruf: Menü **Erstellen** oder Kontextmenü der Projektmappe

Voraussetzung: Die Projektmappe ist selektiert.

Alle in der Projektmappe enthaltenen Projekte werden der Reihe nach übersetzt. Dies betrifft auch die unterhalb eines TwinCAT-Projekts eingebunden Projekte (SPS, C++, etc.). Die dabei für ein SPS-Projekt durchgeführten Schritte werden im Abschnitt Befehl SPS-Projekt erstellen [▶ 973] beschrieben.

17.5.2 Befehl Projektmappe neu erstellen

Funktion: Der Befehl startet den Übersetzungsprozess für alle in der Projektmappe enthaltenen Projekte, auch wenn sie zuletzt fehlerfrei übersetzt wurden.

Aufruf: Menü **Erstellen** oder Kontextmenü der Projektmappe

Voraussetzung: Die Projektmappe ist selektiert.

Bei einer Neuerstellung der Projektmappe wird die Projektmappe zunächst bereinigt (siehe auch: Befehl Projektmappe bereinigen [▶ 971]) und anschließend erstellt (siehe auch: Befehl Projektmappe erstellen [▶ 971]).

Siehe auch:

- Befehl SPS-Projekt neu erstellen [▶ 973]

17.5.3 Befehl Projektmappe bereinigen

Funktion: Der Befehl startet die Bereinigung für alle in der Projektmappe enthaltenen Projekte.

Aufruf: Menü **Erstellen** oder Kontextmenü der Projektmappe

Voraussetzung: Die Projektmappe ist selektiert.

Alle in der Projektmappe enthaltenen Projekte werden der Reihe nach bereinigt. Dies betrifft auch die unterhalb eines TwinCAT-Projekts eingebunden Projekte (SPS, C++, etc.). Die dabei für ein SPS-Projekt durchgeführten Schritte werden im Abschnitt Befehl SPS-Projekt bereinigen [▶ 973] beschrieben.

17.5.4 Befehl Überprüfe alle Objekte

Funktion: Der Befehl veranlasst einen Übersetzungslauf, also eine Syntaxprüfung, für alle Objekte, die sich im Projektbaum des SPS-Projekts befinden. Dies ist in erster Linie bei der Erstellung von Bibliotheken bzw. bei der Bearbeitung von Bibliotheksprojekten nützlich.

Aufruf: Kontextmenü des SPS-Projektobjekts (<SPS-Projektname> Project) im **Projektmappen-Explorer**

Im Gegensatz zu dem Befehl SPS-Projekt erstellen [▶ 973], bei dem nur die verwendeten Objekte überprüft werden, werden bei der Ausführung dieses Befehls alle Objekte des SPS-Projekts syntaktisch überprüft.

Der Befehl führt nicht zur Codegenerierung. Es wird auch keine Datei mit Informationen zum Übersetzungslauf im Projektverzeichnis angelegt.

17.5.5 Befehl TwinCAT-Projekt erstellen

Symbol: 

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codeerzeugung für das gerade aktive TwinCAT-Projekt.

Aufruf: Menü **Erstellen**, wenn aktuell ein TwinCAT-Projekt selektiert ist, oder Kontextmenü des TwinCAT-Projekts

Voraussetzung: Das TwinCAT-Projekt ist selektiert.

Alle in dem TwinCAT-Projekt enthaltenen Projekte (SPS, C++, etc.) werden der Reihe nach übersetzt. Die dabei für ein SPS-Projekt durchgeführten Schritte werden im Abschnitt Befehl SPS-Projekt erstellen [▶ 973] beschrieben.

Siehe auch:

- Befehl TwinCAT-Projekt neu erstellen [▶ 972]

17.5.6 Befehl TwinCAT-Projekt neu erstellen

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codeerzeugung für das gerade aktive TwinCAT-Projekt, auch wenn es zuletzt fehlerfrei übersetzt wurde.

Aufruf: Menü **Erstellen**, wenn aktuell ein TwinCAT-Projekt selektiert ist, oder Kontextmenü des TwinCAT-Projekts

Voraussetzung: Das TwinCAT-Projekt ist selektiert.

Bei einer Neuerstellung des Projekts wird das TwinCAT-Projekt zunächst bereinigt (siehe auch: Befehl TwinCAT-Projekt bereinigen [▶ 972]) und anschließend erstellt (siehe auch: Befehl TwinCAT-Projekt erstellen [▶ 972]).

17.5.7 Befehl TwinCAT-Projekt bereinigen

Funktion: Der Befehl löscht die lokale Übersetzungsinformation für das gerade aktive SPS-Projekt und aktualisiert das Sprachmodell aller Objekte.

Aufruf: Menü **Erstellen**, wenn aktuell ein TwinCAT-Projekt selektiert ist, oder Kontextmenü des TwinCAT-Projekts

Voraussetzung: Das TwinCAT-Projekt ist selektiert.

Alle in dem TwinCAT-Projekt enthaltenen Projekte (SPS, C++, etc.) werden der Reihe nach bereinigt. Die dabei für ein SPS-Projekt durchgeführten Schritte werden im Abschnitt Befehl SPS-Projekt bereinigen [▶ 973] beschrieben.

Siehe auch:

- Befehl TwinCAT-Projekt neu erstellen [▶ 972]

17.5.8 Befehl SPS-Projekt erstellen

Symbol: 

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codeerzeugung für das gerade aktive SPS-Projekt.

Aufruf: Menü **Erstellen**, wenn aktuell ein SPS-Projekt selektiert ist, oder Kontextmenü des SPS-Projektobjekts (<SPS-Projektname> Project) im **Projektmappen-Explorer**

Voraussetzung: Das SPS-Projekt ist selektiert.

Bei der Übersetzung führt TwinCAT eine syntaktische Prüfung aller verwendeten Objekte des SPS-Projekts durch. Der Übersetzungsvorgang wird automatisch immer durchgeführt, wenn Sie das Projekt mit einem geänderten Programm einloggen möchten. Nach Abschluss der Überprüfung zeigt TwinCAT eventuelle Fehlermeldungen oder Warnungen in der Ansicht Fehlerliste [▶ 935] an.

Außerdem wird beim Erstellen des Projektes die Übersetzungsinformation des SPS-Projekts angelegt und in einer lokalen Datei (*.compileinfo) im Projektordner gespeichert.

Wenn das Programm seit dem letzten fehlerfreien Übersetzungsprozess nicht mehr verändert wurde, wird es nicht neu übersetzt. Wenn die syntaktische Prüfung dennoch wiederholt werden soll, verwenden Sie den Befehl SPS-Projekt neu erstellen [▶ 973].

17.5.9 Befehl SPS-Projekt neu erstellen

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codeerzeugung für das gerade aktive SPS-Projekt, auch wenn es zuletzt fehlerfrei übersetzt wurde.

Aufruf: Menü **Erstellen**, wenn aktuell ein SPS-Projekt selektiert ist, oder Kontextmenü des SPS-Projektobjekts (<SPS-Projektname> Project) im **Projektmappen-Explorer**

Voraussetzung: Das SPS-Projekt ist selektiert.

Bei einer Neuerstellung des Projekts wird das Projekt zunächst bereinigt (siehe auch: Befehl SPS-Projekt bereinigen [▶ 973]) und anschließend erstellt (siehe auch: Befehl SPS-Projekt erstellen [▶ 973]).

17.5.10 Befehl SPS-Projekt bereinigen

Funktion: Der Befehl aktualisiert das Sprachmodell aller Objekte des gerade aktiven SPS-Projekts.

Aufruf: Menü **Erstellen**, wenn aktuell ein SPS-Projekt selektiert ist, oder Kontextmenü des SPS-Projektobjekts (<SPS-Projektname> Project) im **Projektmappen-Explorer**

Voraussetzung: Das SPS-Projekt ist selektiert.

Wenn das SPS-Projekt bereinigt wird, wird lediglich das Sprachmodell aller Objekte im SPS-Projekt aktualisiert. Die Übersetzungsinformation auf dem Zielsystem bleibt bestehen.

Siehe auch:

- Befehl SPS-Projekt neu erstellen [▶ 973]

17.6 Debuggen

17.6.1 Befehl Neuer Haltepunkt

Symbol: 

Funktion: Der Befehl öffnet den Dialog **Eigenschaften Haltepunkt**.

Aufruf: Menü **Debuggen**, Schaltfläche **New** in der Ansicht **Haltepunkt** (**PLC > Fenster > Haltepunkte**).

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Mit dem Befehl **Haltepunkt umschalten** können Sie im Onlinebetrieb einen neuen Haltepunkt direkt an der aktuellen Cursor-Position setzen

Siehe auch:

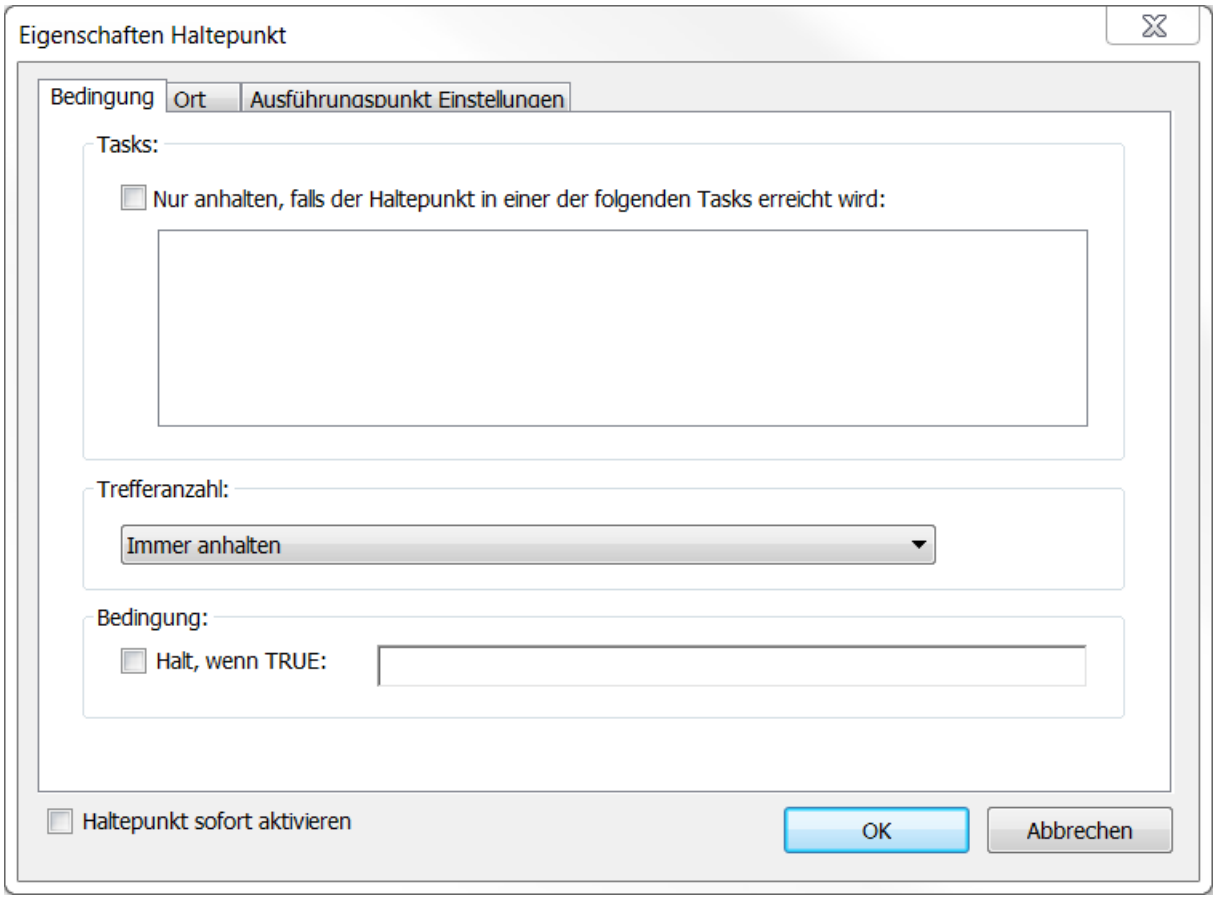
- Befehl Haltepunkt umschalten [▶ 978]
- Menü PLC: Befehl Haltepunkte [▶ 987]
- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

Dialog Eigenschaften Haltepunkt

Haltepunkt sofort aktivieren	☑ Der Haltepunkt ist aktiviert. ☐ Der Haltepunkt ist nicht aktiviert. Zur späteren Aktivierung klicken Sie in der Ansicht **Haltepunkte** auf die Schaltfläche .

Registerkarte Bedingung

Der Dialog legt fest, unter welchen Voraussetzungen die Programmabarbeitung am Haltepunkt stoppen soll.



Tasks

Nur anhalten, falls der Haltepunkt in einer der folgenden Tasks erreicht wird	: TwinCAT wertet den Haltepunkt nur aus, wenn er von bestimmten Tasks erreicht wird. Die gewünschten Tasks müssen aktiviert werden. Sie können beispielsweise eine einzige „Debug Task“ definieren und damit verhindern, dass beim Debugging auch andere Tasks betroffen werden, die den Baustein ebenfalls verwenden.

Trefferanzahl

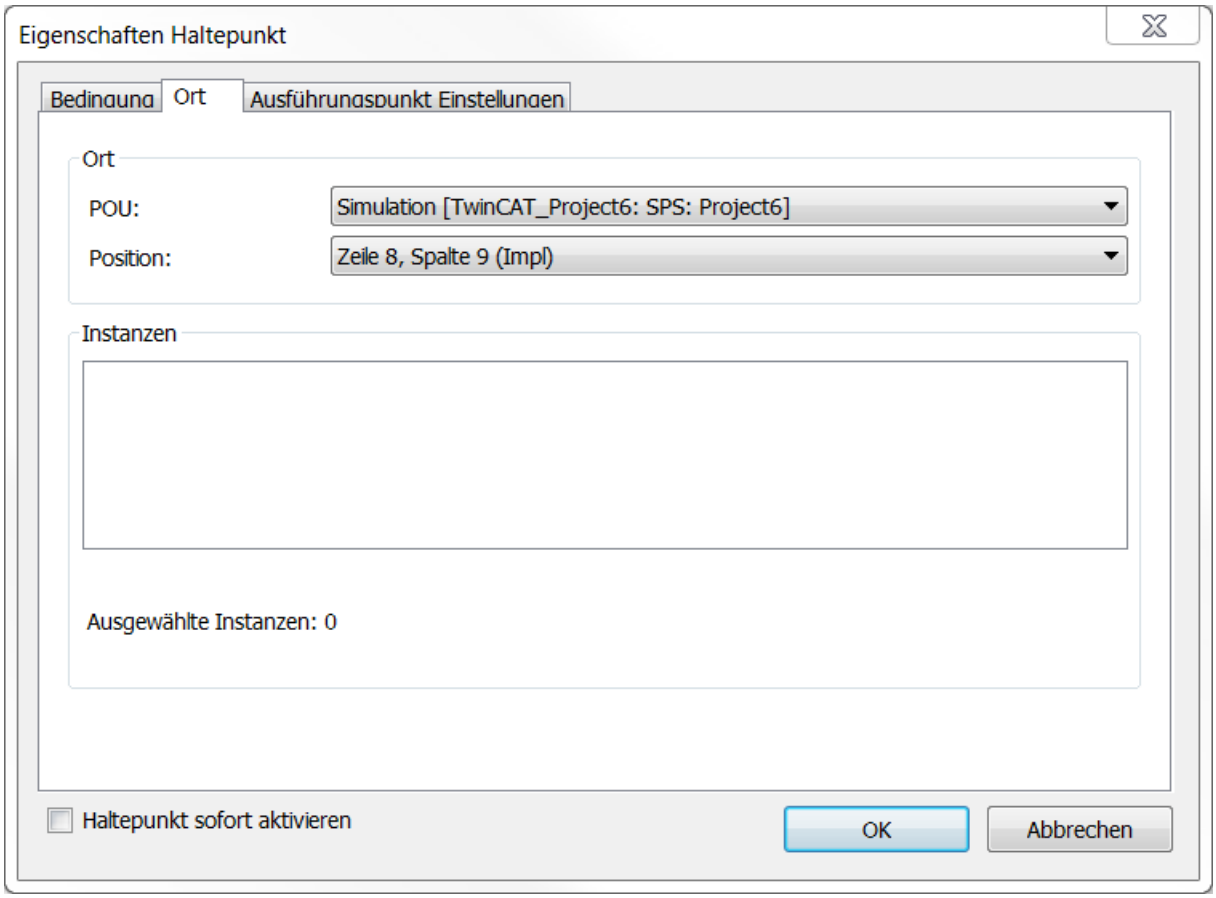
Trefferanzahl	Immer anhalten: Das Programm stoppt immer an diesem Haltepunkt. Alternativ: Das Programm hält am Haltepunkt an, wenn der Haltepunkt so oft getroffen wurde, wie im Folgenden definiert ist (gewünschte Trefferzahl eintragen oder aus der Nummernliste auswählen): • Anhalten, wenn die Trefferanzahl gleich • Anhalten, wenn die Trefferanzahl ein Vielfaches von • Anhalten, wenn die Trefferanzahl größer oder gleich

Bedingung

Halt, wenn TRUE	Definition bedingter Haltepunkte. Die Bedingung können Sie nur im Onlinebetrieb eintragen. : TwinCAT wertet die angegebene Bedingung aus und hält das Programm an diesem Haltepunkt nur an, wenn das Ergebnis TRUE ist. Als Bedingung können Sie gültige boolesche Ausdrücke eintragen. Beispiele: x>100, x[y]=z, a AND b, boolVar.

Die Verwendung bedingter Haltepunkte verlangsamt die Code-Ausführung, auch wenn die Bedingung nicht TRUE ist.

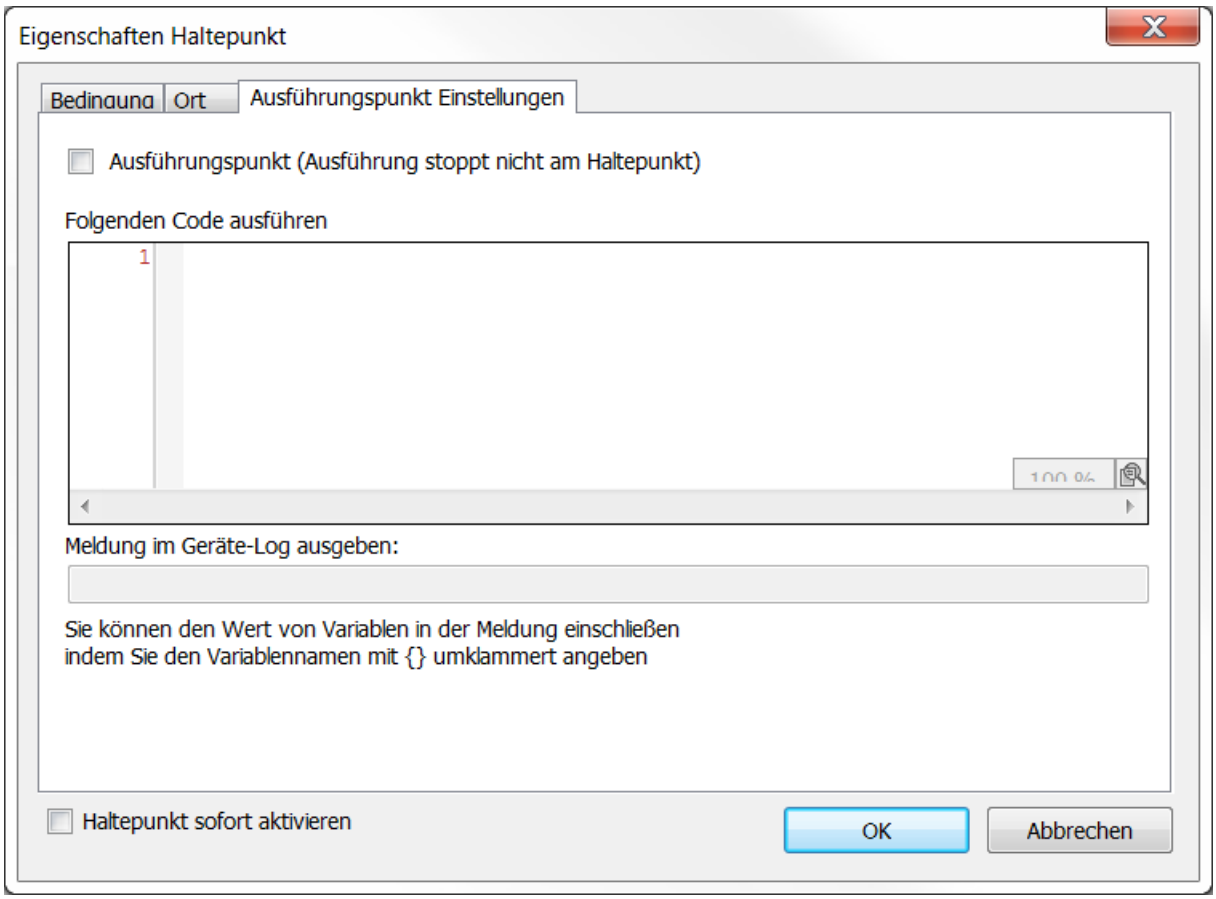
Registerkarte Ort



POU	Baustein des aktiven SPS-Projekts, in dem der Haltepunkt positioniert sein soll.
Position	Position des Haltepunkts in der POU. Angabe in Form von Zeilen- und Spaltennummern (Texteditor) oder als Netzwerk- oder Elementnummern.
Instanzen	Bei Funktionsbausteinen müssen Sie festgelegen, ob der Haltepunkt in der Implementierung oder in einer Instanz gesetzt werden soll ☑ TwinCAT setzt den Haltepunkt in der Instanz. Bei dieser Option wählen Sie den Instanzpfad aus. ☐ TwinCAT setzt den Haltepunkt in der Implementierung.

Registerkarte Ausführungspunkt

Hier kann ein bestehender Haltepunkt in einen Ausführungspunkt umgewandelt werden.



Ausführungspunkt (Ausführung stoppt nicht am Haltepunkt)	☑: Der Haltepunkt wird zum Ausführungspunkt. Die Abarbeitung hält an diesem Punkt nicht an, es wird jedoch der angegebene Code ausgeführt. aktiviert: ●, deaktiviert: ○
Folgenden Code ausführen	Code, der beim Erreichen des Ausführungspunkts ausgeführt wird. Schleifenkonstruktionen (For, While) und IF- oder CASE-Ausdrücke sind nicht möglich.
Meldung im Geräte-Log ausgeben	Diese Option ist nicht verfügbar.

Siehe auch:

- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

17.6.2 Befehl Haltepunkt bearbeiten

Symbol: 

Funktion: Der Befehl öffnet den Dialog **Eigenschaften Haltepunkte**.

Aufruf: Menü **Debuggen**, Schaltfläche  in der Ansicht **Haltepunkte** (**PLC > Fenster > Haltepunkte**)

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Der Cursor steht auf einem Haltepunkt.

Siehe auch:

- Befehl Neuer Haltepunkt > Dialog Eigenschaften Haltepunkt [▶ 974]

- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

17.6.3 Befehl Haltepunkt aktivieren

Symbol: 

Funktion: Der Befehl aktiviert einen deaktivierten Haltepunkt.

Aufruf: Menü **Debuggen**, Schaltfläche  in der Ansicht **Haltepunkte** (**PLC > Fenster > Haltepunkte**)

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Der Cursor steht auf einem deaktivierten Haltepunkt.

Siehe auch:

- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

17.6.4 Befehl Haltepunkt deaktivieren

Symbol: 

Funktion: Der Befehl deaktiviert einen aktivierten Haltepunkt.

Aufruf: Menü **Debuggen**, Schaltfläche  in der Ansicht **Haltepunkte** (**PLC > Fenster > Haltepunkte**)

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Der Cursor steht auf einem aktivierten Haltepunkt.

Siehe auch:

- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

17.6.5 Befehl Haltepunkt umschalten

Tastaturkürzel: **[F9]**

Funktion: Der Befehl setzt einen Haltepunkt oder löscht einen bestehenden.

Aufruf: Menü **Debuggen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Der Cursor steht auf einem Haltepunkt.

Siehe auch:

- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

17.6.6 Befehl Prozedurschritt

Symbol: 

Tastaturkürzel: **[F10]**

Funktion: Der Befehl führt die Anweisung, an der das Programm aktuell steht, aus und hält vor der nächsten Anweisung im Programmierbaustein an.

Aufruf: Menü **Debuggen**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Das Programm steht an einer Halteposition (Debugbetrieb).

Wenn die auszuführende Anweisung einen Aufruf enthält (von einem Programm, einer Funktionsbaustein-Instanz, einer Funktion, einer Methode oder einer Aktion), wird der untergeordnete Programmierbaustein vollständig in einem Schritt durchlaufen und zum Aufruf zurückgekehrt. Vor der nächsten Anweisung (in der nächsten Codezeile) wird angehalten.

Wählen Sie den Befehl **Einzelschritt**, um in einen untergeordneten Baustein zu springen und diesen schrittweise auszuführen.

Siehe auch:

- Befehl Einzelschritt [▶ 979]
- Dokumentation PLC: Schrittweises Abarbeiten eines Programms (Stepping) [▶ 224]

17.6.7 Befehl Einzelschritt

Symbol: 

Tastaturkürzel: **[F11]**

Funktion: Der Befehl führt die Anweisung, an der das Programm aktuell steht, aus und hält vor der nächsten Anweisung.

Aufruf: Menü **Debuggen**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Das Programm steht an einer Halteposition (Debugbetrieb).

Wenn die auszuführende Anweisung einen Aufruf enthält (von einem Programm, einer Funktionsbaustein-Instanz, einer Funktion, einer Methode oder einer Aktion), wird in diesen untergeordneten Programmierbaustein gesprungen. Dessen Code erscheint in einem eigenen Editor. Die erste Anweisung dort wird ausgeführt und vor der nächsten Anweisung wird angehalten. Die neue aktuelle Halteposition ist dann im aufgerufenen Programmierbaustein.

Wählen Sie den Befehl **Prozedurschritt**, um im aktuell aktiven Programmierbaustein zu bleiben und den Aufruf in einem Schritt zu durchlaufen.

Siehe auch:

- Befehl Prozedurschritt [▶ 978]
- Dokumentation PLC: Schrittweises Abarbeiten des Programms (Stepping) [▶ 224]

17.6.8 Befehl Ausführen bis Rücksprung

Symbol: 

Tastaturkürzel: **[Umschalt] + [F11]**

Funktion: Der Befehl führt das Programm bis zum nächsten Rücksprung aus und hält danach an.

Aufruf: Menü **Debuggen**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Das Programm steht an einer Halteposition (Debugbetrieb).

Wenn die aktuelle Halteposition in einem untergeordneten Programmierbaustein ist, wird dieser bis zum Ende durchlaufen. Dann wird zur Aufrufstelle im aufrufenden Programmierbaustein zurückgesprungen und dort angehalten (in der Zeile mit dem Aufruf).

Wenn die aktuelle Halteposition im Hauptprogramm ist, wird der Programmierbaustein bis zum Ende durchlaufen. Dann wird zurück an den Anfang (an den Programmstart an die erste Codezeile im Programmierbaustein) gesprungen und dort angehalten.

Siehe auch:

- Dokumentation PLC: Schrittweises Abarbeiten des Programms (Stepping) [▶ 224]

17.6.9 Befehl Ausführen bis Cursor

Symbol: 

Funktion: Der Befehl führt ein Programm bis zu einer mit dem Cursor gekennzeichneten Position aus.

Aufruf: Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Das Programm steht an einer Halteposition (Debugbetrieb). Sie haben mit dem Cursor eine beliebige Codezeile in einem beliebigen Programmierbaustein gekennzeichnet.

Die Anweisungen, die zwischen der aktuellen Halteposition und der Cursorposition liegen, werden in einem Schritt ausgeführt. Dann hält die Ausführung an der Cursorposition an, die damit zur nächsten Halteposition wird. Beachten Sie, dass die Codezeile, an der Sie den Cursor gesetzt haben, erreicht, aber nicht ausgeführt wird.

Siehe auch:

- Dokumentation PLC: Schrittweises Abarbeiten des Programms (Stepping) [▶ 224]

17.6.10 Befehl Nächste Anweisung anzeigen

Symbol: 

Funktion: Der Befehl zeigt die Programmanweisung an, die im nächsten Schritt abgearbeitet wird.

Aufruf: Menü **Debuggen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Das Programm steht an einer Halteposition. Die Halteposition ist in einer für Sie nicht sichtbaren Codezeile.

Der Befehl bewirkt, dass das Fenster mit der aktuellen Halteposition, die im Code gelb und mit dem Symbol  gekennzeichnet ist, aktiv und die Halteposition sichtbar wird. Das ist nützlich, wenn Sie viele Editoren geöffnet haben und die Halteposition sich verdeckt in einem nicht aktiven Editor befindet.

Siehe auch:

- Dokumentation PLC: Schrittweises Abarbeiten des Programms (Stepping) [▶ 224]

17.6.11 Befehl Nächste Anweisung festlegen

Symbol: 

Funktion: Der Befehl legt fest, welche Anweisung als nächstes ausgeführt wird.

Aufruf: Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb. Das Programm steht an einer Halteposition (Debugbetrieb). Sie haben mit dem Cursor eine beliebige Codezeile in einem beliebigen Programmierbaustein gekennzeichnet

Die mit Cursor gekennzeichnete Codezeile wird zur aktuellen Halteposition, ohne die Anweisungen dazwischen oder die angesprungene Anweisung auszuführen.

Siehe auch:

- Dokumentation PLC: Schrittweises Abarbeiten des Programms (Stepping) [▶ 224]

17.7 TwinCAT

17.7.1 Befehl Konfiguration aktivieren

Symbol: 

Funktion: Der Befehl aktiviert eine neue Konfiguration. Die vorherige alte Konfiguration wird überschrieben.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

Innerhalb des Bestätigungsfensters, das nach Ausführung dieses Befehls erscheint, können Sie einstellen, ob für alle SPS-Projekte des TwinCAT-Projekts die Einstellung **Autostart Boot Projekt** aktiviert werden soll.

Siehe auch:

- Befehl Bootprojekt aktivieren
- Befehl Autostart Bootprojekt

17.7.2 Befehl Restart TwinCAT System

Symbol: 

Funktion: Der Befehl startet TwinCAT im Run-Modus.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

17.7.3 Befehl Restart TwinCAT (Config Mode)

Symbol: 

Funktion: Der Befehl startet TwinCAT im Konfigurationsmodus (Config-Modus).

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

17.7.4 Befehl Reload Devices

Symbol: 

Funktion: Der Befehl lädt die angelegten I/O-Geräte.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

17.7.5 Befehl Scan

Symbol: 

Funktion: Der Befehl startet einen Gerätescan. Das System sucht nach verfügbaren I/O-Geräten, verbundenen „Boxen“ und gegebenenfalls Busmodulen und IP-Link-Erweiterungsmodulen.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

Voraussetzung: In der TwinCAT-Projektstruktur im **Projektmappen-Explorer** ist das Objekt „I/O“ markiert.

17.7.6 Befehl Toggle Free Run State

Symbol: 

Funktion: Der Befehl setzt gefundene I/O-Geräte in den Free-Run-Modus. Das bedeutet, dass z. B. I/O-Kanäle von Busklemmen auf einen bestimmten Status gesetzt (geschrieben) werden können, ohne dass ein SPS-Projekt oder eine sonstige auslösende Task aktiv ist.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

Voraussetzung: Das System befindet sich aktuell im Konfigurationsmodus.

Wenn sich das Zielsystem zuvor im Run-Modus befand, muss der Befehl **Reload Devices** einmal ausgeführt werden, bevor die I/O-Treiber für das Gerät in den Free-Run-Status gesetzt werden können.

17.7.7 Befehl Show Online Data

Symbol: 

Funktion: Der Befehl stellt eine Verbindung mit dem ausgewählten Zielsystem her und stellt die auf dem Zielsystem aktiven Parameterwerte und Einstellungen in den entsprechenden Ansichten dar.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

17.7.8 Befehl Choose Target System

Funktion: Drop-down-Liste zur Auswahl des Zielgeräts für die Steuerungsapplikation.

Aufruf: TwinCAT XAE Base Symbolleistenoptionen

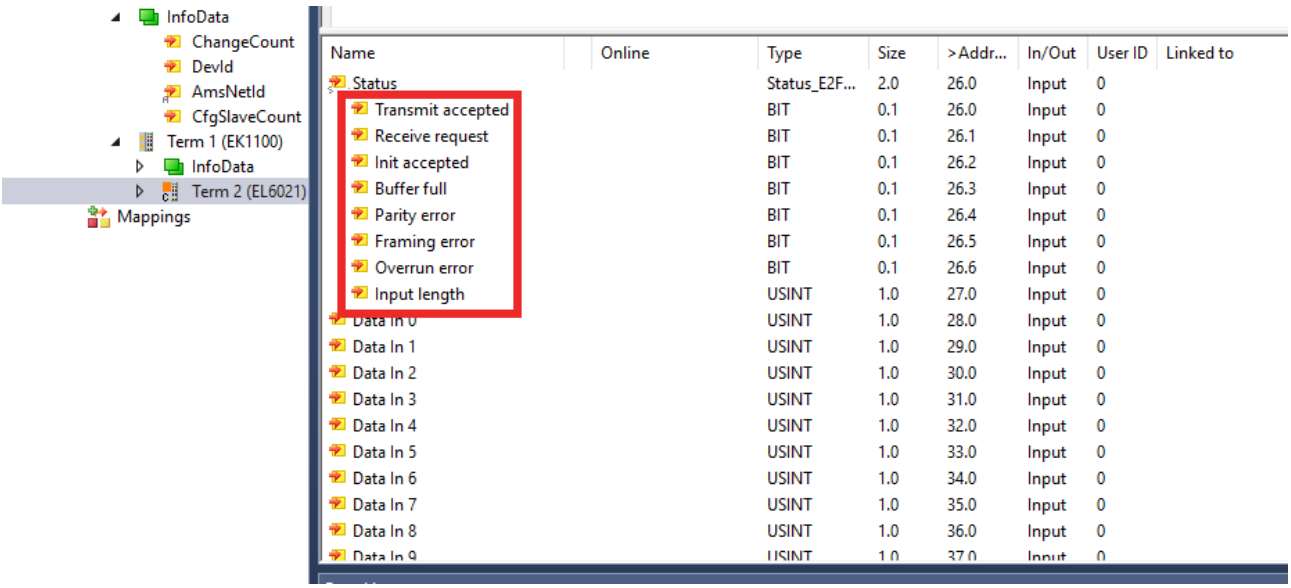
Wenn der Steuerungscode direkt in Ihre lokale Laufzeit Ihres Programmiergeräts geladen werden soll, wählen Sie den Eintrag <Lokal>. Wenn Sie ein anderes Zielgerät auswählen wollen, wählen Sie in der Drop-down-Liste den Eintrag **Zielsystem wählen**.

17.7.9 Befehl Show Sub Items

Symbol: 

Funktion: Der Befehl zeigt in der Übersichtsansicht eines Geräts die Unterelemente eines Elements mit deren Eigenschaften und Werte an. Der Befehl kann aktiviert oder deaktiviert werden. Der Befehl bezieht sich nicht auf die Darstellung der Elemente im TwinCAT-Projektbaum.

Aufruf: Menü **TwinCAT, TwinCAT Base XAE Symbolleistenoptionen**



17.7.10 Befehl Software Protection

Symbol: 

Funktion: Der Befehl öffnet den Dialog **Software Protection**.

Aufruf: Menü **TwinCAT**

In dem Software-Protection-Dialog können Sie die Security- und Benutzer-Einstellungen eines TwinCAT-Projekts definieren.

Weitere Informationen zu den Security- und Benutzer-Einstellungen finden Sie in der Dokumentation Software Protection.

17.7.11 Befehl Hide Disabled Items

Symbol: 

Funktion: Der Befehl ermöglicht es, deaktivierte Objekte im gesamten Projektbaum unsichtbar und wieder sichtbar zu schalten. Auf diese Weise können ausschließlich aktive Objekte dargestellt und die Übersichtlichkeit innerhalb des Projektbaums erhöht werden.

Aufruf: Menü **TwinCAT**, **TwinCAT XAE Base Symbolleistenoptionen**

17.8 PLC

17.8.1 Fenster

17.8.1.1 Befehl Überwachungsliste <n>

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Überwachungsliste <n>**. Eine Überwachungsliste können Sie mit Variablen aus Ihrem Projekt füllen, um im Onlinebetrieb für diese Variablen innerhalb einer einzigen Ansicht die Werte monitoren, forcen oder schreiben zu können. n kann 1,2,3,4 sein, das heißt, dass Sie bis zu vier Überwachungslisten konfigurieren können.

Aufruf: Menü **PLC > Fenster**

Siehe auch:

- Dokumentation PLC: Überwachungslisten verwenden [▶ 239]

17.8.1.2 Befehl Alle Forces anzeigen

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Alle Forces anzeigen**, die eine spezielle Form einer Überwachungsliste ist.

Aufruf: Menü **PLC > Fenster**

Voraussetzung: Ein SPS-Projekt ist im Offline- oder Onlinebetrieb geöffnet.

Die Ansicht enthält alle derzeit zum Forcen vorbereiteten und alle geforcten Variablen des SPS-Projekts in einer Liste. In der Liste sind die Aktionen möglich, die auch in anderen Überwachungslisten möglich sind.

Alles Forces anzeigen

Tabellarische Anzeige aller geforcten und für das Forcen vorbereiteten Variablen der Applikation

Ausdruck	Variablenname
Datentype	Datentyp der Variablen
Wert	Aktuell geforcter Wert der Variablen
Vorbereiteter Wert	Für das Forcen vorbereiteter Wert
Überschriebener Wert am Anfang des Zyklus	Bei Eingängen wird der eigentliche Wert bereits vor dem Ausführen des Anwendercodes durch den Forcewert überschrieben. Somit ist dies der eigentliche Wert. Bei Ausgängen ist dies der geforcte Wert.
Überschriebener Wert am Ende des Zyklus	Bei Ausgängen ist dies der Wert, der im Zyklus berechnet wird. Dieser Wert wird jedoch am Ende des Zyklus durch den Forcewert überschrieben. Bei Eingängen ist dies der geforcte Wert.

Zusätzlich gibt es im Auswahlmenü **Force aufheben** folgende Befehle:

- **Forcen aufheben und alle ausgewählten Werte beibehalten**: Für alle selektierten Einträge der Liste werden die Variablen auf den geforcten Wert gesetzt und das Forcen aufgehoben.
- **Forcen aufheben und alle ausgewählten Werte wiederherstellen**: Für alle selektierten Einträge der Liste werden die Variablen auf den Wert zurückgesetzt, den sie vor dem Forcen hatten, und das Forcen wird aufgehoben.

Siehe auch:

- Dokumentation PLC: Forcen und Schreiben von Variablen [▶ 225]
- Dokumentation PLC: Überwachungslisten verwenden [▶ 239]

17.8.1.3 Befehl Querverweisliste

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Querverweisliste**.

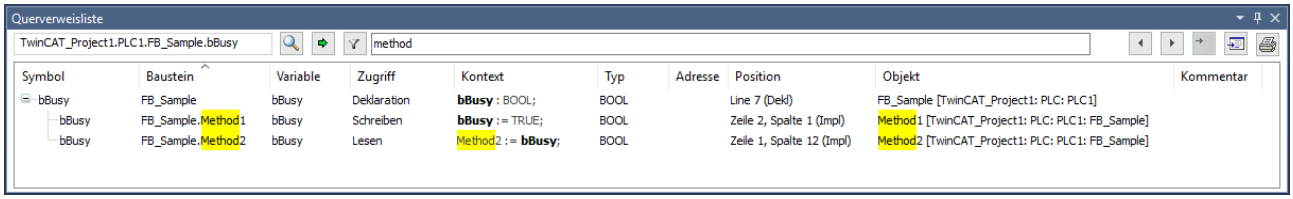
Aufruf: Menü **PLC > Fenster**

Ansicht Querverweisliste

Die Ansicht zeigt eine Liste der Querverweise im Projekt für ein Symbol. Das Symbol kann eine Variable, eine POU (Programm, Funktionsbaustein, Funktion) oder ein anwenderspezifischen Datentyp (DUT) sein.

Dabei bietet die Querverweisliste grundsätzlich zwei Sucharten:

- Textsuche: Durch die Eingabe eines Symbolnamens werden die Querverweise aller Symbole im Projekt mit diesem Namen angezeigt. Falls mehrere Symbole mit gleichem Namen gefunden werden, kann die Anzeige über das Kontextmenü auf einzelne Deklarationen eingeschränkt werden.
- Deklarationssuche: Das Symbol kann über die Eingabehilfe oder durch die Eingabe eines qualifizierten Pfads, beispielsweise MAIN.nVar ausgewählt werden. Danach werden nur die Verwendungsstellen dieses Symbols angezeigt, auch wenn noch andere Symbole mit dem gleichen Namen existieren.



Werkzeugleiste

Name (Eingabefeld)	Symbolname (Variablenname, Bausteinname, DUT-Name) Eingabemöglichkeiten: • Auswahl eines deklarierten Symbols über die Eingabehilfe (Schaltfläche ) • Händisches Eingeben des Symbolnamens. Auslösen der Suche über die Schaltfläche  oder die [Eingabetaste]. Sie können für die Textsuche die Platzhalter „*“ (beliebig viele Zeichen) oder „?“ (genau ein beliebiges Zeichen) in Kombination mit einer Teilzeichenkette eines Variablenbezeichners verwenden. Verwenden Sie „%“, wenn Sie nach IEC-Adressen suchen wollen. Beispiele: „%MW8“, „%M*“ Zusätzliche Möglichkeiten von außerhalb der Ansicht **Querverweisliste**: • Verwenden des Kontextmenübefehls **Alle Verweise suchen**, wenn der Name eines deklarierten Symbols in einem Editor selektiert ist oder der Cursor im Namen steht. • Automatisch, wenn der Name eines deklarierten Symbols in einem Editor selektiert ist oder der Cursor im Namen steht. Eine automatische Suche ist auch möglich, wenn das Objekt im Projektbaum selektiert ist. Voraussetzung: Die Ansicht **Querverweisliste** ist geöffnet und die TwinCAT-Option **Querverweise automatisch bei Selektionsänderung aktualisieren**, Kategorie **Intelligentes Kodieren**, ist aktiviert. Folgende Eingaben sind gültig: • Variablenname, einfach oder qualifiziert: z. B. „nVar“, „MAIN.nVar“ • Bausteinname: z. B. „MAIN“, „FB_MyFB“ • DUT-Name: z. B. „ST_MySTRUCT“ • Zeichenfolgen in Kombination mit Platzhalter „*“ (beliebige Zeichen) oder „?“ (genau ein beliebiges Zeichen): Beispiel: „nVar*“ betrifft nVar1, nVarGlob2, nVar45 usw... „nVar?“ betrifft nVar1, nVar2, nVarX usw., nicht aber nVarGlob2, nVar45 usw. • „%<IEC-Adresse>“: TwinCAT sucht nach Variablen, die dieser Adresse zugewiesen sind, und direkte Speicherzugriffe. Beispiel: „%QB0“, „%Q0 := 2" Groß/Kleinschreibung sowie Leerzeichen an Anfang und Ende der Eingabezeichenfolge werden nicht berücksichtigt.
	Eingabehilfe öffnen zur Auswahl eines Symbols.
	Querverweise finden: Die Suche wird durchgeführt.
	Spalten definieren, in denen nach der Zeichenfolge gesucht wird
Filtern (Eingabefeld)	Zeichenfolge, nach der in den angewählten Spalten gesucht wird Die Fundstellen werden gelb markiert. Querverweise ohne diese Zeichenfolge werden ausgeblendet.
	Quellposition des vorherigen Querverweises anzeigen
	Quellposition des nächsten Querverweises anzeigen
	Ergebnisse auf aktuelle Deklaration beschränken Verfügbar, wenn für ein Symbol mehrere Deklarationen gefunden wurden. Begrenzt die Anzeige auf die Deklaration, die Sie gerade in der Liste selektiert haben.
	Quellposition des selektierten Querverweises anzeigen: Der Fokus springt zur Verwendungsstelle des Symbols.
	Querverweisliste drucken: Der Standarddialog zum Einrichten eines Druckauftrags erscheint.

Tabelle der gefundenen Querverweise

Symbol	Die Fundstellen für die Symbole (Variablen, POUs, DUTs) werden nach ihrer Deklaration gruppiert. Die Deklarationsstelle bildet den Wurzelknoten, darunter eingerückt erscheinen die Verwendungsstellen im Projekt. Dabei wird genau der Ausdruck angezeigt, den das Symbol an der Verwendungsstelle hat. Beispiel: Gibt es im Projekt eine globale Variable „nVar“ und in einem Baustein eine lokal deklarierte Variable „nVar“, dann erscheinen nach einer Textsuche nach den Querverweisen zwei Wurzelknoten-Einträge in der Liste und darunter jeweils die Verwendungsstellen der Variable „nVar“.
Baustein	Bausteinname, DUT-Name; auch beispielsweise Taskname im Falle eines Bausteinaufrufs in der Taskkonfiguration.
Variable	Reiner Variablenname. Beispiel: „nVar“.
Zugriff	Art des Zugriffs auf die Variable an der Verwendungsstelle: Deklaration / Lesen / Schreiben / Aufruf. Sonderfall für Pointer: Eine Zuweisung der Art `pSample := ADR(nVar1)` wird bei Suche nach „nVar1“ als Schreiben | Adresse angezeigt. Grund: Eventuelle Schreibzugriffe auf „pSample“ werden bei Suche nach „nVar1“ nicht angezeigt. Über die Pointervariable sind auch Schreibzugriffe möglich.
Kontext	Kontext der Verwendung der Variablen. Beispiel: „nVar := 1“
Typ	Datentyp der Variablen.
Adresse	IEC-Adresse, falls der Variablen zugewiesen. Beispiel: „AT%QB0“
Position	Position der Verwendungsstelle innerhalb des Editors der betroffenen POU: beispielsweise Zeilennummer, Netzwerknummer, Deklarationsteil oder Implementierungsteil. Beispiel: „Zeile 1, Spalte 1 (Impl)“ oder „Zeile 9 (Dekl)“.
Objekt	POU-Name + in eckigen Klammern der komplette Pfad der Verwendungsstelle. Beispiel: „MAIN [TwinCAT_SampleProject: SPS: SamplePLCProject]“
Kommentar	Kommentar, falls in der Deklaration der Variable vorhanden.

Die Suche liefert alle Fundstellen im Projekt sowie in eingehängten, nicht kompilierten Bibliotheken.

Befehle im Kontextmenü der Querverweisliste

Quellcodeposition anzeigen	Öffnet den betroffenen Baustein und markiert die Verwendungsstelle: Für Root-Einträge die Deklaration, für darunter liegende Kind-Einträge die jeweilige Verwendungsstelle. Alternativ können Sie auf eine Zeile doppelklicken.
Ergebnisse auf aktuelle Deklaration beschränken	Grenzt im Fall von mehreren gefundenen Deklarationen die Anzeige der Ergebnisse auf die ausgewählte Symboldeklaration ein.
Alles erweitern	In der Liste erscheinen alle einzelnen Fundstellen.
Alles reduzieren	In der Liste erscheinen nur die Wurzelknoten aller Fundstellen.

Siehe auch:

- Befehl Ergebnisse auf aktuelle Deklaration beschränken
- Befehl Alles einklappen
- Befehl Alles ausklappen
- Dokumentation PLC: Verwendungsstellen mit der Querverweisliste finden [▶ 165]

17.8.1.4 Befehl Haltepunkte

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Haltepunkte**.

Aufruf: Menü **PLC > Fenster**

Ansicht Haltepunkte

Die Ansicht zeigt Ihnen eine Übersicht aller definierten Haltepunkte einer Applikation. Innerhalb der Ansicht stehen Ihnen alle Befehle für Haltepunkte zur Verfügung.

Tabelle der aktuellen Haltepunkte

Applikation	Wählen Sie das gewünschte SPS-Projekt aus der Liste.
POU	Name des Bausteins, der den Haltepunkt enthält.
Position	Haltepunkt-Position innerhalb der POU • Texteditor: Zeilen- plus Spaltennummer • Grafischer Editor: Netzwerk oder Elementnummer „(Impl)“ im Fall von Funktionsbausteinen zeigt an, dass der Haltepunkt in der Implementierung des Funktionsbausteins sitzt, nicht in einer Instanz.
Instanzpfad	Vollständiger Objektpfad der Haltepunkt-Position.
Tasks	Namen der Tasks, bei deren Ausführung der Haltepunkt wirksam sein soll. Wenn keine Einschränkung gilt, steht hier „(alle)“.
Bedingung	• Immer anhalten: Keine zusätzliche Aktivierungsbedingung definiert. Der Haltepunkt ist immer aktiv. • Boolescher Ausdruck. Der Ausdruck muss TRUE liefern, damit der Haltepunkt aktiv ist.
Trefferzählbedingung	Angabe, wann (in welcher Abhängigkeit von der Trefferanzahl) der Haltepunkt wirksam werden soll.
Aktuelle Trefferanzahl	Angabe, wie oft der Haltepunkt während der Ausführung bis jetzt bereits durchlaufen („getroffen“) wurde.

Werkzeugleiste

	Neuer Haltepunkt (entspricht dem Befehl Neuer Haltepunkt [▶ 973] im Menü **Debuggen**)	Öffnet den Dialog **Eigenschaften Haltepunkt**
	Haltepunkt löschen	Entfernt den Haltepunkt. Verwechseln Sie den Befehl nicht mit dem Befehl Deaktivieren.
	Haltepunkt aktivieren/deaktivieren (entspricht dem Befehl Haltepunkt aktivieren [▶ 978] und dem Befehl Haltepunkt deaktivieren [▶ 978] im Menü **Debuggen**)	Schaltet den Haltepunkt oder Ausführungspunkt zwischen Status „aktiviert“ und „deaktiviert“ hin und her. • Haltepunkt aktiviert • Haltepunkt deaktiviert • Ausführungspunkt aktiviert • Ausführungspunkt deaktiviert Im Gegensatz zum **Haltepunkt löschen** bleibt ein deaktivierter Haltepunkt in der Liste erhalten und kann wieder aktiviert werden.
	Eigenschaften	Öffnet den Dialog **Eigenschaften Haltepunkt** erscheint zur Bearbeitung der Haltepunkt-Parameter. Im Onlinebetrieb können Sie hier den Haltepunkt zum Ausführungspunkt umwandeln.
	Gehe zur Quellcodeposition	Öffnet die Online-Ansicht des betreffenden Bausteins. Der Cursor steht an der Haltepunkt-Position.
	Alle Haltepunkte löschen	Löscht alle Haltepunkte und Ausführungspunkte der Applikation. Die Liste wird geleert. Nicht zu verwechseln mit Deaktivieren!
	Alle Haltepunkte aktivieren	Aktiviert alle gerade deaktivierten Haltepunkte und Ausführungspunkte.
	Alle Haltepunkte deaktivieren	Deaktiviert alle gerade aktivierten Haltepunkte und Ausführungspunkte. Die Punkte bleiben in der Liste und können wieder aktiviert werden.

Siehe auch:

- Befehl Neuer Haltepunkt > Dialog Eigenschaften Haltepunkt [▶ 974]
- Befehl Haltepunkt umschalten [▶ 978]
- Dokumentation PLC: Haltepunkte verwenden [▶ 222]

17.8.1.5 Befehl Aufrufliste

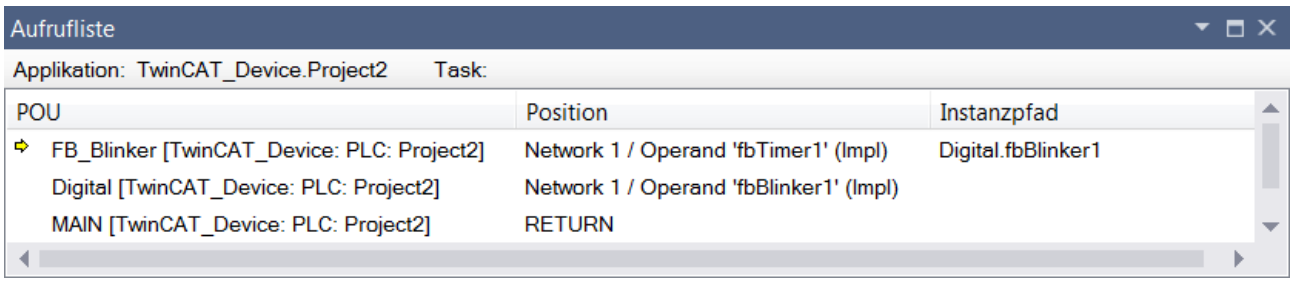
Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Aufrufliste**.

Aufruf: Menü **PLC > Fenster**

Ansicht Aufrufliste

Diese Ansicht ist nützlich, wenn Sie Programme schrittweise ausführen wollen. Sie zeigt die aktuell erreichte Position mit vollständigem Aufrufpfad.



Applikation	Name das aktiven SPS-Projekts, die den gerade erreichten Programmbaustein kontrolliert.
Task	Name der Task, die den gerade erreichten Programmbaustein kontrolliert.

POU	Name des Programmbausteins, in dem die Programmausführung steht. Die erste Zeile in der Liste beschreibt die aktuelle Ausführungsposition. Sie ist mit einem gelben Pfeil markiert. Wenn diese Position in einem Baustein liegt, der von einem anderen aufgerufen wird, wird die Position des Aufrufs in der zweiten Zeile beschrieben. Wenn der Aufrufer wiederum von einem anderen Baustein aufgerufen wird, wird diese Aufrufposition in der dritten Zeile beschrieben usw.
Position	Position innerhalb des Programmbausteins, an der die Programmausführung steht • Zeilen- und Spaltennummer bei Texteditoren • Netzwerk- oder Elementnummer bei grafischen Editoren
Instanzpfad	Instanz, in der die Programmausführung steht.

Die Aufrufliste ist auch im Offlinebetrieb verfügbar und im normalen Onlinebetrieb, wenn Sie gerade keine Debugging-Funktionen benützen. In diesem Fall enthält sie die zuletzt während einer schrittweisen Ausführung angezeigte Position, allerdings in „gegrauter“ Schrift.

Die Ansicht **Aufrufbaum** liefert im Unterschied zur **Aufrufliste** jederzeit Aufrufinformationen zu einem Baustein.

Siehe auch:

- Dokumentation PLC: Verwenden von Haltepunkten [▶ 222]

17.8.1.6 Befehl Speicher

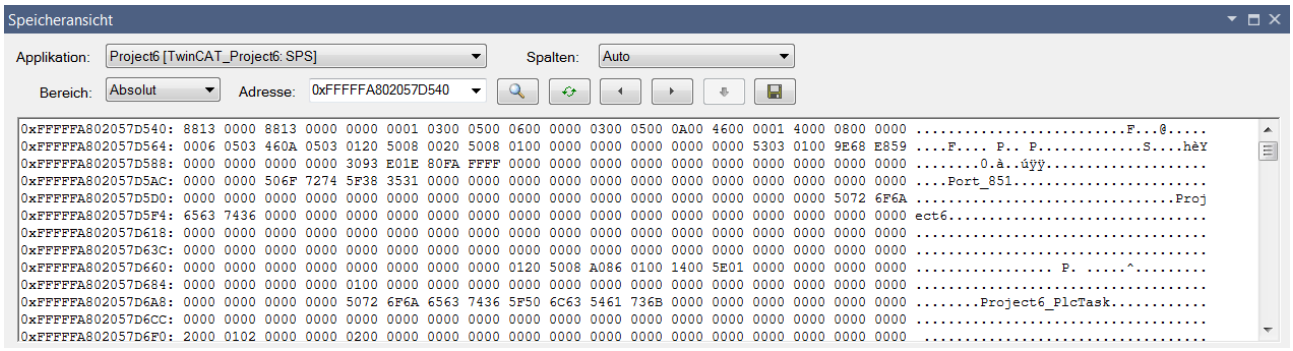
Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Speicheransicht**.

Aufruf: Menü **PLC > Speicher**

Voraussetzung: Die Steuerung unterstützt die Funktionalität grundsätzlich. Mindestens eine Applikation ist geladen und im Onlinebetrieb.

Ansicht Speicheransicht



Applikation	Auswahl des SPS-Projektes, für die die Speicheransicht dargestellt werden soll. Mit diesem Projekt müssen Sie auf der Steuerung eingeloggt sein. Es muss nicht das „aktive SPS-Projekt“ sein.
Bereich	• Absolut: Speicher wird direkt und vollständig adressiert. Die Adresse steht im Eingabefeld daneben. • Area <i>: Speicherbereiche der Steuerung, beginnend mit Area 0. Speicherbereiche, die ausschließlich für Code reserviert sind, werden nicht angezeigt.
Adresse	Absolute Startadresse des Core Dumps Voraussetzung: In Bereich ist Absolut ausgewählt.
Offset	Adressversatz zum gewählten Speicherbereich in Byte, zum Beispiel 0x0200, 16#0200 oder als Dezimalzahl 512 Voraussetzung: In Bereich ist ein Speicherbereich ausgewählt, zum Beispiel Area 0. TwinCAT bietet alle aktuell verwendeten Speicherbereiche zur Auswahl an. Speicherbereiche, die ausschließlich für Code reserviert sind, werden nicht angezeigt.
	Adresse für eine Variable herausfinden: Eingabeunterstützung zur Auswahl einer IEC-Variablen erscheint. Wenn Sie eine Variable ausgewählt haben, belegt TwinCAT die Startadresse mit der Variablenadresse vor.
	Speicheransicht laden/aktualisieren
	Vorheriges Speichersegment anzeigen: Navigieren zum vorherigen Speichersegment
	Nächstes Speichersegment anzeigen: Navigieren zum nächsten Speichersegment
	**Hinweis TwinCAT prüft nicht, ob die Änderungen zulässig sind. Sie können die Applikation durch unbedachte Änderungen zum Absturz bringen** Änderungen auf SPS laden: TwinCAT überträgt die neuen Daten auf die Steuerung. Voraussetzung: Sie haben ein oder mehrere Bytes in der Speicheransicht überschrieben.
	Speicherinhalt in Datei speichern: Dialog Speicherinhalt als Binärdatei erscheint. Wählen Sie einen Ablageort.
Spalten	Anzahl der Spalten der hexadezimale Darstellung des Speicherauszugs. Je Spalte werden zwei Bytes angezeigt. Bei Auto passt sich die Spaltenanzahl an die Fenstergröße an. Rechts davon werden die Daten als Text angezeigt.

17.8.1.7 Befehl Aufrufbaum

Symbol: 

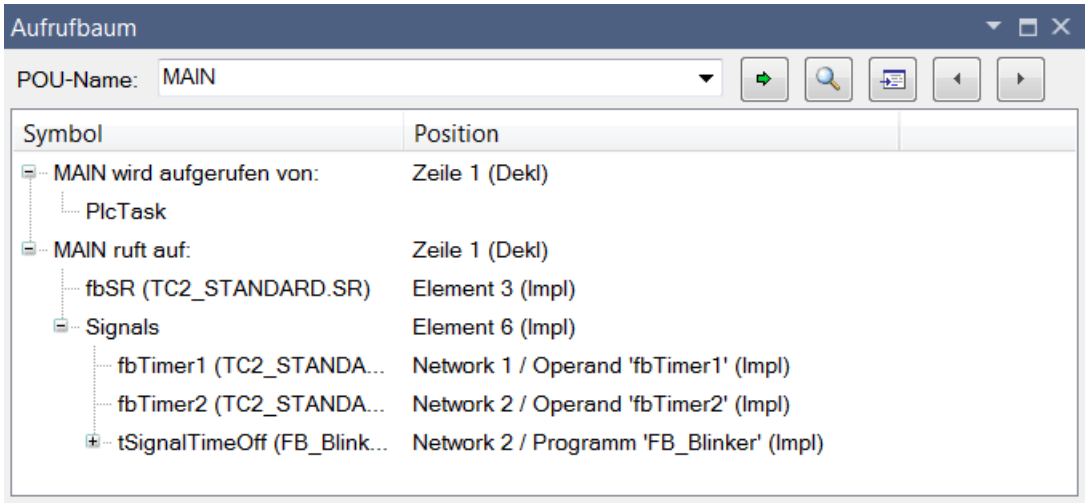
Funktion: Der Befehl öffnet die Ansicht **Aufrufbaum.**

Aufruf: Menü **PLC > Fenster**

Ansicht Aufrufbaum

Der Aufrufbaum steht jederzeit bereits vor dem Kompilieren (Übersetzen) der Applikation zur Verfügung. Er ist eine statische Darstellung der Aufrufer und der Aufrufe des Bausteins, den Sie explizit angeben. Somit enthält der Baum immer zwei Wurzelknoten, unter denen die jeweilige Aufrufabfolge als nacheinander eingerückte Einträge zu sehen ist. Rekursive Aufrufe sind in dieser Baumdarstellung schnell erkennbar.

Beispiel für einen Aufrufbaum (1) für Baustein (2) MAIN:



- (3) Knoten „<Bausteinname> wird aufgerufen von:“
- (4) Knoten „<Bausteinname> ruft auf:“

POU-Name	Name des Programmbausteins, kann manuell, oder durch Ziehen aus einer anderen Ansicht, oder mithilfe der Schaltfläche eingegeben werden. Die Auswahlliste enthält die zuletzt eingegebenen Bausteinnamen.

Symbolleiste und Tastaturbedienung

Baustein finden	TwinCAT sucht nach dem in „Bausteinname“ angegebenen Baustein und stellt seine Aufrufer und seine Aufrufe dar.
Baustein aus Eingabehilfe entnehmen	Der Dialog **Eingabehilfe** erscheint zur Auswahl eines Bausteinaufrufs oder Instanzaufrufs. Der Aufrufbaum wird nach der Auswahl automatisch aktualisiert.
Quellcodeposition des selektierten Bausteins anzeigen	TwinCAT springt zur Verwendungsstelle des Bausteins im Quellcode Ihres Programms.
Quellcodeposition des nächsten Bausteins anzeigen Quellcodeposition des vorherigen Bausteins anzeigen	Die Selektion im Aufrufbaum springt zum nächsten oder vorherigen Baustein in der Aufrufstruktur. Gleichzeitig wird die dazugehörige Quellcodeposition im jeweiligen Editor geöffnet. Ein Doppelklick auf einen Eintrag im Aufrufbaum öffnet ebenfalls die zugehörige Quellcodeposition.

Darstellung des Aufrufbaums

Position	Für die Wurzelknoten im Aufrufbaum: Zeilennummer der Deklaration („Dekl“) des Bausteins. Für die Aufrufer oder Aufrufe unter den Wurzelknoten: Je nach Implementierungssprache Zeilennummer, Spaltennummer, Netzwerknummer ihrer Position.

Kontextmenü für den gerade im Baum selektierten Eintrag

Alles reduzieren	Die ausgeklappten Einträge im Aufrufbaum werden bis auf die beiden Wurzelknoten zugeklappt.
Quellcodeposition anzeigen	TwinCAT springt zur Verwendungsstelle des Bausteins im Quellcode Ihres Programms.
Als neuen Wurzelknoten setzen	Der im Aufrufbaum selektierte Eintrag erscheint in „Bausteinname“. Der Baum wird automatisch für die neuen Wurzelknoten angepasst.

Im Unterschied zum statischen Aufrufbaum, der jederzeit Aufrufinformationen zu einem Baustein liefert, ist die Ansicht **Aufrufliste** für unmittelbare Information während der schrittweisen Abarbeitung eines Programms vorgesehen. Die Aufrufliste zeigt immer den kompletten Aufrufpfad der gerade erreichten Position.

17.8.1.8 Befehl Online Change Memory Reserve Settings

Funktion: Der Befehl öffnet die Ansicht **Online Change Memory Reserve**.

Aufruf: Menü **PLC > Fenster**.

In der Ansicht werden für Funktionsbausteine Speicherreserven für den Online-Change konfiguriert.

Applikation durchsuchen	• Durchsucht das ausgewählte SPS-Projekt nach Funktionsbausteinen und zeigt sie im Bereich **Funktionsbausteine** an • Aktualisiert den Bereich **Funktionsbausteine**, nachdem das SPS-Projekt erneut übersetzt wurde • Aktualisiert den Bereich **Funktionsbausteine** nach einem Online-Change
Auswahlliste mit den SPS-Projekten des geöffneten TwinCAT-Projekts	Auswahl des SPS-Projekts, dessen Funktionsbausteine in dieser Ansicht angezeigt und/oder bearbeitet werden sollen

Funktionsbausteine:

Alle	Alle Funktionsbausteine des ausgewählten SPS-Projekts werden angezeigt.
Keine Speicherreserve	Alle Funktionsbausteine mit Speicherreserve 0 Bytes werden angezeigt.
<Speicherreserve> Bytes	Anzeige aller Funktionsbausteine mit der Anzahl Bytes, die in **Speicherreserve** definiert ist.
Informationen zu den Funktionsbausteinen Bei der Selektion eines Bausteins für die Konfiguration der Speicherreserve ist auch eine Mehrfachauswahl möglich.	
Funktionsbaustein	Name des Funktionsbausteins
Größe	Größe des Funktionsbausteins Größe einer Instanz des Funktionsbausteins Angabe in Bytes
Instanzanzahl	Anzahl der Instanzen des Funktionsbausteins im Projekt
Speicherreserve	Anzeige der Speicherreserve pro Instanz des Funktionsbausteins
Zusätzlicher Speicher für alle Instanzen	Produkt aus **Instanzanzahl** und **Speicherreserve**
Verbleibende Speicherreserve	Anzahl Bytes, die noch pro Funktionsbaustein-Instanz als Reserve zur Verfügung stehen

Einstellungen:

Speicherreserve (in Bytes)	Eingabefeld für die Speicherreserve für den selektierten Funktionsbaustein. Angabe in Bytes Voraussetzung: Das SPS-Projekt befindet sich noch nicht auf der Steuerung oder Sie haben durch eine Klick auf die Schaltfläche **Erlauben** im Bereich **Bearbeitung erlauben** die Änderung der Speicherreserve erlaubt.
Für Auswahl anwenden	Die **Speicherreserve (in Bytes)** wird dem Funktionsbaustein zugeordnet und die Tabellenspalte **Speicherreserve** wird aktualisiert. Bei Mehrfachauswahl wird der eingegebene Wert jedem Funktionsbaustein zugeordnet. Um die Spalten **Größe**, **Instanzanzahl**, **Zusätzlicher Speicher für alle Instanzen** und **Verbleibende Größe der Speicherreserve** zu aktualisieren, wählen Sie zunächst den Befehl **Erstellen > Erstellen** und klicken Sie anschließend auf die Schaltfläche **Applikation durchsuchen**.

Bearbeitung erlauben:

Erlauben	Das Eingabefeld **Speicherreserve (in Bytes)** wird editierbar. Diese Schaltfläche wird in **Bearbeitbar** geändert.

Information:

Anzahl FBs	Gesamtanzahl der Funktionsbausteine in dem SPS-Projekt
Zusatzspeicher für alle Instanzen	Summe der Speicherreserven aller Funktionsbaustein-Instanzen des SPS-Projekts Angabe in Bytes

Siehe auch:

- Dokumentation PLC: SPS-Projekt programmieren > Speicherreserve für Online-Change konfigurieren [▶ 139]

17.8.1.9 Befehl SPS Lesezeichen

Symbol: 

Funktion: Der Befehl öffnet die Ansicht **Lesezeichen**.

Aufruf: Menü **PLC > Fenster**

Vorheriges Lesezeichen	Springt zum Lesezeichen, das in der Tabelle eine Zeile über der selektierten Zeile angezeigt wird, und öffnet die entsprechende POU im Editor.
Nächstes Lesezeichen	Springt zum Lesezeichen, das in der Tabelle eine Zeile unter der selektierten Zeile angezeigt wird, und öffnet die entsprechende POU im Editor.
	Löscht das selektierte Lesezeichen aus der Tabelle und in der entsprechenden POU.

Auflistung der Lesezeichen des Projekts mit den Informationen Lesezeichen, Objekt und Position:

Lesezeichen	Von TwinCAT vergebene Bezeichnung der Lesezeichen in nummerierter, aufsteigender Reihenfolge: Lesezeichen_0, Lesezeichen_2 usw. Wenn das Lesezeichen selektiert ist und Sie in das Feld klicken, wird es editierbar und Sie können die Bezeichnung des Lesezeichens ändern.
Objekt	Name und Projektpfad der POU, in der das Lesezeichen gesetzt ist
Position	Position des Lesezeichens innerhalb der POU Beispiel: Zeile3, Spalte 1 (Impl) (Impl): im Implementierungsteil der POU (Decl): im Deklarationsteil der POU

Sie können Reihenfolge der Lesezeichen per Drag-and-drop verändern.

Wenn Sie eine Zeile doppelklicken, öffnet TwinCAT das entsprechende Objekt im Editor und springt zu diesem Lesezeichen.

Siehe auch

- Befehl Vorheriges Lesezeichen [▶ 998]
- Befehl Nächstes Lesezeichen [▶ 997]
- Lesezeichen setzen und verwenden [▶ 166] (Dokumentation „PLC“)

17.8.2 Core Dump

17.8.2.1 Befehl Core Dump erzeugen

Funktion: Der Befehl bewirkt, dass TwinCAT zunächst prüft, ob bereits eine Core Dump-Datei auf dem Zielsystem verfügbar ist.

- Wenn auf dem Zielsystem eine Core Dump-Datei verfügbar ist, bietet TwinCAT Ihnen an, diese Datei ins Projektverzeichnis zu laden. Die Abfrage, ob die Core Dump-Datei vom Zielsystem geladen werden soll, können Sie mit drei unterschiedlichen Möglichkeiten beantworten.
  - Ja: Falls die Core Dump-Datei des Zielsystems zu dem aktuell eingeloggten SPS-Projekt passt, lädt TwinCAT die Core Dump-Datei vom Zielsystem ins Projektverzeichnis. Diese Datei können Sie öffnen, indem Sie das SPS-Projekt anschließend ausloggen und den Befehl Core Dump laden [▶ 996] verwenden.
  - Nein: Eine neue Core Dump-Datei wird im Projektverzeichnis erzeugt. Die Voraussetzung hierfür ist, dass das SPS-Projekt gerade an einem Haltepunkt steht oder ein Ausnahmefehler aufgetreten ist.
  - Abbrechen: Die Erzeugung einer Core Dump-Datei wird abgebrochen.
- Wenn auf dem Zielsystem keine Core Dump-Datei verfügbar ist, veranlasst TwinCAT das Erzeugen einer neuen Dump-Datei mit den aktuellen SPS-Projektdaten im Projektverzeichnis. Die Voraussetzung hierfür ist, dass das SPS-Projekt gerade an einem Haltepunkt steht oder ein Ausnahmefehler aufgetreten ist.

Die erzeugte Core Dump-Datei wird direkt im SPS-Projektverzeichnis abgelegt: <SPS-Projektname>.<SPS-Projekt-GUID>.core

Aufruf: Menü PLC > Core Dump

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Automatische Erzeugung eines Core Dumps auf dem Zielsystem

Wenn das SPS-Projekt, das auf einem Zielsystem läuft, gerade nicht in einer Entwicklungsumgebung eingeloggt ist, erzeugt das Laufzeitsystem im Falle eines Ausnahmefehlers automatisch einen Core Dump auf dem Zielsystem. Diese Datei befindet sich im Boot-Ordner des Zielsystems (standardmäßig unter C:\TwinCAT\3.1\Boot\Plc).

Das automatische Laden dieser Dump-Datei vom Zielsystem in das lokale Projektverzeichnis ist mit Hilfe des **Befehls Core Dump erzeugen** möglich. Das manuelle Kopieren der Core Dump-Datei vom Zielsystem auf den Entwicklungsrechner ist ebenfalls möglich.

Das Anzeigen des Dumps mit Hilfe des Befehls Core Dump laden [▶ 996] kann für die (nachträgliche) Fehleranalyse verwendet werden.

Core Dump nur nutzbar mit zugehöriger Compile-Info-Datei

Wenn Sie eine Core Dump-Datei archivieren oder abspeichern, beachten Sie bitte, dass zum Laden eines Core Dumps das zugehörige Projekt und die zugehörige Compile-Info-Datei (*.compileinfo-Datei, die z.B. beim Erstellen des Projekts im „_CompileInfo“-Ordner abgelegt wird) vorliegen müssen. Falls dies nicht der Fall ist, kann TwinCAT den Dump später nicht mehr verwenden.

Bitte beachten Sie hierzu auch die Einstellungsmöglichkeiten auf der Registerkarte Settings [▶ 969]. Mit Hilfe der Einstellung **Core Dump** können Sie konfigurieren, ob die Core Dump-Datei, die sich möglicherweise im Projektverzeichnis befindet, zusammen mit den verfügbaren Compile-Info-Dateien in einem TwinCAT-Dateiarchiv gespeichert werden soll.

Siehe auch:

- Dokumentation PLC: SPS-Projekt zur Laufzeit > Fehleranalyse mit Core Dump [▶ 259]
- Befehl Core Dump laden [▶ 996]

17.8.2.2 Befehl Core Dump laden

Funktion: TwinCAT durchsucht das Projektverzeichnis nach Core Dump-Dateien.

- Wenn TwinCAT im Projektverzeichnis eine Core Dump-Datei findet, werden Sie gefragt, ob Sie diesen Core Dump laden oder nach einer Dump-Datei browsen möchten.
- Wenn TwinCAT im Projektverzeichnis keine Core Dump-Datei findet, können Sie nach einer anderen Dump-Datei browsen.

Das Laden ins Projekt bewirkt, dass eine Online-Ansicht des SPS-Projekts erscheint, mit dem Stand, den das SPS-Projekt zum Zeitpunkt des Erzeugens des Core Dumps hatte. Darin können Sie die Variablenwerte nachträglich ansehen. Zudem ist der Aufrufbaum verfügbar.

Aufruf: Menü PLC > Core Dump

Voraussetzung: Die Applikation ist im Offlinebetrieb.

Sie können die Core Dump-Ansicht nur über den Befehl Core Dump schließen [▶ 997] wieder schließen. Der Befehl Ausloggen ist in dieser Ansicht nicht wirksam!

Core Dump nur nutzbar mit zugehöriger Compile-Info-Datei

Wenn Sie eine Core Dump-Datei archivieren oder abspeichern, beachten Sie bitte, dass zum Laden eines Core Dumps das zugehörige Projekt und die zugehörige Compile-Info-Datei (*.compileinfo-Datei, die z.B. beim Erstellen des Projekts im „_CompileInfo“-Ordner abgelegt wird) vorliegen müssen. Falls dies nicht der Fall ist, kann TwinCAT den Dump später nicht mehr verwenden.

Bitte beachten Sie hierzu auch die Einstellungsmöglichkeiten auf der Registerkarte Settings [▶ 969]. Mit Hilfe der Einstellung **Core Dump** können Sie konfigurieren, ob die Core Dump-Datei, die sich möglicherweise im Projektverzeichnis befindet, zusammen mit den verfügbaren Compile-Info-Dateien in einem TwinCAT-Dateiarchiv gespeichert werden soll.

Siehe auch:

- Dokumentation PLC: SPS-Projekt zur Laufzeit > Fehleranalyse mit Core Dump [▶ 259]
- Befehl Core Dump erzeugen [▶ 995]
- Befehl Core Dump schließen [▶ 997]

17.8.2.3 Befehl Core Dump schließen

Funktion: Der Befehl schließt die gerade in der Entwicklungsumgebung geöffnete Core Dump-Ansicht des SPS-Projekts.

Aufruf: Menü PLC > Core Dump

Voraussetzung: Das SPS-Projekt ist im Offlinebetrieb und Sie haben eine Core Dump-Datei ins Projekt geladen.

Siehe auch:

- Dokumentation PLC: SPS-Projekt zur Laufzeit > Fehleranalyse mit Core Dump [▶ 259]

17.8.3 SPS Lesezeichen

17.8.3.1 Befehl Lesezeichen ein-/ausschalten

Symbol: 

Funktion: Der Befehl setzt oder löscht ein Lesezeichen an der aktuellen Position.

Aufruf: Menü **SPS > SPS Lesezeichen**

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in einer Programmzeile.

Siehe auch

- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.3.2 Befehl Nächstes Lesezeichen

Symbol: 

Funktion: Der Befehl springt in der Ansicht **Lesezeichen** und im Projekt zum nächsten Lesezeichen und öffnet die entsprechende POU. Die Reihenfolge, in der die Lesezeichen angesprungen werden, entspricht dabei der Reihenfolge der Lesezeichen in der Tabelle der Ansicht **Lesezeichen**.

Aufruf:

- Menü **SPS > SPS Lesezeichen**
- Schaltfläche  **Nächstes Lesezeichen** in der Ansicht **Lesezeichen**

Voraussetzung:

- Ein Projekt ist geöffnet
- Die Ansicht **Lesezeichen** ist geöffnet

Siehe auch:

- Befehl SPS Lesezeichen [▶ 994]
- Befehl Nächstes Lesezeichen (aktiver Editor) [▶ 998]
- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.3.3 Befehl Vorheriges Lesezeichen

Symbol: 

Funktion: Der Befehl springt in der Ansicht **Lesezeichen** und im Projekt zum vorherigen Lesezeichen und öffnet die entsprechende POU. Die Reihenfolge, in der die Lesezeichen angesprungen werden, entspricht dabei der Reihenfolge der Lesezeichen in der Tabelle der Ansicht **Lesezeichen**.

Aufruf:

- Menü **SPS > SPS Lesezeichen**
- Schaltfläche  Vorheriges Lesezeichen in der Ansicht Lesezeichen

Voraussetzung:

- Ein Projekt ist geöffnet
- Die Ansicht **Lesezeichen** ist geöffnet

Siehe auch:

- Befehl SPS Lesezeichen [▶ 994]
- Befehl Vorheriges Lesezeichen (aktiver Editor) [▶ 998]
- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.3.4 Befehl Alle Lesezeichen löschen

Symbol: 

Funktion: Der Befehl löscht alle Lesezeichen des geöffneten Projekts.

Aufruf: Menü **SPS > SPS Lesezeichen**

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in der POU.

Siehe auch:

- Befehl Alle Lesezeichen löschen (aktiver Editor) [▶ 999]
- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.3.5 Befehl Nächstes Lesezeichen (aktiver Editor)

Symbol: 

Funktion: Der Befehl springt zum nächsten Lesezeichen im aktiven Editor.

Aufruf: Menü **SPS > SPS Lesezeichen**

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in der POU

Siehe auch:

- Befehl Nächstes Lesezeichen [▶ 997]
- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.3.6 Befehl Vorheriges Lesezeichen (aktiver Editor)

Symbol: 

Funktion: Der Befehl springt zum vorherigen Lesezeichen im aktiven Editor.

Aufruf: Menü **SPS > SPS Lesezeichen**

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in der POU

Siehe auch:

- Befehl Vorheriges Lesezeichen [▶ 998]
- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.3.7 Befehl Alle Lesezeichen löschen (aktiver Editor)

Symbol: 

Funktion: Der Befehl löscht alle Lesezeichen im aktiven Editor

Aufruf: Menü **SPS > SPS Lesezeichen**

Voraussetzung: Eine POU ist im Editor geöffnet und der Cursor steht in der POU.

Siehe auch:

- Befehl Alle Lesezeichen löschen [▶ 998]
- Lesezeichen setzen [▶ 166] (Dokumentation „PLC“)

17.8.4 Befehl Laden

Funktion: Der Befehl bewirkt ein Übersetzen des aktiven SPS-Projekts mit anschließendem Download auf die Steuerung.

Aufruf: Menü **PLC**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Mit dem Befehl führt TwinCAT eine Syntaxprüfung durch und erzeugt den Programmcode. Dieser Code wird auf die Steuerung geladen. Weiterhin erzeugt TwinCAT im Projektverzeichnis das Übersetzungsprotokoll <projectname>.<devicename>.<application ID>.compileinfo.

Beim Download werden alle Variablen mit Ausnahme von persistenten Variablen neu initialisiert.

Die Beschreibung des Befehls **Einloggen** erläutert die möglichen Situationen beim Einloggen und Laden.

Wenn Sie versuchen, ein SPS-Projekt zu laden, während die gleiche Version dieses Projektes bereits auf der Steuerung liegt, erscheint die Meldung: „Programm ist unverändert. Applikation wurde nicht geladen“. TwinCAT lädt das Projekt nicht auf die SPS.

Beim Laden erscheint in der Ansicht **Ausgabe** ein Protokoll der ablaufenden Aktionen (Erzeugen des Codes, Durchführen der Initialisierungen etc.). Weiterhin werden Informationen zu den Speicherbereichen, zur Größe des Codes, der globalen Daten und des allozierten Speichers ausgegeben. Zum Zweck der Übersichtlichkeit werden im Gegensatz zum Online-Change die geänderten Bausteine nicht mehr aufgelistet.

Siehe auch:

- Befehl Einloggen [▶ 1001]

17.8.5 Befehl Online-Change

Funktion: Der Befehl dient dem Anstoßen eines Online-Change auf das gerade aktive SPS-Projekt. Dabei lädt TwinCAT nur die geänderten Teile eines bereits auf der Steuerung laufenden SPS-Projekts neu in die Steuerung.

Aufruf: Menü **PLC**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Ein Online-Change ist nicht möglich nach den Befehlen **Alles bereinigen** und **Bereinigen**. Der Bereinigen-Prozess löscht die Compileinformationen (Übersetzungsprotokoll), die automatisch bei jeder Codeerzeugung gespeichert werden und die die Basis für einen Online-Change sind.

⚠ WARNUNG

Sach- und Personenschäden durch unerwartetes Verhalten der Maschine oder Anlage

Ein Online-Change verändert das laufende Anwendungsprogramm und bewirkt keinen Neustart. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

- Stellen Sie sicher, dass der neue Programmcode das gewünschte Verhalten des gesteuerten Systems bewirkt.

Projektspezifische Initialisierungen

Wenn ein Online-Change durchgeführt wird, werden die projektspezifischen Initialisierungen (Referenzfahrt etc.) nicht ausgeführt, weil die Maschine ihren Status beibehält. Aus diesem Grund hat der neue Programmcode möglicherweise nicht den gewünschten Effekt.

Schwerwiegende Änderungen im Downloadcode

Wenn der Online-Change schwerwiegende Änderungen im Downloadcode bewirkt (z. B. Verschieben von Variablen nötig), informiert ein Dialog über die Effekte und ermöglicht, den Online-Change abzubrechen.

Schneller Online-Change

Für kleine Änderungen (z. B. kleine Änderung im Implementierungsbereich und Verschieben von Variablen nicht nötig) wird ein „schneller Online-Change“ durchgeführt. In diesem Fall wird nur der jeweils geänderte Baustein übersetzt und nachgeladen. Insbesondere wird in dem Fall kein Initialisierungscode erzeugt. Das bedeutet, dass auch kein Code zur Initialisierung von Variablen mit dem Attribut 'init_on_onlchange' erzeugt wird. In der Regel wird das keine Auswirkungen haben, da das Attribut meist dazu verwendet wird, um Variablen mit Adressen zu initialisieren, es kann aber beim schnellen Online-Change nicht dazu kommen, dass eine Variable ihre Adresse ändert.

Um die Wirkung des Attributs init_on_onlchange auf den gesamten Applikationscode sicherzustellen, schalten Sie den schnellen Online-Change mithilfe der Compiler-Definition no_fast_online_change generell für das SPS-Projekt aus. Fügen Sie die Definition zu diesem Zweck in den Eigenschaften des SPS-Projekts in der Kategorie Übersetzen [▶ 951] ein.

Keine Wirkung des Attributs 'init_on_onlchange' bei einzelnen FB-Variablen

Das Attribut 'init_on_onlchange' [▶ 843] wirkt nur bei globalen Variablen, Programmvariablen und lokalen statischen Variablen von Funktionsbausteinen.

Um einen Funktionsbaustein bei einem Online Change neu zu initialisieren muss die Funktionsbausteininstanz mit dem Attribut deklariert werden. Für eine einzelne Variable in einem Funktionsbaustein wird das Attribut nicht ausgewertet.

Zeigervariablen

Zeiger behalten ihren Wert aus dem letzten Zyklus. Wenn ein Zeiger auf eine Variable zeigt, die durch den Online-Change ihre Größe verändert hat und dadurch im Speicher verschoben wurde, liefert die Zeigervariable nicht mehr die korrekte Position der Variablen. Stellen Sie sicher, dass Pointer in jedem Zyklus erneut zugewiesen werden.

Bei einem Online-Change mit möglicherweise unbeabsichtigten Konsequenzen listet TwinCAT in dem Details-Dialog die geänderten Schnittstellen, betroffenen Variablen und alle Bausteine, für die neuer Code generiert wurde. Wenn sich Speicherorte ändern, wird in einem Dialog auf mögliche Probleme in Zusammenhang mit Pointern hingewiesen.

Siehe auch:

- Dokumentation PLC: SPS-Projekt programmieren [▶ 68]

Was verhindert einen Online-Change?

Es gibt Aktionen in TwinCAT, nach denen ein Online-Change auf einer Steuerung nicht mehr möglich ist. Danach ist immer ein Download [▶ 265] des Projekts erforderlich. Ein typischer Fall sind die Aktionen **Bereinigen** und **Alles bereinigen**, die die beim letzten Download abgelegte Übersetzungsinformation löschen. Aber es gibt auch „normale“ Editieraktionen, die dazu führen, dass beim nächsten Einloggen ein Online-Change nicht mehr möglich ist. Folgende Aktionen können einen Online-Change verhindern:

Checkfunktionen	Aktivieren oder Entfernen einer Checkfunktion [▶ 172] (CheckBounds, CheckRange, CheckDiv etc.). Änderung in der Schnittstelle einer Checkfunktion (auch das Einfügen und Löschen von lokalen Variablen).
Taskkonfiguration	Ändern in den Konfigurationseinstellungen.
Projekteinstellungen	Kategorie Übersetzen [▶ 951]: In der Sektion **Einstellungen** (Konstanten ersetzen), Änderung in den Compiler-Defines Kategorie Common [▶ 948]: ID-Änderungen in TwinCAT-Dateien minimieren.
Bausteineigenschaften	Änderung der Option Externe Implementierung
Funktionsbaustein	Ändern des Basisbausteins eines Funktionsbausteins (EXTENDS [▶ 201] FB_Base), auch das Einfügen oder Löschen eines solchen Basisbausteins. Änderung in der Schnittstellenliste (IMPLEMENTS [▶ 208] I_Sample). Ausnahme: Hinzufügen einer neuen Schnittstelle am Ende der Liste.
Datentyp	Ändern des Datentyps einer Variable von einem benutzerdefinierten Datentyp zu einem anderen benutzerdefinierten Datentyp (beispielsweise von TON zu TOF). Ändern des Datentyps von einem benutzerdefinierten Datentyp zu einem Basisdatentyp (beispielsweise von TON zu TIME). Hinweis: Als Workaround ändern Sie gleichzeitig mit dem Datentyp immer auch den Namen der Variablen. Dann wird die Variable als neue Variable initialisiert und die alte entfernt. Ein Online-Change ist danach möglich.

Siehe auch:

- Ausführen eines Online-Change [▶ 263]
- Befehl Einloggen [▶ 1001]

17.8.6 Befehl Einloggen

Symbol: 

Funktion: Der Befehl verbindet das Programmiersystem (das ausgewählte SPS-Projekt) mit dem Zielsystem (Steuerung) und stellt somit den Onlinebetrieb her. Eine Instanz des SPS-Projekts wird auf dem Zielsystem erzeugt und geladen.

Aufruf: Menü **PLC** oder **TwinCAT SPS Symbolleistenoptionen** oder Kontextmenü des SPS-Projektobjekts (<SPS-Projektname>Project) im **Projektmappen-Explorer**

Voraussetzung: Das SPS-Projekt ist fehlerfrei und das Zielsystem befindet sich im Run-Modus.

Mögliche Situationen beim Einloggen:

- Das SPS-Projekt existiert noch nicht auf der Steuerung: Sie werden aufgefordert, den Download zu bestätigen.
- Das SPS-Projekt liegt bereits auf der Steuerung und wurde seit dem letzten Download nicht verändert. Das Einloggen erfolgt ohne weitere Interaktion mit Ihnen.
- Das SPS-Projekt liegt bereits auf der Steuerung, wurde aber seit dem letzten Download verändert. Sie werden aufgefordert, eine der folgenden Optionen zu wählen:
  - Mit Online-Change einloggen (Beachten Sie zum Online-Change die Hinweise im Abschnitt „Befehl Online-Change [▶ 1000]“)
  - Mit Download einloggen
  - Ohne Änderung einloggen

  An dieser Stelle erhalten Sie außerdem die Möglichkeit, das Bootprojekt auf der Steuerung zu aktualisieren.
- Eine unbekannte Version des SPS-Projekts liegt bereits auf der Steuerung. Sie werden gefragt, ob TwinCAT diese ersetzen soll.
- Eine Version des SPS-Projekts liegt bereits auf der Steuerung und läuft. Sie werden gefragt, ob TwinCAT trotzdem einloggen und das gerade laufende SPS-Programm überschreiben soll.
- Das SPS-Programm auf der Steuerung hält gerade an einem Haltepunkt. Sie haben ausgeloggt und das Programm geändert: TwinCAT warnt Sie, dass im Falle eines Online-Change oder Downloads die SPS komplett angehalten wird. Dies geschieht auch, wenn mehrere Tasks vorhanden sind und nur eine von dem Haltepunkt betroffen ist.

Übersetzen des Projekts vor dem Einloggen

Wenn ein SPS-Projekt seit seiner letzten Änderung noch nicht kompiliert wurde, übersetzt TwinCAT das Projekt vor dem Einloggen. Dieser Vorgang entspricht dem Befehl **Übersetzen im ausgeloggten Zustand**.

Wenn während des Übersetzens Fehler auftreten, erscheint ein Meldungsdialog. Die Fehler werden in der Ansicht **Fehlerliste** ausgegeben. Sie können dann entscheiden, ob Sie einloggen wollen, ohne das Programm auf die Steuerung zu laden.

Siehe auch:

- Befehl SPS-Projekt erstellen [▶ 973]

Fehler beim Login

Wenn während des Einloggens auf die Steuerung ein Fehler auftritt, bricht TwinCAT den Ladevorgang mit einer Fehlermeldung ab. Der Fehlerdialog bietet Ihnen die Möglichkeit, die Fehlerdetails anzuzeigen. Ist ein Ausnahmefehler aufgetreten und der Text *SOURCEPOSITION* in der Log-Meldung enthalten, können Sie mit dem Befehl Im Editor anzeigen die betroffene Funktion im Editor anzeigen. Dabei springt der Cursor an die fehlerverursachende Zeile.

Ausgabe von Informationen zum Ladevorgang

Wenn TwinCAT das Projekt beim Einloggen auf die Steuerung lädt, werden folgende Informationen im Meldungsfenster ausgegeben:

- Generierte Codegröße
- Größe der globalen Daten
- Resultierender Speicherbedarf auf der Steuerung
- Eine Liste der betroffenen Bausteine (bei Online-Change)

Im Onlinebetrieb können Sie die Einstellungen von Geräten oder Modulen nicht verändern. Um Geräteparameter zu ändern, müssen Sie das SPS-Projekt ausloggen. Abhängig vom Bussystem kann es jedoch einige spezielle Parameter geben, die Sie auch im Onlinebetrieb verändern können.

Die Konfiguration der Ansicht speichert TwinCAT im Online- und Offlinebetrieb separat voneinander. Zusätzlich werden Ansichten, welche in einer Betriebsart nicht nutzbar sind, geschlossen. Aus diesem Grund kann sich die Ansicht beim Einloggen automatisch ändern.

17.8.7 Befehl Start

Symbol: 

Tastaturkürzel: **[F5]**

Funktion: Der Befehl startet die Ausführung des Programms.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Wenn Sie den Befehl aus dem Menü **PLC** aufrufen, wirkt er auf das gerade aktive SPS-Projekt.

17.8.8 Befehl Stop

Symbol: 

Tastaturkürzel: **[Umschalt] + [F5]**

Funktion: Der Befehl stoppt die Ausführung des Programms.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Wenn Sie den Befehl aus dem Menü **PLC** aufrufen, wirkt er auf das gerade aktive SPS-Projekt.

17.8.9 Befehl Ausloggen

Symbol: 

Funktion: Der Befehl beendet die Verbindung zwischen Entwicklungssystem und Zielsystem (Steuerung oder simuliertes Gerät), wodurch in den Offlinebetrieb gewechselt wird.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

17.8.10 Befehl Reset kalt

Symbol: 

Funktion: Der Befehl setzt alle Variablen des aktiven SPS-Projekts, mit Ausnahme der PERSISTENT-Variablen und der RETAIN-Variablen, auf ihren Initialisierungswert zurück.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Alle Variablen des aktiven SPS-Projekts, mit Ausnahme der PERSISTENT-Variablen und der RETAIN-Variablen, werden zurückgesetzt. Die Situation entspricht der beim Start eines Anwendungsprogramms, das gerade auf die Steuerung geladen wurde („Kaltstart“).

Haltepunkte des SPS-Programms, die vor dem Zurücksetzen des aktiven SPS-Projekts aktiviert waren, sind nach der Ausführung des Befehls weiterhin aktiviert. Haltepunkte, die vorher deaktiviert waren, sind nach der Ausführung des Befehls weiterhin deaktiviert.

Wenn Sie den Befehl auswählen, während der Programmablauf gerade auf einem Haltepunkt stoppt, werden Sie gefragt, ob der aktuelle Zyklus noch beendet werden soll. Alternativ führt TwinCAT den Reset sofort aus. Allerdings sind nicht alle Laufzeitsysteme in der Lage einen Reset ohne vorheriges Beenden des aktuellen Zyklus durchzuführen.

Nach dem Reset müssen Sie das SPS-Programm mit dem Befehl **Start** starten.

Siehe auch:

- Dokumentation PLC: Remanente Variablen - PERSISTENT, RETAIN [▶ 726]
- Dokumentation PLC: Reset des SPS-Projekts durchführen [▶ 230]
- Befehl Reset Ursprung [▶ 1004]
- Befehl Start [▶ 1003]

17.8.11 Befehl Reset Ursprung

Symbol: 

Funktion: Der Befehl setzt alle Variablen des aktiven SPS-Projekts, einschließlich der remanenten Variablen (RETAIN-, PERSISTENT-Variablen), auf ihre Initialisierungswerte zurück und löscht das Anwendungsprogramm auf der Steuerung.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Ein Reset deaktiviert die aktuell im Programm gesetzten Haltepunkte. Wenn Sie den Befehl auswählen, während der Programmablauf gerade auf einem Haltepunkt stoppt, werden Sie gefragt, ob der aktuelle Zyklus noch beendet werden soll. Alternativ führt TwinCAT den Reset sofort aus. Allerdings sind nicht alle Laufzeitsysteme in der Lage einen Reset ohne vorheriges Beenden des aktuellen Zyklus durchzuführen.

Siehe auch:

- Dokumentation PLC: Remanente Variablen - PERSISTENT, RETAIN [▶ 726]
- Dokumentation PLC: Reset des SPS-Projekts durchführen [▶ 230]
- Befehl Reset kalt [▶ 1003]

17.8.12 Befehl Einzelzyklus

Symbol: 

Funktion: Der Befehl führt das aktive SPS-Programm für einen Zyklus aus.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und das Programm steht an einem Programmschritt.

17.8.13 Befehl Ablaufkontrolle

Symbol: 

Funktion: Der Befehl aktiviert oder deaktiviert die Ablaufkontrolle.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

Eine aktive Ablaufkontrolle verlängert die Laufzeit des SPS-Projekts!

Siehe auch:

- Dokumentation PLC: Ablaufkontrolle [▶ 231]

17.8.14 Befehl Werte forcen

Symbol:

Funktion: Der Befehl setzt den Wert einer Variablen auf der Steuerung dauerhaft auf einen vordefinierten Wert.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb.

⚠ VORSICHT

Sach- und Personenschäden durch unerwartetes Verhalten der Maschine oder Anlage

Das außerordentliche Ändern von Variablenwerten in einem auf der Steuerung laufenden SPS-Programm kann zu einem unerwarteten Verhalten der gesteuerten Maschine führen. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

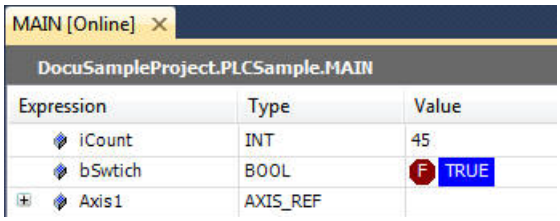
- Evaluieren Sie vor einem Forcen von Variablenwerten mögliche Gefahren und treffen Sie entsprechende Sicherheitsvorkehrungen.

Mit dem Befehl setzt TwinCAT eine oder mehrere Variablen der aktiven Applikation auf der Steuerung permanent auf definierte Werte. Dieses Setzen erfolgt jeweils am Beginn und am Ende eines Abarbeitungszyklus. Abfolge der Abarbeitung: 1. Eingänge lesen, 2. Werte forcen 3. Code abarbeiten, 4. Werte forcen, 5. Ausgänge schreiben.

Sie können Werte vorbereiten durch

- Klicken in das Feld **vorbereiteter Wert** im Deklarationsteil und geben Sie den neuen Wert ein. Bei einer booleschen Variablen ändern Sie den Wert durch einfaches Klicken in das Feld.
- Klicken in das Inline-Monitoring-Feld im Implementierungsteil des FUP/KOP/AWL-Editors und geben Sie den neuen Wert ein.
- Klicken in das Feld **vorbereiteter Wert** im Überwachen-Fenster und geben Sie den neuen Wert ein.

Ein „geforcter“ Wert wird mit einem F Symbol gekennzeichnet.



TwinCAT führt das Forcen durch, bis es explizit durch den Anwender aufgehoben wird durch

- den Befehl **Forcen aufheben**
- das Aufheben des Forces über den Dialog **Wert vorbereiten**
- Ausloggen aus der Applikation

Der Befehl **Werte forcen für alle Applikationen**, der alle SPS-Projekte im TwinCAT-Projekt betrifft, ist standardmäßig nicht in einem Menü enthalten.

Siehe auch:

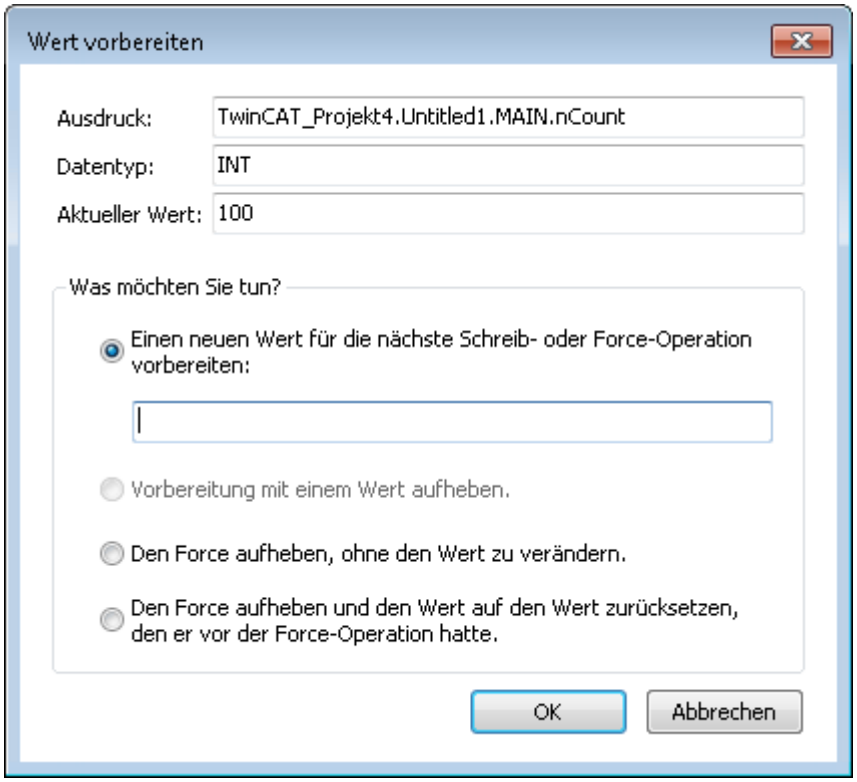
- Befehl Forcen aufheben [▶ 1006]
- Dialog Wert vorbereiten [▶ 1006]
- Dokumentation PLC: Forcen und Schreiben von Variablen [▶ 225]

17.8.14.1 Dialog Wert vorbereiten

Funktion: Der Dialog dient zur Vorbereitung eines Werts für eine bereits geforcte Variable. TwinCAT führt die vorbereitete Aktion mit dem nächsten Forcen aus.

Aufruf: TwinCAT öffnet den Dialog in folgenden Situationen:

- wenn Sie im Deklarationsteil in das Feld **vorbereiteter Wert** einer geforcten Variable klicken
- wenn Sie im Implementierungsteil in das Inline-Monitoring-Feld einer geforcten Variablen klicken
- wenn Sie im Überwachen-Fenster in das Feld **vorbereiteter Wert** einer geforcten Variable klicken



Einen neuen Wert für die nächste Schreib- oder Force-Operation vorbereiten	Wert, den TwinCAT bei der nächsten Force-Operation auf die Variable schreibt
Vorbereitung mit einem Wert aufheben	TwinCAT löscht den vorbereiteten Wert.
Den Force aufheben, ohne den Wert zu verändern.	TwinCAT behält den geforcten Wert bei und beendet das Forcen. TwinCAT markiert die Variable mit <Unforce>.
Den Force aufheben und den Wert auf den Wert zurücksetzen, den er vor der Force-Operation hatte.	TwinCAT setzt den geforcten Wert zurück und beendet das Forcen. Dabei wird die Variable mit <Unforce and restore> markiert.

Siehe auch:

- Befehl Werte forcen [▶ 1005]

17.8.15 Befehl Forcen aufheben

Symbol: 

Funktion: Der Befehl setzt das Forcen aller Variablen zurück. Die Variablen erhalten dabei ihren aktuellen Wert aus der Steuerung.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt befindet sich im Onlinebetrieb.

⚠ VORSICHT

Sach- und Personenschäden durch unerwartetes Verhalten der Maschine oder Anlage

Das außerordentliche Ändern von Variablenwerten in einem auf der Steuerung laufenden SPS-Programm kann zu einem unerwarteten Verhalten der gesteuerten Maschine führen. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

- Evaluieren Sie vor einem Zurücksetzen der geforcten Variablenwerte mögliche Gefahren und treffen Sie entsprechende Sicherheitsvorkehrungen.

Der Befehl **Forcen aufheben für alle Applikationen**, der alle SPS-Projekte im TwinCAT-Projekt betrifft, ist standardmäßig nicht in einem Menü enthalten.

Siehe auch:

- Befehl Werte forcen [▶ 1005]
- Dokumentation PLC: Forcen und Schreiben von Variablen [▶ 225]

17.8.16 Befehl Werte schreiben

Symbol: 

Funktion: Der Befehl setzt den Wert einer Variablen auf der Steuerung einmalig auf einen vordefinierten Wert.

Aufruf: Menü **PLC**, **TwinCAT SPS Symbolleistenoptionen**

Voraussetzung: Das SPS-Projekt befindet sich im Onlinebetrieb.

⚠ VORSICHT

Sach- und Personenschäden durch unerwartetes Verhalten der Maschine oder Anlage

Das außerordentliche Ändern von Variablenwerten in einem auf der Steuerung laufenden SPS-Programm kann zu einem unerwarteten Verhalten der gesteuerten Maschine führen. Abhängig von der gesteuerten Maschine können Schäden an Maschine und Werkstücken entstehen oder Gesundheit und Leben von Personen gefährdet werden.

- Evaluieren Sie vor einem Schreiben von Variablenwerten mögliche Gefahren und treffen Sie entsprechende Sicherheitsvorkehrungen.

Mit dem Befehl setzen Sie eine oder mehrere Variablen des aktiven SPS-Projekts auf der Steuerung einmalig auf definierte Werte. Das Schreiben erfolgt einmal zu Beginn des nächsten Zyklus.

Sie können Werte vorbereiten durch

- Klicken in das Feld **vorbereiteter Wert** im Deklarationsteil und geben Sie den neuen Wert ein. Bei einer booleschen Variablen ändern Sie den Wert durch einfaches Klicken in das Feld.
- Klicken in das Inline-Monitoring-Feld im Implementierungsteil des FUP/KOP/AWL-Editors und geben Sie den neuen Wert ein.
- Klicken in das Feld **vorbereiteter Wert** im Überwachen-Fenster und geben Sie den neuen Wert ein.

Der Befehl **Werte schreiben für alle Applikationen**, der alle SPS-Projekte im TwinCAT-Projekt betrifft, ist standardmäßig nicht in einem Menü enthalten.

Siehe auch:

- Befehl Werte forcen [▶ 1005]
- Dokumentation PLC: Forcen und Schreiben von Variablen [▶ 225]

17.8.17 Befehl Darstellung - Binär, Dezimal, Hexadezimal

Funktion: Die Befehle des Untermenüs **Darstellung** dienen der Einstellung des Formats für die Darstellung der Werte beim Monitoring im Onlinebetrieb.

Aufruf: Menü **PLC**, Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Offline- oder Onlinebetrieb.

Die Darstellungsformate „Binär“ und „Hexadezimal“ sind vorzeichenlos, „Dezimal“ ist vorzeichenbehaftet.

Siehe auch:

- Dokumentation PLC: Monitoring von Werten [▶ 234]
- Dokumentation PLC: Deklarationseditor [▶ 654]

17.8.18 Befehl Darstellung Vererbung – Einfach, Strukturiert

Verfügbar ab TC3.1 Build 4026

Funktion: Die Befehle des Untermenüs **Darstellung Vererbung** dienen der Einstellung des Formats für die Darstellung der Vererbungshierarchie von Funktionsbausteinen und Strukturen beim Monitoring im Onlinebetrieb.

Aufruf: Menü **PLC**, **Kontextmenü**

Voraussetzung: Das SPS-Projekt ist im Offline- oder Onlinebetrieb.

Strukturiert	Die Vererbungshierarchie von Funktionsbausteinen und Strukturen wird in einer Baumstruktur dargestellt. Die Variablen werden dabei als Kindknoten des Funktionsbausteins oder der Struktur dargestellt, in dem sie deklariert sind.
Einfach	Die Darstellung erfolgt als flache Liste.

Siehe auch:

- Dokumentation PLC: Monitoring von Werten [▶ 234]
- Dokumentation PLC: Deklarationseditor [▶ 654]

17.8.19 Befehl Lokalisierungsvorlage erzeugen

Funktion: Der Befehl öffnet den Dialog **Lokalisierungsvorlage erzeugen**. Hier definieren Sie, welche Textinformationen aus dem Projekt in eine Übersetzungsvorlage des Dateiformats pot exportiert werden sollen.

Aufruf: Menü **PLC > Project Localization**

Voraussetzung: Ein Projekt ist geöffnet.

Dialog Lokalisierungsvorlage erzeugen

Der Dialog dient der Auswahl der textuellen Informationen, die in die Lokalisierungsvorlage aufgenommen werden sollen.

Folgende Informationen einschließen

Namen	Texte wie beispielsweise Dialogtitel, Objektnamen im SPS-Projektbaum
Bezeichner	Variablenbezeichner, Bespiel: „nCounter“
Zeichenfolgen	Beispielsweise „count“ in der folgenden Deklaration: sVar : STRING := 'count'
Kommentare	Kommentartexte in den Programmierbausteinen

Positionsinformationen	Auswahl, welche Positionen der oben ausgewählten Textkategorien im Projekt in die Übersetzungsdatei aufgenommen werden sollen. Die Positionsinformation steht jeweils in der/den ersten Zeile(n) eines Abschnitts für eine Übersetzung.
	Beispiel:
	`#: D:\Proj1.project\Project_Settings:1`
	`msgid „Projekteinstellungen“`
	`msgstr „„`
	• „Alle“: Alle gefundenen Positionen des Texts werden aufgeführt.
	• „Erstes Auftreten“: In die Übersetzungsdatei wird die Position im Projekt aufgenommen, an der der zu übersetzende Text erstmalig auftritt.
	• „Keine“

Erzeugen	Die Schaltfläche öffnet den Dialog zum Speichern einer Datei. Die Übersetzungsvorlage wird in einer Textdatei vom Typ POT Translation Template (*.pot) angelegt. Jedes weitere Erzeugen erstellt wieder eine komplette neue Vorlagedatei.

17.8.20 Befehl Lokalisierungen verwalten

Funktion: Der Befehl öffnet den Dialog **Lokalisierung verwalten**. Im Dialog wählen Sie die gewünschte Lokalisierungssprache oder die Originalversion des Projekts aus. Weiterhin können Sie hier Lokalisierungsdateien *.<Sprache>.po ins Projekt aufnehmen oder daraus entfernen.

Aufruf: Menü **PLC > Project Localization**

Voraussetzung: Ein Projekt ist geöffnet.

Dialog Lokalisierung verwalten

Verfügbare Lokalisierungen	Liste der im Projekt vorliegenden Lokalisierungsdateien.
	Beispiel:
	`proj1-de.po`
	`proj1-en.po`
	`<Originalversion>`
	Die Originalversion ist immer verfügbar. Nur in der Originalversion kann das Projekt editiert werden.

Hinzufügen	Die Schaltfläche öffnet den Dialog zum Auswählen einer weiteren po-Datei aus dem Dateisystem.
Entfernen	Die Schaltfläche entfernt die links ausgewählte po-Datei aus dem Projekt.
Standardlokalisierung	 Die gerade selektierte Lokalisierung wird zur Standardlokalisierung. Der Eintrag wird fett dargestellt.
Lokalisierung wechseln	Mit der Schaltfläche wechseln Sie zur gerade selektierten Lokalisierung.
OK	Das Projekt wird in der Landessprache dargestellt, die durch die unter Dateien ausgewählte Datei geliefert wird. Wenn Sie **<Originalversion>** auswählen, erscheint das Projekt in der editierbaren, nicht lokalisierten Fassung.

17.8.21 Befehl Lokalisierung umschalten

Symbol: 

Funktion: Der Befehl schaltet zwischen der aktuell eingestellten Projektlokalisierung und der <Originalversion> um.

Aufruf:

- Menü **Project > Localization**
- Schaltfläche im Dialog **Lokalisierungen verwalten**
- Schaltfläche in der Werkzeugleiste

Voraussetzung: Ein Projekt ist geöffnet. Eine Standardlokalisierung für das Projekt ist im Dialog **Lokalisierungen verwalten** definiert.

Siehe auch:

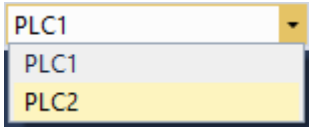
- Befehl Lokalisierungen verwalten [▶ 1009]

17.8.22 Befehl Active PLC Project

Funktion: Drop-down-Liste zur Auswahl des aktiven SPS-Projekts. Das aktuell fokussierte Projekt wird automatisch als aktives SPS-Projekt gesetzt.

Aufruf: TwinCAT SPS Symbolleistenoptionen

Voraussetzung: Das TwinCAT Projekt enthält mehrere SPS-Projekte.



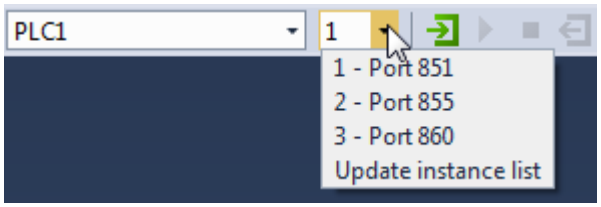
Siehe auch:

- Befehl Active PLC Instance [▶ 1010]
- Befehl Einloggen [▶ 1001]

17.8.23 Befehl Active PLC Instance

Funktion: Drop-down-Liste zur Auswahl der aktiven SPS-Instanz des zugehörigen SPS-Projekts.

Aufruf: TwinCAT SPS Symbolleistenoptionen



Siehe auch:

- Befehl Active PLC Project [▶ 1010]
- Befehl Einloggen [▶ 1001]

17.9 Extras

17.9.1 Befehl Optionen

Funktion: Der Befehl öffnet den Dialog **Optionen** zur Konfiguration der TwinCAT-Optionen. Diese Optionen definieren das Verhalten und Aussehen der TwinCAT-Benutzeroberfläche. TwinCAT speichert die aktuellen Einstellungen auf dem lokalen System als Standardeinstellungen.

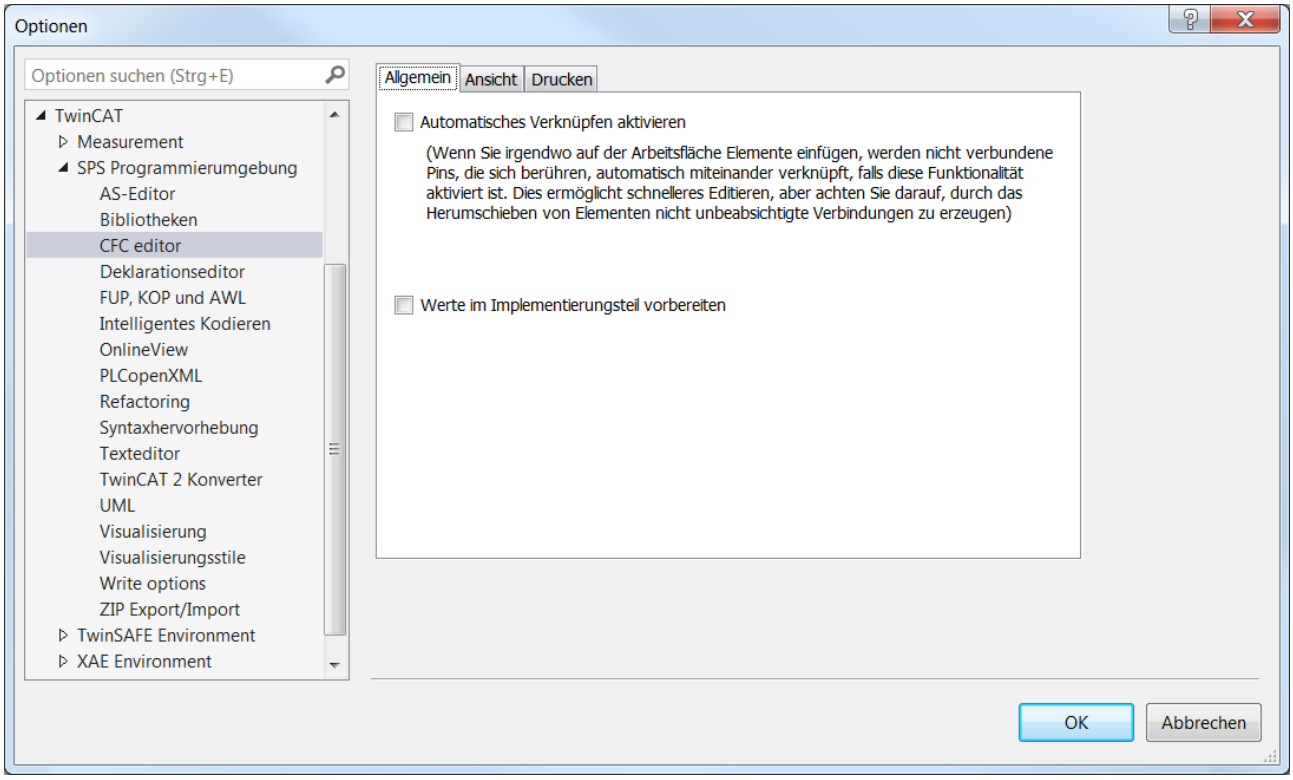
Aufruf: Menü **Extras > Optionen > TwinCAT > SPS Programmierumgebung**

17.9.1.1 Dialog Optionen - CFC-Editor

Funktion: Der Dialog dient der Konfiguration der Einstellungen für das Editieren und Drucken im CFC-Editor.

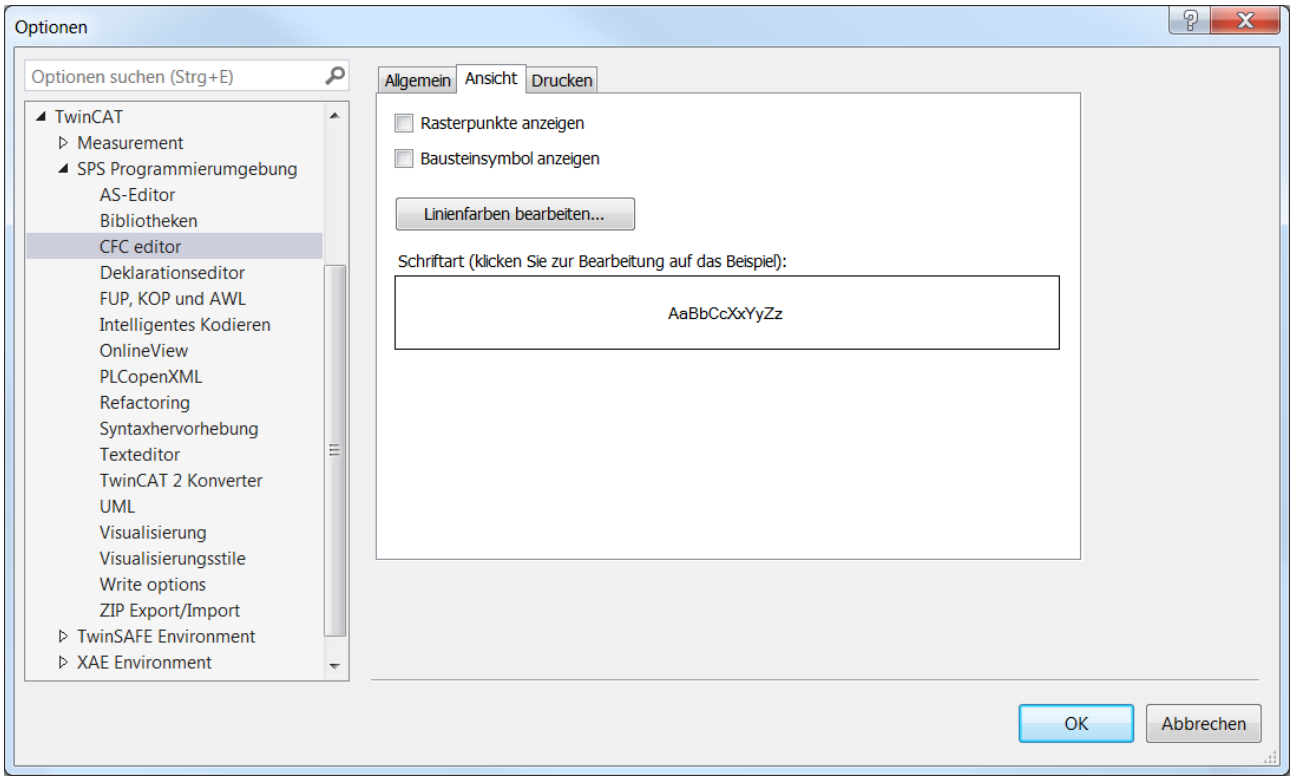
Aufruf: **TwinCAT > SPS Programmierumgebung > CFC-Editor**

Registerkarte Allgemein



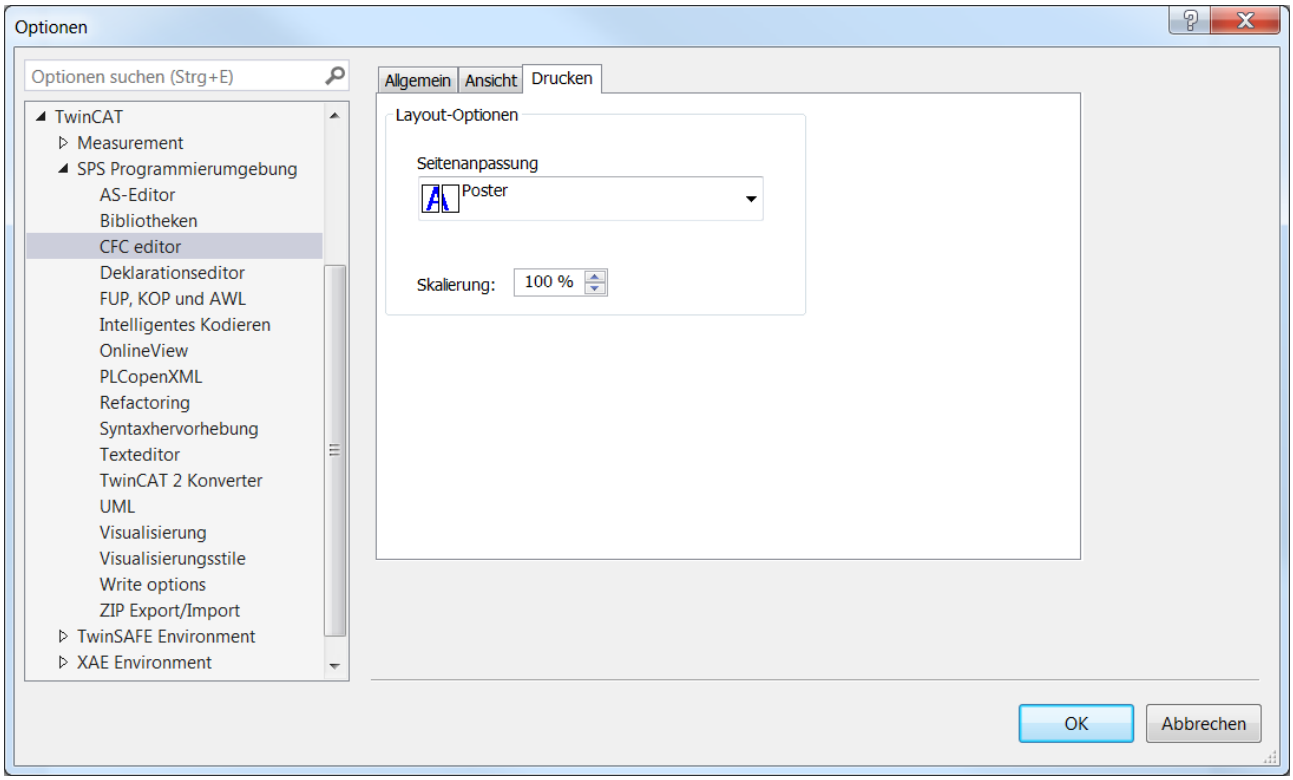
Automatisches Verknüpfen aktivieren	☑ : Wenn Sie ein CFC-Element auf die Arbeitsfläche des Editors ziehen und einfügen, verbindet TwinCAT automatisch unverknüpfte Pins, die sich „berühren“. Achten Sie darauf, dass Sie beim Verschieben von Elementen keine unerwünschten Verknüpfungen erzeugen!
Werte im Implementierungsteil vorbereiten	☑ : Im Onlinebetrieb können Sie auch im Implementierungsteil des CFC-Bausteins Variablenwerte für das Schreiben und Forcen vorbereiten. Außerdem zeigt TwinCAT die gerade vorbereiteten Werte in der Inline-Monitoring-Box der Variablen in spitzen Klammern an.

Registerkarte Ansicht



Rasterpunkte anzeigen	☑ : Im Editor sind Rasterpunkte Gültigkeitsbereich, an denen Sie die Elemente positionieren können.
Bausteinsymbol anzeigen	☑ : TwinCAT stellt im CFC-Editor vorhandene Funktionsbausteine, die mit einem Bitmap verknüpft sind, als Symbol dar. Voraussetzung: Sie haben für einen Funktionsbaustein oder eine Funktion die Verknüpfung entweder in den Objekteigenschaften erstellt oder über eine Bibliothek geladen.
Linienfarben bearbeiten	Öffnet den Dialog **Linienfarben bearbeiten** zur Definition der Farben für die Verbindungslinien in Abhängigkeit vom anliegenden Datentyp. Die Linien erscheinen im Offline- und Onlinebetrieb in diesen Farben, außer TwinCAT übermalt diese Farben durch die fette schwarze und blaue Liniendarstellung, die booleschen Datenfluss anzeigt. • Typ hinzufügen: Fügt der Liste eine Datentyp hinzu. • Typ entfernen
Schriftart	Anzeige der Schriftart und Schaltfläche zur Änderung der Schriftart.

Registerkarte Drucken



Layout-Optionen

Seitenanpassung	Seite oder Poster
Skalierung	Mögliche Werte: 20 % - 200 %

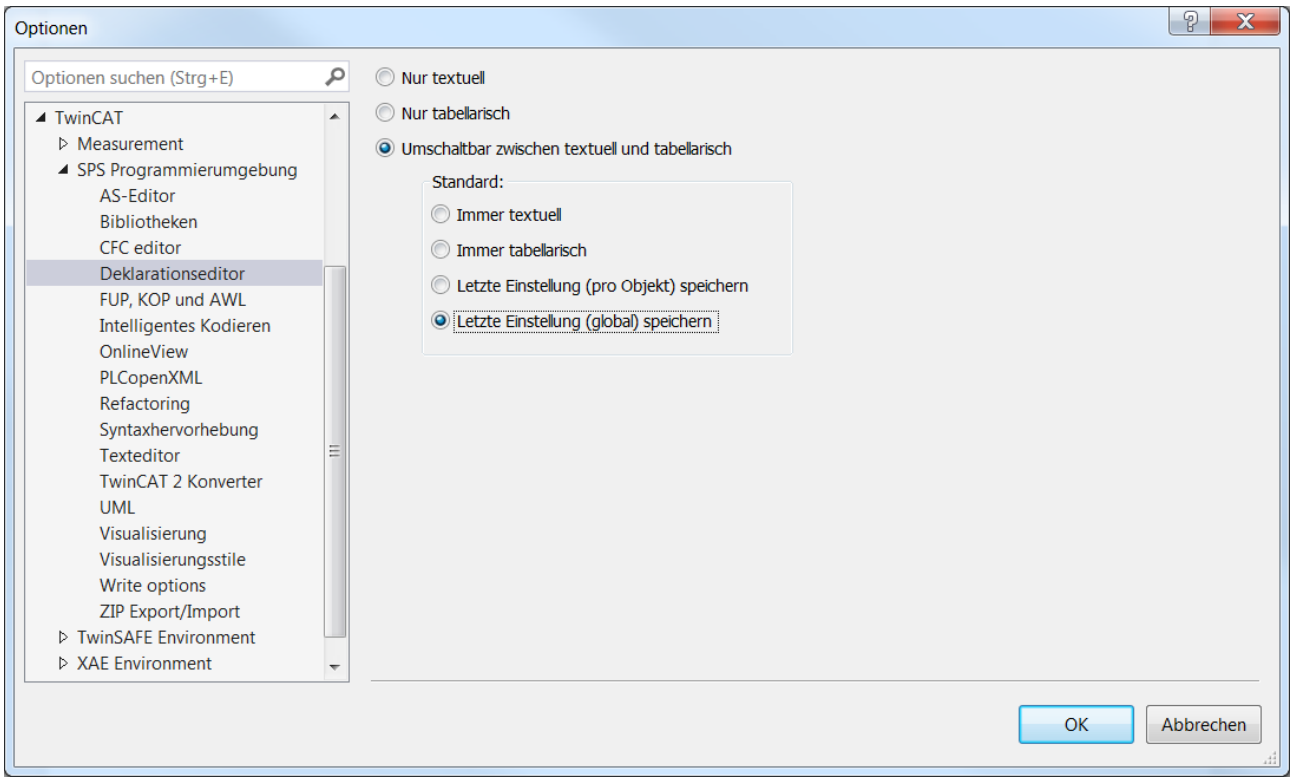
Siehe auch:

- Dokumentation PLC: Programmieren in CFC [▶ 119]
- Dokumentation PLC: Programmiersprachen und ihre Editoren [▶ 654]

17.9.1.2 Dialog Optionen - Deklarationseditor

Funktion: Der Dialog dient der Konfiguration der Anzeigeeinstellungen für den Deklarationseditor.

Aufruf: TwinCAT > SPS Programmierumgebung > Deklarationseditor



Nur textuell	Textuelle Ansicht des Deklarationseditors
Nur tabellarisch	Tabellarische Ansicht des Deklarationseditors
Umschaltbar zwischen textuell und tabellarisch	Der Deklarationseditor bietet zwei Schaltflächen, um zwischen der textuellen und der tabellarischen Ansicht zu wechseln: : textuelle Ansicht : tabellarische Ansicht Die folgende Option definiert die Ansicht, die standardmäßig beim Öffnen eines Programmierobjekts erscheint: • Immer textuell • Immer tabellarisch • Letzte Einstellung (pro Objekt) speichern • Letzte Einstellung (global) speichern

Siehe auch:

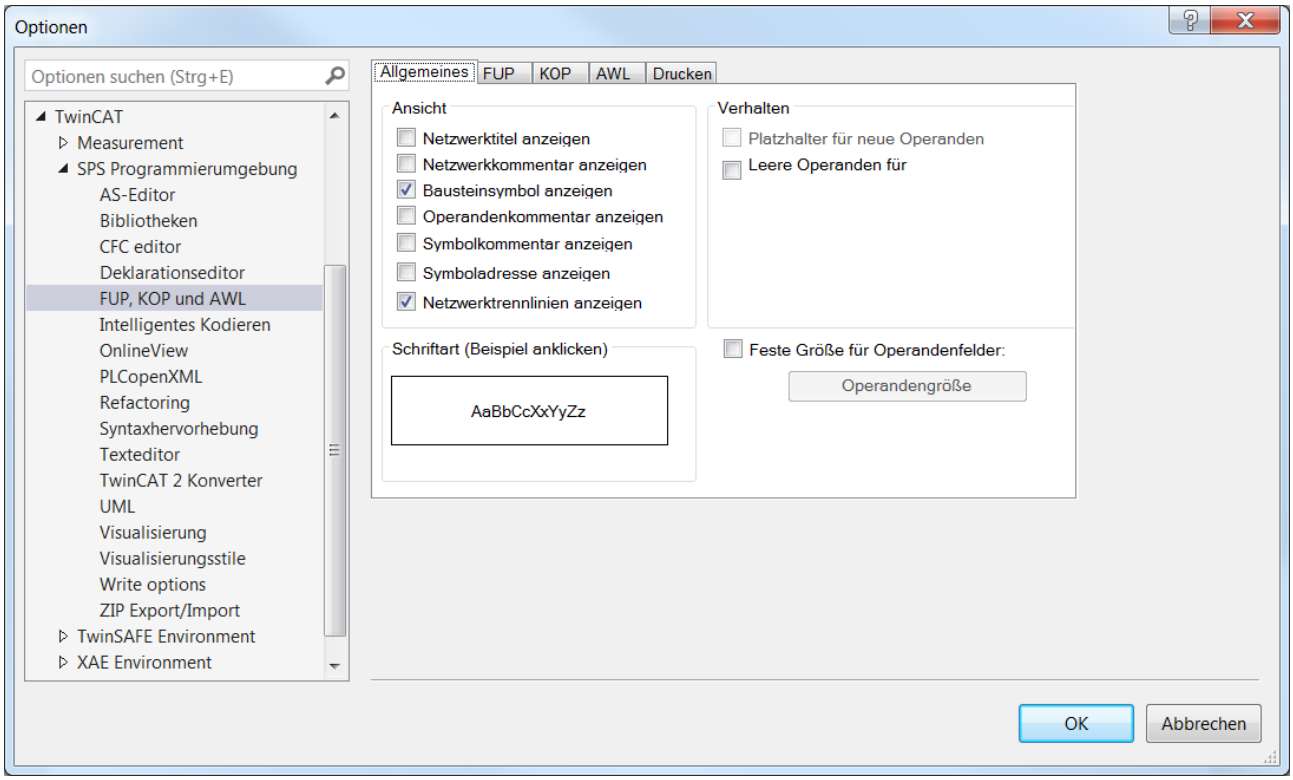
- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]

17.9.1.3 Dialog Optionen - FUP, KOP und AWL

Funktion: Der Dialog dient der Konfiguration der Darstellungsoptionen für den FUP/KOP/AWL-Editor.

Aufruf: **TwinCAT > SPS Programmierumgebung > FUP, KOP und AWL**

Registerkarte Allgemeines



Ansicht

Netzwerktitel anzeigen	Netzwerktitel wird in der oberen linken Ecke des Netzwerks angezeigt.
Netzwerkkommentar anzeigen	Netzwerkkommentar wird in der oberen linken Ecke des Netzwerks angezeigt. Wenn TwinCAT zusätzlich den Netzwerktitel darstellt, erscheint der Kommentar in der Zeile unterhalb.
Bausteinsymbol anzeigen	Bausteinsymbol wird im Bausteinelement im FUP- und KOP-Editor angezeigt. Auch die Standardoperatoren haben Symbole.
Operandenkommentar anzeigen	TwinCAT zeigt den Kommentar an, den Sie einer Variablen im Implementierungsteil gegeben haben. Der Operandenkommentar bezieht sich nur auf die lokale Verwendungsstelle der Variablen, im Gegensatz zum „Symbolkommentar“.
Symbolkommentar anzeigen	TwinCAT zeigt den Kommentar, den Sie einer Variablen oder einem Symbol bei der Deklaration gegeben haben, oberhalb des Variablennamens an. Zusätzlich oder anstelle des Symbolkommentars können Sie auch einen lokalen „Operandenkommentar“ zuweisen.
Symboladresse anzeigen	Wenn einem Symbol (Variable) eine Adresse zugewiesen ist, wird diese Adresse oberhalb des Variablennamens angezeigt.
Netzwerktrennlinien anzeigen	Zwischen den einzelnen Netzwerken wird eine Trennlinie angezeigt.

Verhalten

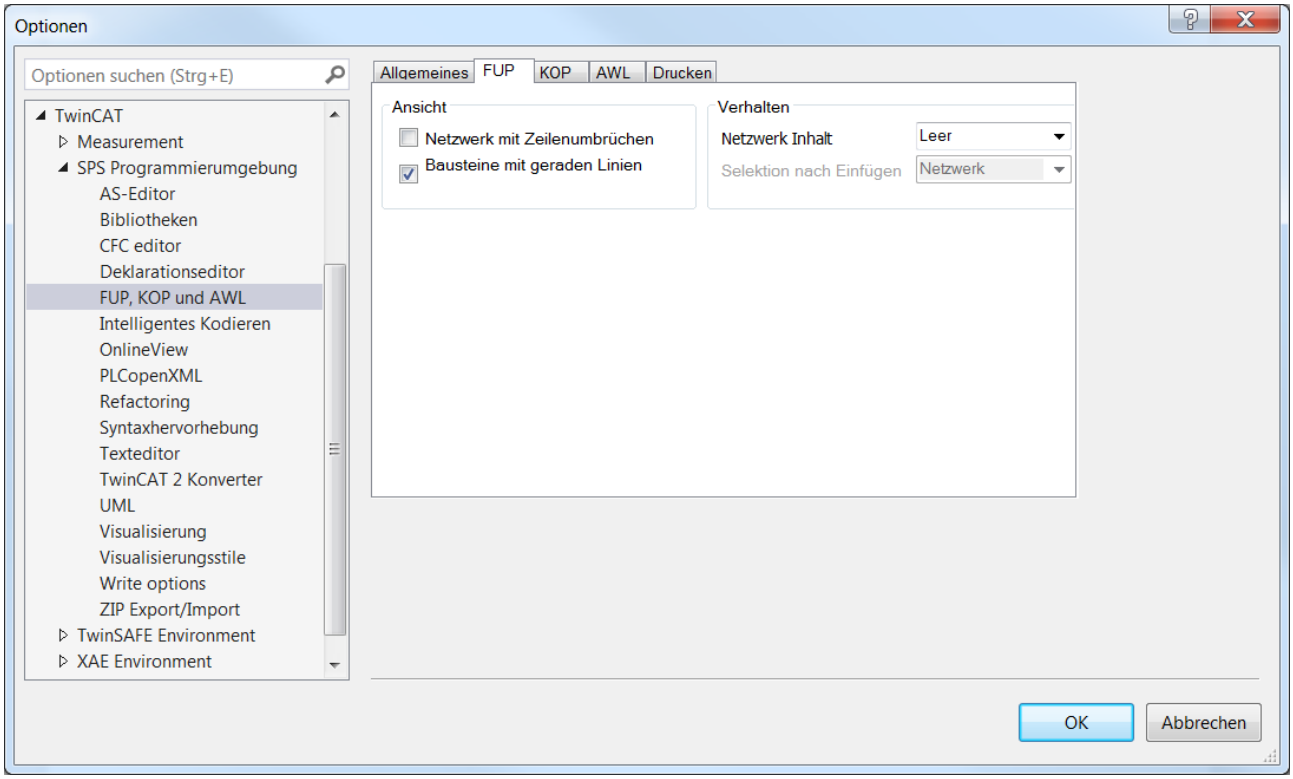
Platzhalter für neue Operanden	Operandenfeld der Pins der neuen Funktionsbausteine wird leer gelassen (anstelle von „???“).
Leere Operanden für Funktionsbaustein-Pins	Fügt anstelle von ??? leere Operanden ein.

Schriftart

Klick auf das Eingabefeld öffnet den Dialog **Schriftart**.

Feste Größe für Operandenfelder	☑ : Operandengröße bearbeiten ist aktivierbar.
Operandengröße bearbeiten	Dialog **Größe Operanden** erscheint zum Einstellen der Anzahl Zeichen und Zeilen.

Registerkarte FUP



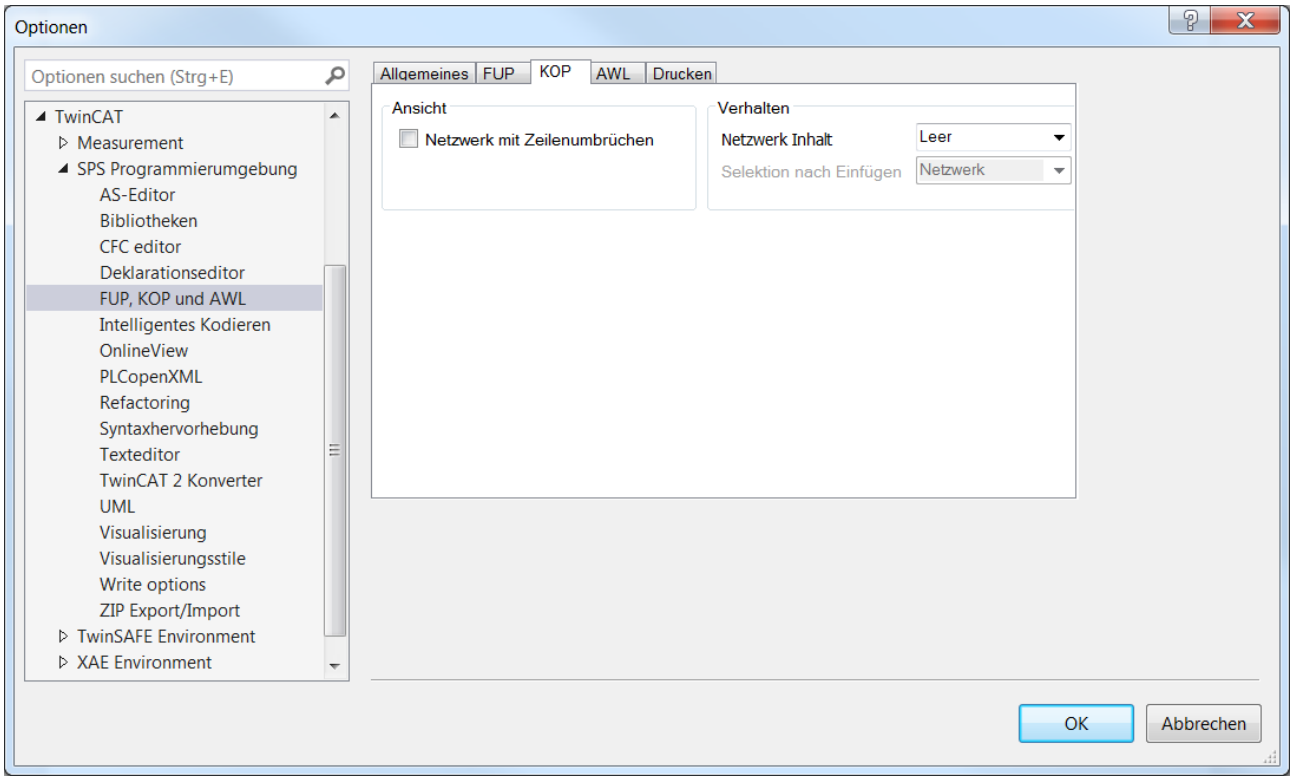
Ansicht

Netzwerk mit Zeilenumbrüchen	☑ : Darstellung der Netzwerke mit Zeilenumbrüchen, so dass TwinCAT möglichst viele Bausteine in der aktuellen Breite des Fensters darstellen kann.
Bausteine mit geraden Linien verbinden	☑ : Linien zwischen den Elementen erhalten eine feste, kurze Länge.

Verhalten

Netzwerk Inhalt	Auswahlliste: Inhalt eines neuen Netzwerks.
Selektion nach Einfügen	Auswahlliste: Element, das TwinCAT nach dem Einfügen eines neuen Netzwerks selektiert.

Registerkarte KOP



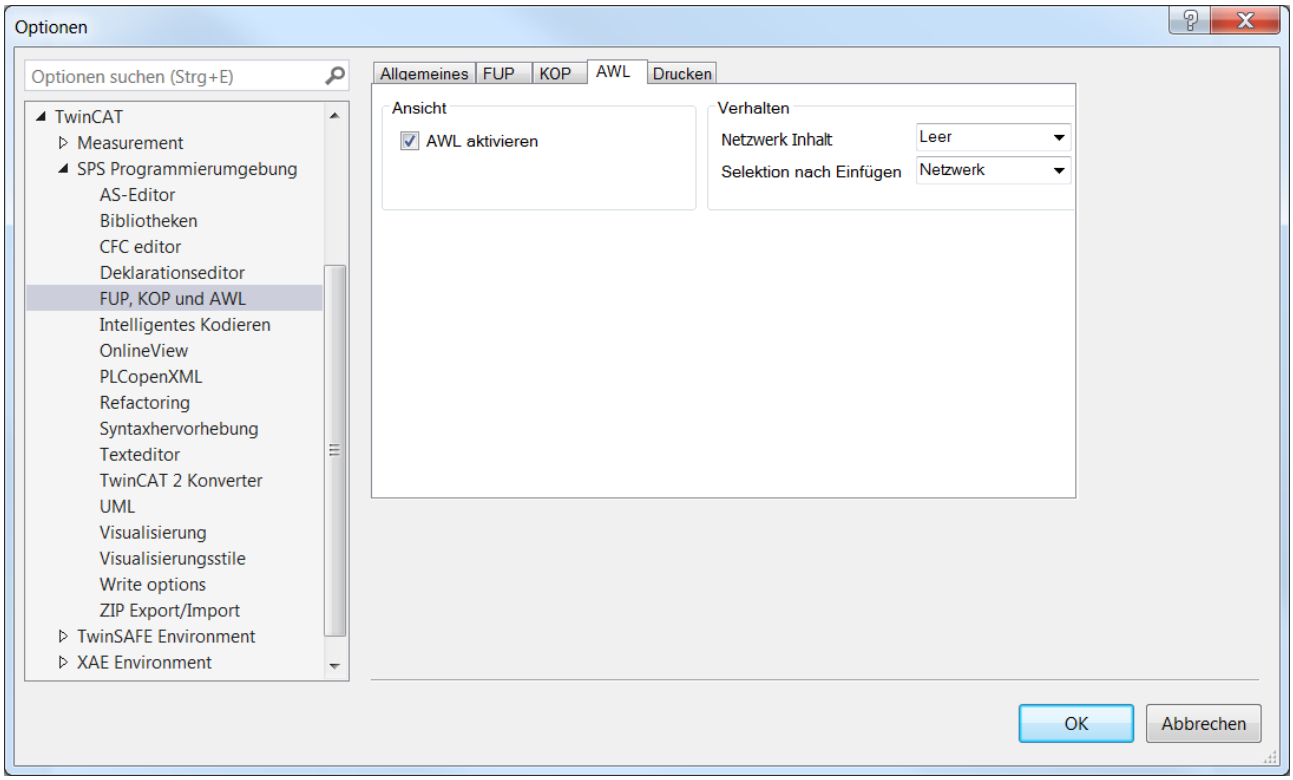
Ansicht

Netzwerk mit Zeilenumbrüchen	☑ : Darstellung der Netzwerke mit Zeilenumbrüchen, so dass TwinCAT möglichst viele Bausteine in der aktuellen Breite des Fensters darstellen kann.

Verhalten

Netzwerk Inhalt	Auswahlliste: Inhalt eines neuen Netzwerks.
Selektion nach Einfügen	Auswahlliste: Element, das TwinCAT nach dem Einfügen eines neuen Netzwerks selektiert.

Registerkarte AWL



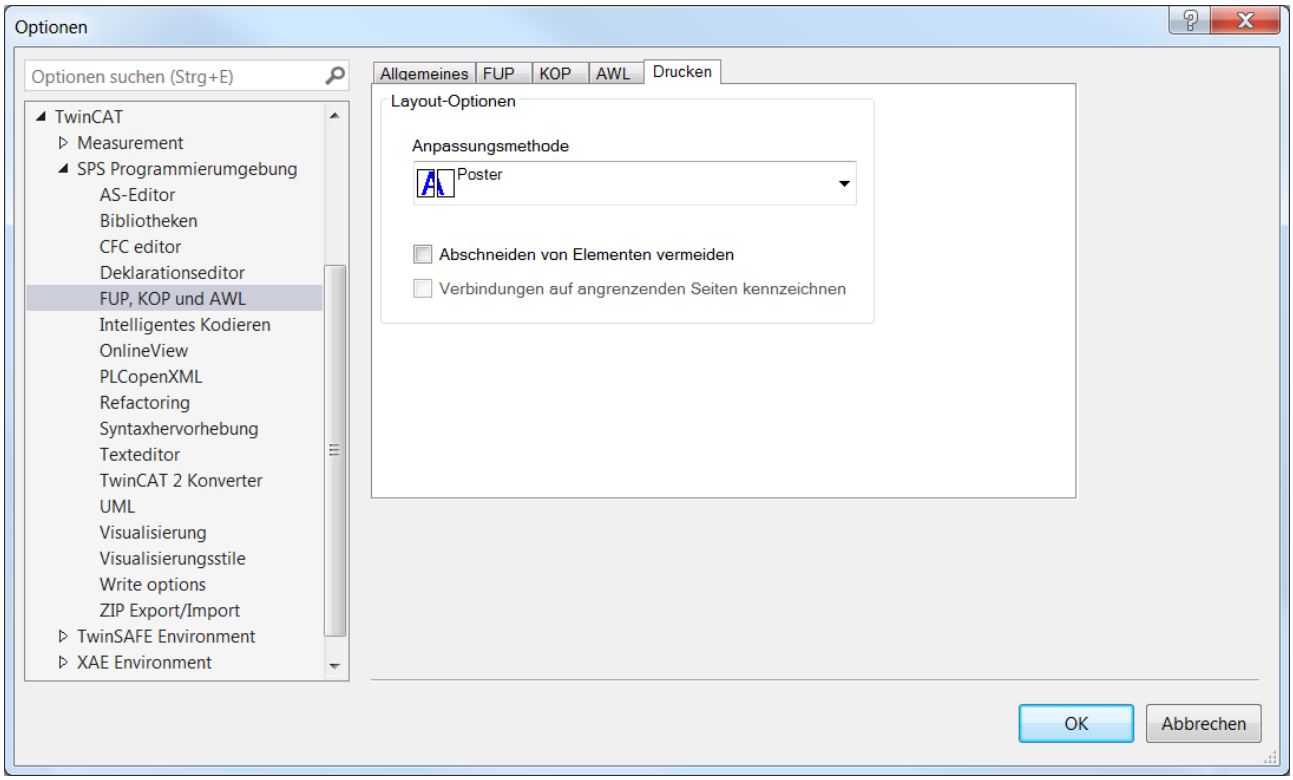
Ansicht

AWL aktivieren	Die Implementierungssprache AWL steht im Entwicklungssystem zur Verfügung.

Verhalten

Netzwerk Inhalt	Auswahlliste: Inhalt eines neuen Netzwerks.
Selektion nach Einfügen	Auswahlliste: Element, das TwinCAT nach dem Einfügen eines neuen Netzwerks selektiert.

Registerkarte Drucken



Layout-Optionen

Anpassungsmethode	Auswahlliste zur Größenanpassung.
Abschneiden von Elementen vermeiden	Elemente, die nicht auf die Seite passen, werden auf der nächsten Seite gedruckt.
Verbindungen auf angrenzenden Seiten kennzeichnen	Aktivierbar, wenn **Abschneiden von Elementen vermeiden** aktiviert ist.

Siehe auch:

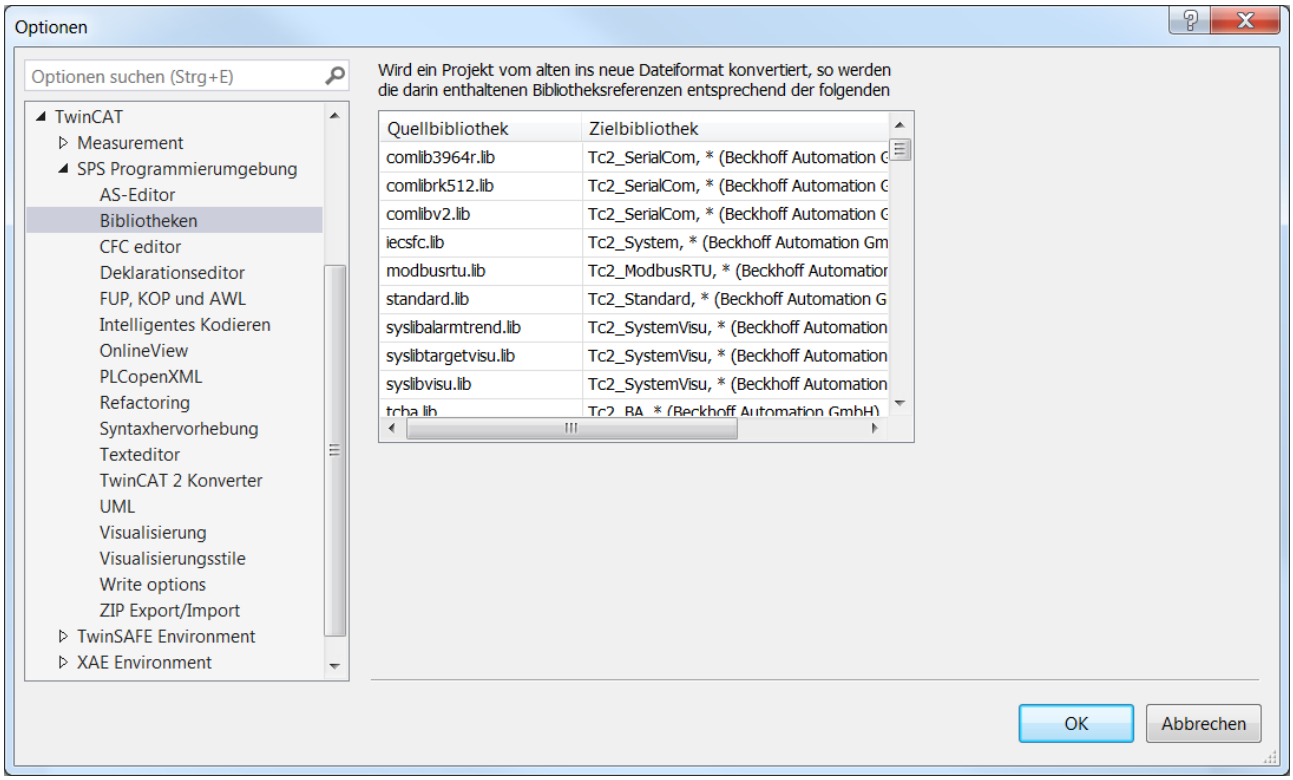
- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: Programmiersprachen und ihre Editoren [▶ 654]

17.9.1.4 Dialog Optionen - Bibliotheken

Funktion: Der Dialog enthält Einstellungen für Bibliotheken.

Aufruf: **TwinCAT > SPS Programmierumgebung > Bibliotheken**

Registerkarte Konvertierung



Die Registerkarte dient dem Verwalten der Abbildungen von Bibliotheksreferenzen, die TwinCAT bei der Konvertierung eines alten Projekts anwendet. Wenn Sie für eine bestimmte Bibliothek noch keine Abbildung gespeichert haben, müssen Sie bei jedem Öffnen eines altes Projekts, in das diese Bibliothek eingebunden ist, die Abbildung neu definieren.

Eine Abbildung definiert, wie eine Bibliotheksreferenz nach der Konvertierung des Projekts ins aktuelle Format aussieht. Es gibt drei Möglichkeiten:

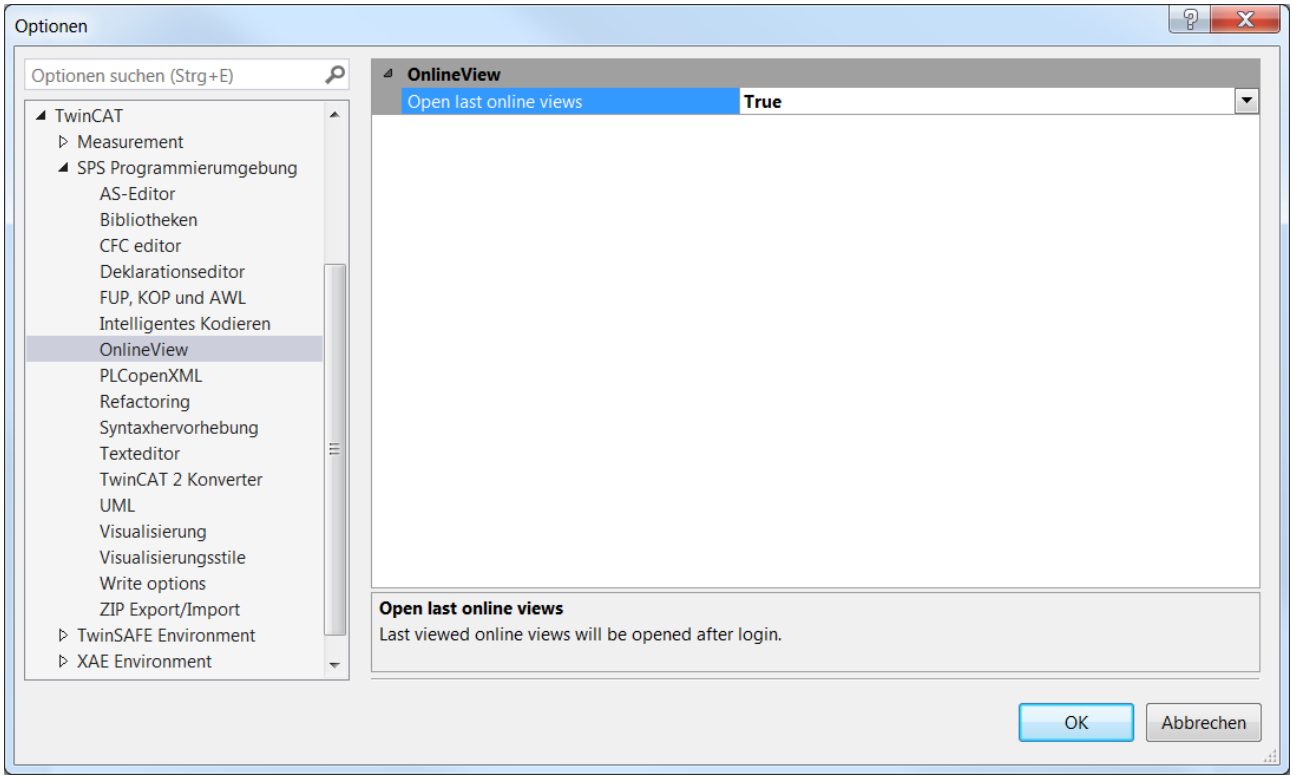
- Sie behalten die Referenz bei. Das bedeutet, dass TwinCAT die Bibliothek ebenfalls ins aktuelle Format (*.library) konvertiert und im lokalen Bibliotheks-Repository installiert.
- Sie ersetzen eine Referenz durch eine andere Referenz. Das bedeutet, dass eine der installierten Bibliotheken die bisher eingebundene Bibliothek ersetzt.
- Sie löschen die Referenz. Das bedeutet, dass das konvertierte Projekt die Bibliothek nicht mehr einbindet.

TwinCAT wendet alle aufgelisteten Abbildungen bei der nächsten Konvertierung eines alten Projekts auf dessen Bibliotheksreferenzen an. Somit müssen Sie die Abbildungsdefinition wiederholen, wenn dieselbe Bibliothek wieder in einem zu konvertierenden Projekt eingebunden ist. In der letzten Zeile können Sie eine neue Abbildung eingeben.	
Quellbibliothek	Pfad der Bibliothek, die im Projekt vor der Konvertierung eingebunden ist. Ein Doppelklick auf einen Eintrag macht das Feld editierbar und die Schaltfläche für die Eingabeunterstützung erscheint.
Zielbibliothek	Name und Speicherort der Bibliothek, die nach der Konvertierung im Projekt eingebunden sein soll. Ein Doppelklick auf einen Eintrag öffnet den Dialog „Zielsystembibliothek setzen“.

Siehe auch:

- Dokumentation PLC: Bibliotheken verwenden [▶ 278]

17.9.1.5 Dialog Optionen - Online View



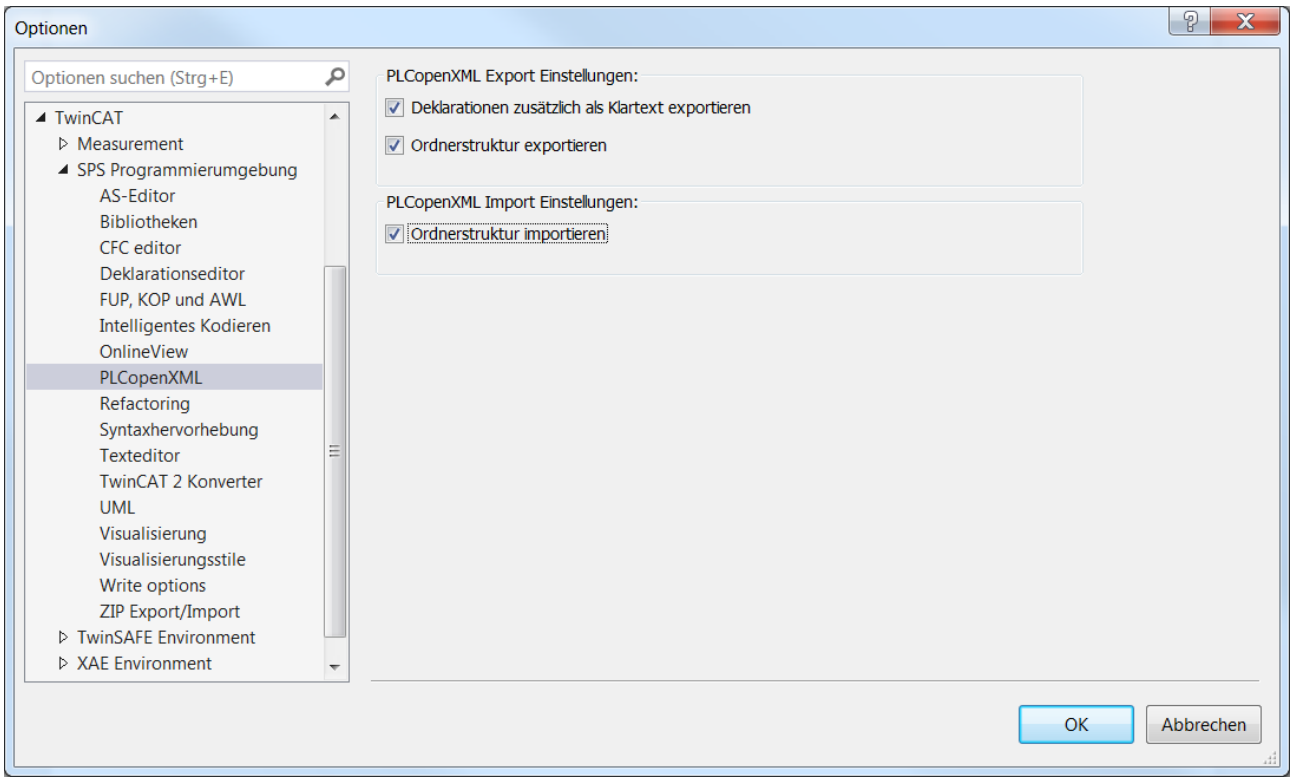
OnlineView

Open last online views	• TRUE (Standardeinstellung): Beim Einloggen werden die Editorfenster der vorherigen Online-Sitzung geöffnet. Die aktuelle Offline-Ansicht bleibt geöffnet. • FALSE: Beim Einloggen bleibt die Offline-Ansicht geöffnet. Die Editorfenster der vorherigen Online-Sitzung werden verworfen und nicht wieder geöffnet.

17.9.1.6 Dialog Optionen - PLCopenXML

Funktion: Der Dialog enthält Einstellungen für das Verhalten von TwinCAT beim PLCopenXML-Export oder -Import.

Aufruf: **TwinCAT > SPS Programmierumgebung > PLCopenXML**



PLCopenXML Export Einstellungen

Deklarationen zusätzlich als Klartext exportieren	Standardmäßig splittet TwinCAT die Deklarationsteile gemäß dem PLCopenXML-Schema in einzelne Variablen auf und verliert somit die Formatierung und manche Kommentarinformationen. ☑ : Formatierung und Kommentare bleiben erhalten. TwinCAT schreibt den Klartext des exportierten Deklarationsteils zusätzlich in die PLCopenXML-Datei und erweitert somit das PLCopenXML-Schema.
Ordnerstruktur exportieren	☑ : TwinCAT exportiert auch die Ordner, wenn sie eines der ausgewählten Objekte enthalten. Das ist eine TwinCAT-spezifische Erweiterung zum PLCopenXML-Schema.

PLCopenXML Import Einstellungen

Ordnerstruktur importieren	☑ : Wenn die Importdatei-Informationen über die Ordnerstruktur der Objekte enthält, importiert TwinCAT diese Struktur mit. ☐ : TwinCAT importiert Objekte ohne Struktur.

Siehe auch:

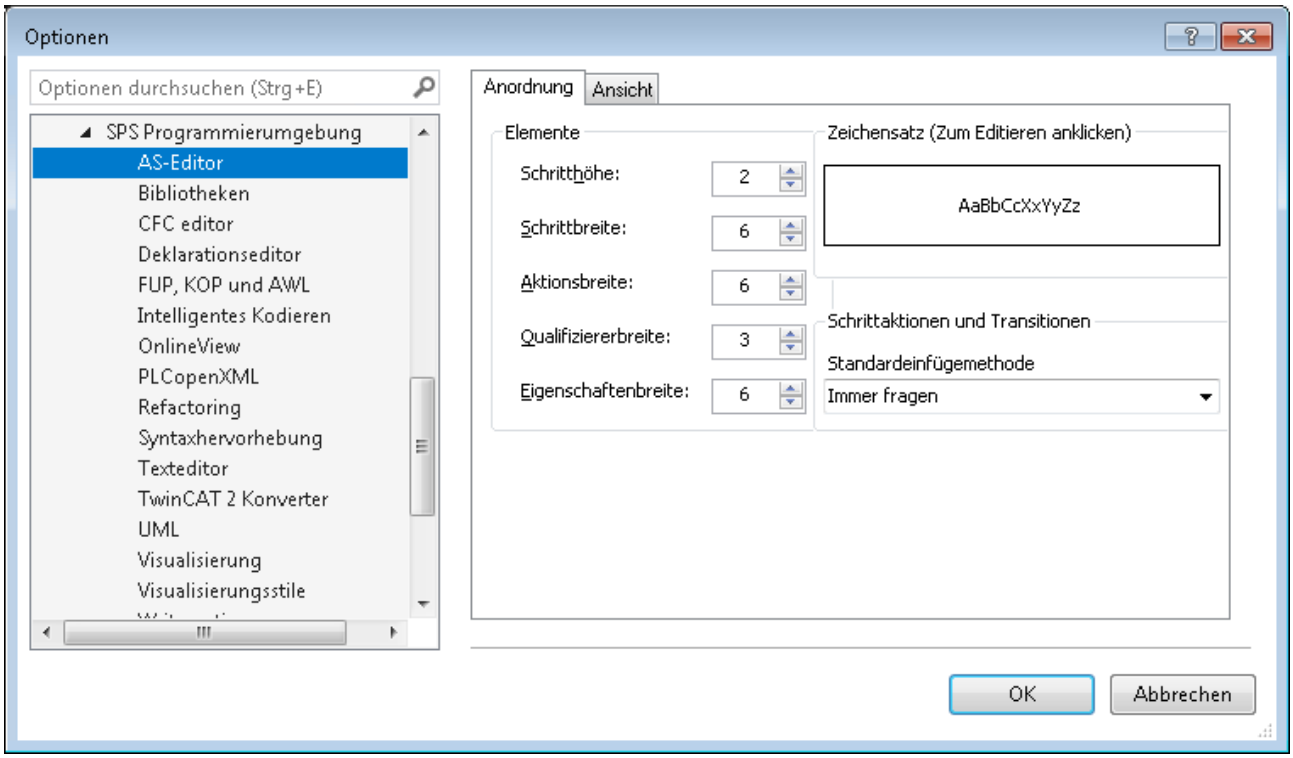
- Befehl PLCopenXML exportieren
- Befehl PLCopenXML importieren
- Dokumentation PLC: SPS-Projekt exportieren und importieren [▶ 64]

17.9.1.7 Dialog Optionen - AS-Editor

Funktion: Der Dialog dient der Konfiguration der Einstellungen für den AS-Editor.

Aufruf: **TwinCAT > SPS Programmierumgebung > AS-Editor**

Registerkarte Anordnung



Elemente

Definiert die Größen der AS-Elemente Schritt, Aktion, Qualifizierer und Eigenschaft. Angabe der Werte in Rastereinheiten angegeben. 1 Rastereinheit = Fontgröße, die Sie aktuell in den Texteditor-Optionen gesetzt haben (Textbereich / Schriftart). Die Einstellungen werden immer sofort in allen gerade geöffneten AS-Editorfenstern wirksam.	
Schritthöhe	Mögliche Werte: 1-100
Schrittbreite	Mögliche Werte: 2-100
Aktionsbreite	Mögliche Werte: 2-100
Qualifiziererbreite	Mögliche Werte: 2-100
Eigenschaftenbreite	Mögliche Werte: 2-100

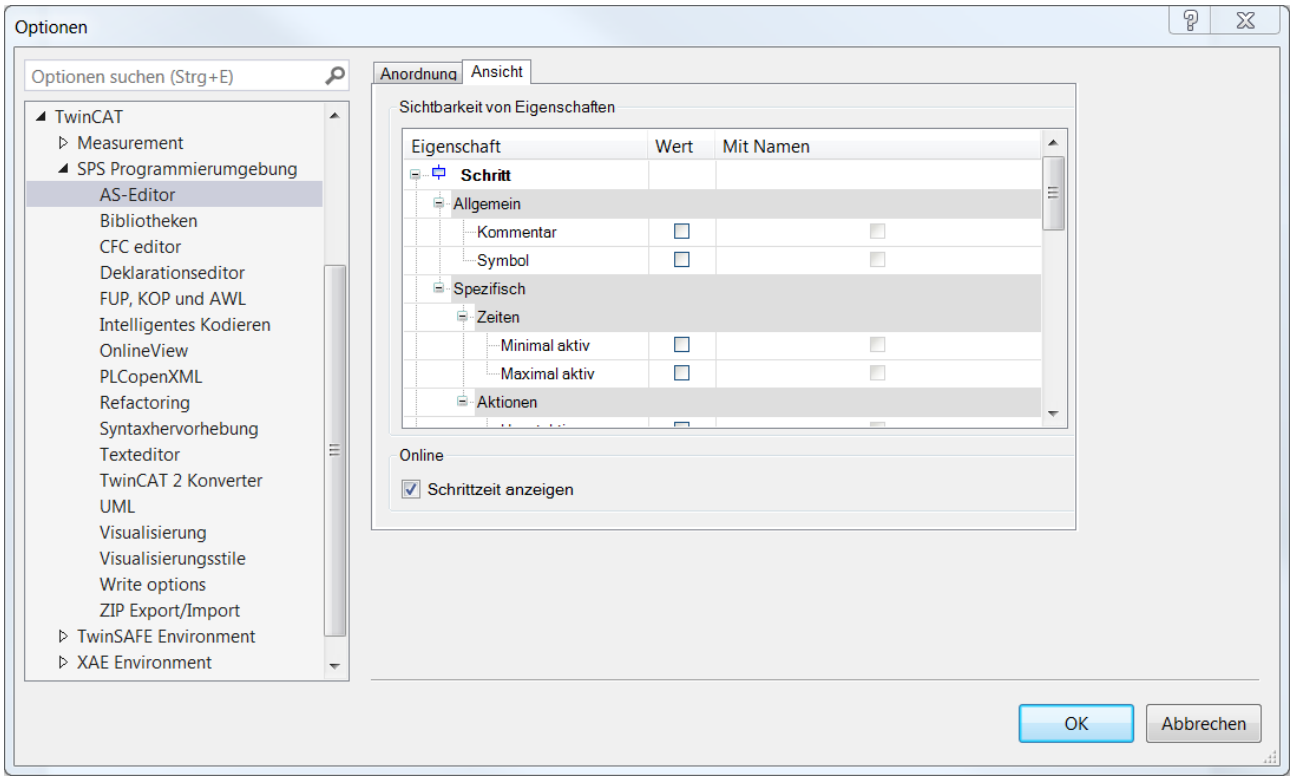
Zeichensatz

Der Beispieltext zeigt die aktuell eingestellte Schriftart. Klicken Sie darauf um die Schriftart zu verändern.

Schrittaktionen

Standardeinfügemethode	• Immer fragen • Implementierung duplizieren • Referenz kopieren

Registerkarte Ansicht



Sichtbarkeit von Eigenschaften

Auflistung der Elementeigenschaften der Kategorien Allgemein und Spezifisch und Definition der Anzeigeoptionen.	
Eigenschaft	Definiert die Elementeigenschaften, die neben dem Element im AS-Diagramm stehen.
Wert	☑ : Anzeige des Eigenschaftenwerts.
Mit Namen	☑ : Anzeige des Eigenschaftenwerts mit Namen.

Online

Schrittzeit anzeigen	☑ : TwinCAT zeigt im Onlinebetrieb rechts von den Schritten die Schrittzeit an.

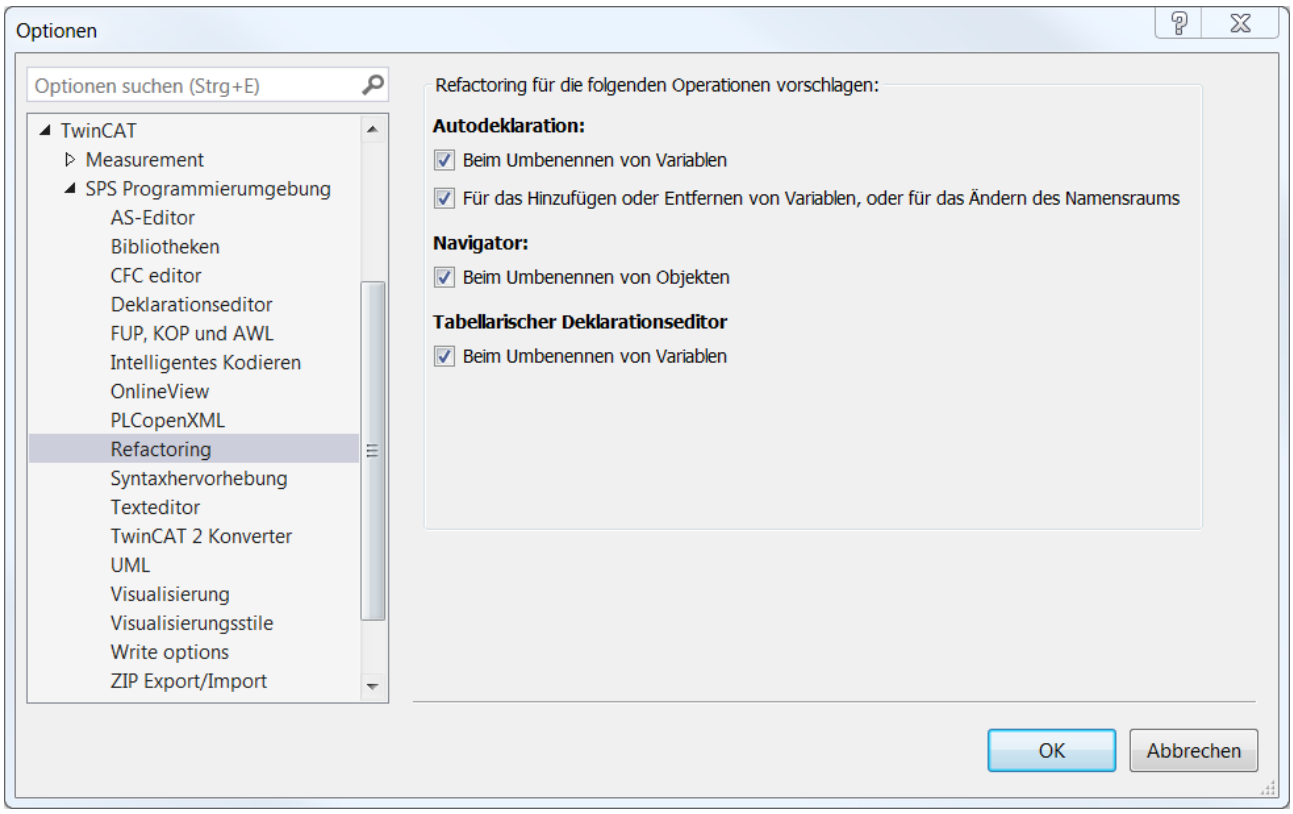
Siehe auch:

- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: Programmiersprachen und ihre Editoren [▶ 654]

17.9.1.8 Dialog Optionen - Refactoring

Funktion: Der Dialog dient zur Festlegung der Operationen im Projekt, für die automatisch Refactoring vorgeschlagen wird. Die Refactoring-Funktionalität unterstützt Sie bei Ihren Verbesserungswünschen.

Aufruf: **TwinCAT > SPS Programmierumgebung > Refactoring**



Autodeklaration	Wenn Sie den Namen einer Variablen ändern oder eine Eingabe- bzw. Ausgabevariable ergänzen, indem Sie die Autodeklaration (Dialog **Variable deklarieren**) aufrufen, ist die Option **Änderungen mit Hilfe von Refactoring anwenden** automatisch aktiviert. Nach Bestätigung des Dialogs öffnet sich der Dialog **Refactoring** und Sie können die Variable projektweit ändern. **Beim Hinzufügen oder Entfernen von Variablen, oder für das Ändern des Gültigkeitsbereichs** ☑: Sie fügen über die Autodeklaration (Dialog **Variable deklarieren**) eine neue Eingangs- oder Ausgangsvariable hinzu oder löschen in der Autodeklaration den Namen einer Variablen und beenden den Dialog mit **OK.** Daraufhin öffnet sich der Dialog **Refactoring**, um projektweit die Variable hinzuzufügen oder zu entfernen **Beim Umbenennen von Variablen** ☑: Sie benennen in der Autodeklaration (Dialog **Variable deklarieren**) den Namen um und beenden den Dialog mit **OK**. Daraufhin öffnet sich der Dialog **Refactoring**, um projektweit die Variable umzubenennen.
Navigator	**Beim Umbenennen von Objekten** ☑: Wenn Sie im SPS-Projektbaum den Namen eines Objekts ändern, erscheint die Eingabeaufforderung, ob TwinCAT „Automatisches Refactoring“ durchführen soll.
Tabellarischer Deklarationseditor	**Beim Umbenennen von Variablen** ☑: Wenn Sie im tabellarischen Deklarationseditor den Namen einer Variablen ändern, erscheint die Eingabeaufforderung, ob TwinCAT „Automatisches Refactoring“ für das Umbenennen durchführen soll.

UML Klassendiagramm	Optionen für die Unterstützung von Refactoring bei im Klassendiagramm-Editor vorgenommenen Änderungen. **Beim Hinzufügen oder Entfernen von Variablen** ☑: Wenn Sie im Klassendiagramm Variablen in den Sektionen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT hinzufügen oder entfernen, wird Refactoring unterstützt. **Beim Umbenennen eines Bausteins** ☑: Wenn Sie im Klassendiagramm einen Bausteinnamen ändern, wird Refactoring unterstützt. **Beim Umbenennen von Variablen oder Eigenschaften** ☑: Wenn Sie im Klassendiagramm eine Variable oder eine Eigenschaft umbenennen, wird Refactoring unterstützt. Wenn die Option **Refactoring-Vorschau überspringen** in den UML-Optionen aktiviert ist, wird das Refactoring je nach Fall möglicherweise ohne vorheriges Anzeigen im Dialog **Refactoring** an allen betroffenen Stellen im Projekt durchgeführt. (siehe Dialog Optionen - UML [▶ 1035])

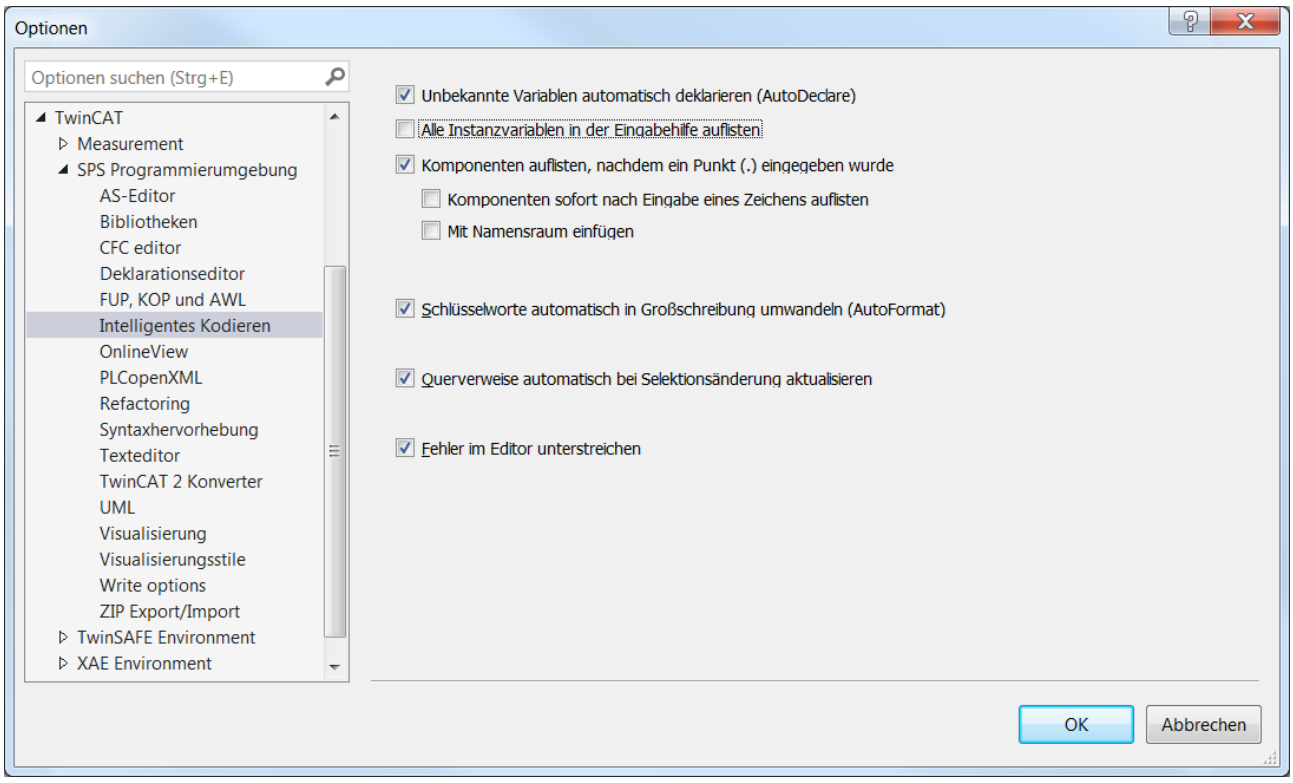
Siehe auch:

- Befehl Variable deklarieren [▶ 915]
- Befehl Refactoring - <Variable> umbenennen [▶ 926]
- Befehl Refactoring - Variable hinzufügen [▶ 928]
- Befehl Refactoring - <Variable> entfernen [▶ 929]
- Dokumentation PLC: Refactoring [▶ 168]

17.9.1.9 Dialog Optionen - Intelligentes Codieren

Funktion: Der Dialog dient der Konfiguration der Einstellungen, die die Eingabe von Code erleichtern.

Aufruf: **TwinCAT > SPS Programmierumgebung > Intelligentes Kodieren**


Optionen
Optionen suchen (Strg+E)
TwinCAT
Measurement
SPS Programmierumgebung
AS-Editor
Bibliotheken
CFC editor
Deklarationseditor
FUP, KOP und AWL
Intelligentes Kodieren
OnlineView
PLCopenXML
Refactoring
Syntaxhervorhebung
Texteditor
TwinCAT 2 Konverter
UML
Visualisierung
Visualisierungsstile
Write options
ZIP Export/Import
TwinSAFE Environment
XAE Environment
Unbekannte Variablen automatisch deklarieren (AutoDeclare)
Alle Instanzvariablen in der Eingabehilfe auflisten
Komponenten auflisten, nachdem ein Punkt (.) eingegeben wurde
Komponenten sofort nach Eingabe eines Zeichens auflisten
Mit Namensraum einfügen
Schlüsselworte automatisch in Großschreibung umwandeln (AutoFormat)
Querverweise automatisch bei Selektionsänderung aktualisieren
Fehler im Editor unterstreichen
OK
Abbrechen

Unbekannte Variablen automatisch deklarieren (AutoDeclare)	☑: Der Dialog **Variable deklarieren** öffnet sich, sobald Sie einen noch nicht deklarierten Bezeichner in einem Programmiersprachen-Editor eingegeben und die Eingabezeile verlassen haben. Damit die AutoDeclare-Funktion auch im ST-Editor zur Verfügung steht, muss ab Build 4026 zusätzlich die Option **Für den ST-Editor aktivieren** aktiviert sein.
Für den ST-Editor aktivieren	Voraussetzung: Die Option **Unbekannte Variablen automatisch deklarieren (AutoDeclare)** ist aktiviert. ☑: Die AutoDeclare-Funktion steht auch im ST-Editor zur Verfügung. ☐: Im ST-Editor steht die AutoDeclare-Funktion nicht zur Verfügung. (Verfügbar ab Build 4026)
Alle Variablen einer Instanz in der Eingabehilfe anzeigen	☑: Die Funktion **Komponenten auflisten** bietet auch die lokalen Variablen einer Funktionsbausteininstanz zur Auswahl an. ☐: Die Funktion **Komponenten auflisten** bietet nur die Eingangsvariablen und Ausgangsvariablen einer FB-Instanz zur Auswahl an.
Komponenten auflisten, nachdem ein Punkt (.) eingegeben wurde	☑: Aktiviert die Funktion **Komponenten auflisten**. Das bedeutet: Wenn Sie an einer Stelle, an der TwinCAT einen Bezeichner erwartet, einen Punkt eingeben, erscheint eine Auswahlliste mit Eingabemöglichkeiten.
Komponenten sofort nach Eingabe eines Zeichens auflisten	Voraussetzung: Option **Komponenten auflisten, nachdem ein Punkt (.) eingegeben wurde** ist aktiviert. ☑: Nach Eingabe einer beliebigen Zeichenfolge erscheint eine Auswahlliste der verfügbaren Bezeichner und Operatoren
Mit Namensraum einfügen	☑: Vor dem Kennzeichner fügt TwinCAT automatisch den Namensraum mit ein.
Schlüsselworte automatisch in Großschreibung umwandeln (Autoformat)	☑: TwinCAT schreibt automatisch alle Schlüsselwörter in Großbuchstaben.
Querverweise automatisch bei Selektionsänderung aktualisieren	☑: Die Querverweisliste zeigt automatisch die Referenzen der Variablen/ POUs/DUTs an, die Sie gerade selektieren oder in der der Cursor steht.
Fehler im Editor unterstreichen	☑: Fehlerhafter oder unbekannter Programmcode wird unterstrichen. (Verfügbar ab Build 4026)
Symbole hervorheben	☑: Alle Verwendungsstellen eines Symbols, auf dem der Cursor steht, werden innerhalb des Editors farblich markiert. So können Sie Querverweise innerhalb des Editors schnell erkennen. (Verfügbar ab Build 4026)

Siehe auch:

- Dokumentation PLC: Programmiersprachen und ihre Editoren [▶ 654]
- Dokumentation PLC: Eingabeunterstützung nutzen [▶ 141]
- Dokumentation PLC: Verwendungsstellen mit der Querverweisliste finden [▶ 165]

17.9.1.10 Dialog Optionen - Ladder-Editor

Symbol:

Funktion: Der Dialog dient der Konfiguration der Darstellungsoptionen für den Kontaktplan-Editor.

Aufruf: **TwinCAT > SPS Programmierumgebung > Ladder-Editor**

Registerkarte Allgemeines

Ansicht

Netzwerktitel anzeigen	Der Netzwerktitel wird in der oberen linken Ecke des Netzwerks angezeigt.
Netzwerkkommentar anzeigen	Das Netzwerkkommentar wird in der oberen linken Ecke des Netzwerks angezeigt. Wenn TwinCAT zusätzlich den Netzwerktitel darstellt, erscheint der Kommentar in der Zeile unterhalb.
Bausteinsymbol anzeigen	Das Bausteinsymbol wird im Bausteinelement im Ladder-Editor angezeigt. Auch die Standardoperatoren haben Symbole.
Operandenkommentar anzeigen	TwinCAT zeigt den Kommentar an, den Sie einer Variablen im Implementierungsteil gegeben haben. Der Operandenkommentar bezieht sich im Gegensatz zum „Symbolkommentar“ nur auf die lokale Verwendungsstelle der Variablen. Der Kommentar wird abhängig vom verfügbaren Platz automatisch umgebrochen.
Symbolkommentar anzeigen	TwinCAT zeigt den Kommentar, den Sie einer Variablen oder einem Symbol bei der Deklaration gegeben haben, oberhalb des Variablennamens an. Zusätzlich oder anstelle des Symbolkommentars können Sie auch einen lokalen „Operandenkommentar“ zuweisen.
Symboladresse anzeigen	Wenn einem Symbol (Variable) eine Adresse zugewiesen ist, wird diese Adresse oberhalb des Variablennamens angezeigt.

Operandengröße

Maximale Anzahl der angezeigten Zeilen	Maximale Anzahl von Zeilen des Operandennamens, die dargestellt werden.
Maximale Durchschnittszahl der Zeichen pro Zeile	Maximale Anzahl der Zeichen pro Zeile für die Darstellung des Operandennamens

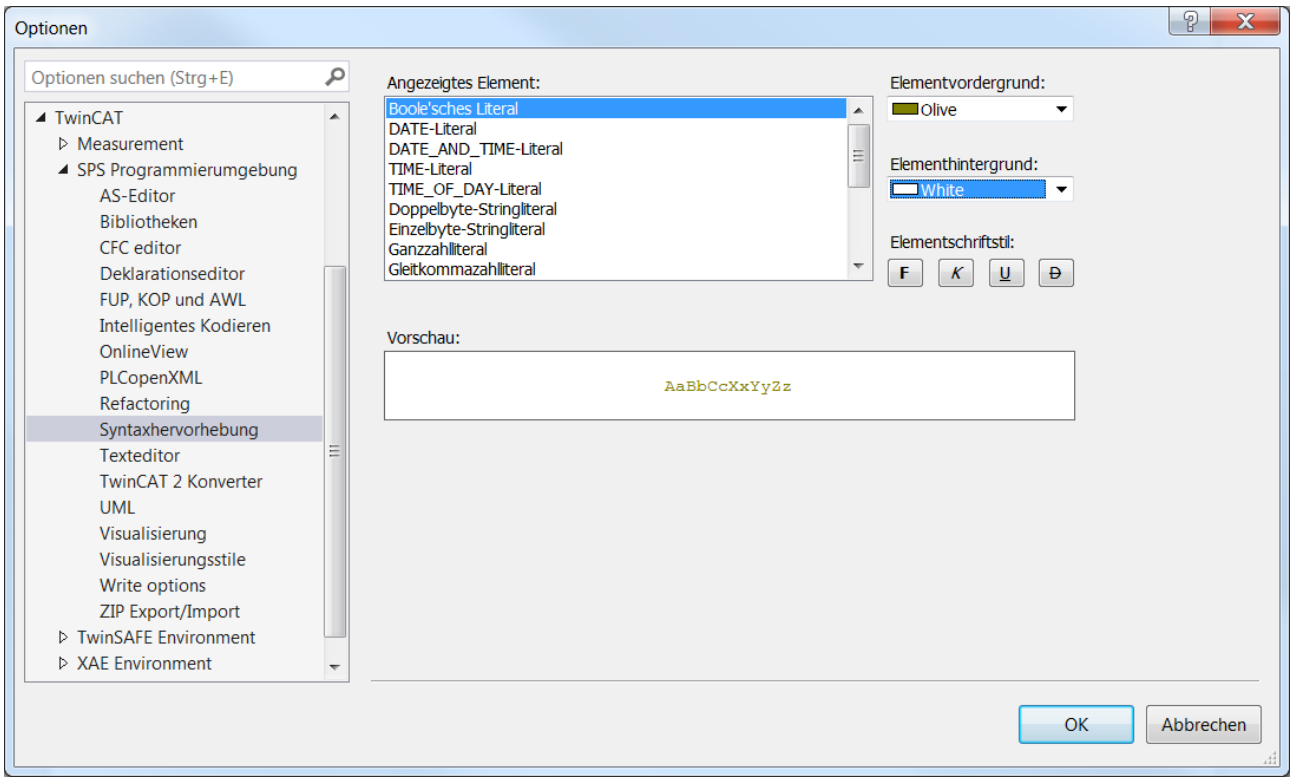
Registerkarte Monitoring

Angezeigte Ziffern für Gleitkommazahlen	Auswahlliste für die Anzahl der Ziffern Beispiel: *750.15* bei eingestellten *5* Ziffern wird zu *7.5e+02* bei eingestellten *2* Ziffern.
Angezeigte Länge für Zeichenfolgen	Auswahlliste für die Anzahl der Zeichen

17.9.1.11 Dialog Optionen - Syntaxhervorhebung

Funktion: Der Dialog dient der Konfiguration der Farb- und Schrifteinstellungen für die Textelemente eines Editors (zum Beispiel Operanden, Pragmas).

Aufruf: **TwinCAT > SPS Programmierumgebung > Syntaxhervorhebung**



Angezeigtes Element	Auswahlliste für Textelemente
Elementvordergrund	Vordergrundfarbe des Textelements
Elementhintergrund	Hintergrundfarbe des Textelements
Elementschriftstil	Schriftstil des Textelements (fett, kursiv, unterstrichen, durchgestrichen)
Vorschau	Der Beispieltext zeigt die aktuell eingestellten Einstellungen in einer Vorschau

17.9.1.12 Dialog Optionen - Texteditor

Funktion: Der Dialog enthält Einstellungen für die Darstellung und das Arbeiten in einem Texteditor.

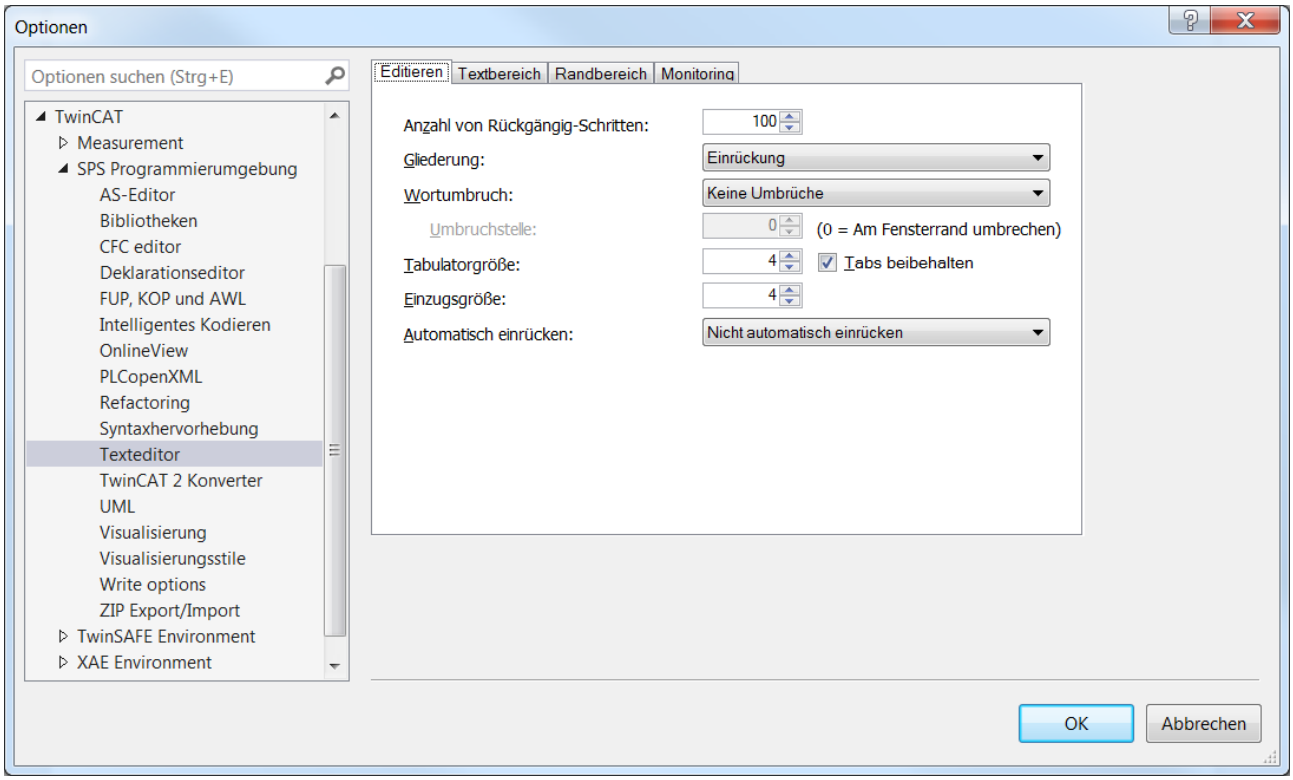
Aufruf: **TwinCAT > SPS Programmierumgebung > Texteditor**

Registerkarte Theme

In dieser Registerkarte stellen Sie das gewünschte Theme für die Oberflächengestaltung des ST-Editors ein. (Verfügbar ab Build 4026)

Theme	Farbschema für den Texteditor. Das gewählte Theme wird im Fenster Vorschau dargestellt.

Registerkarte Editieren

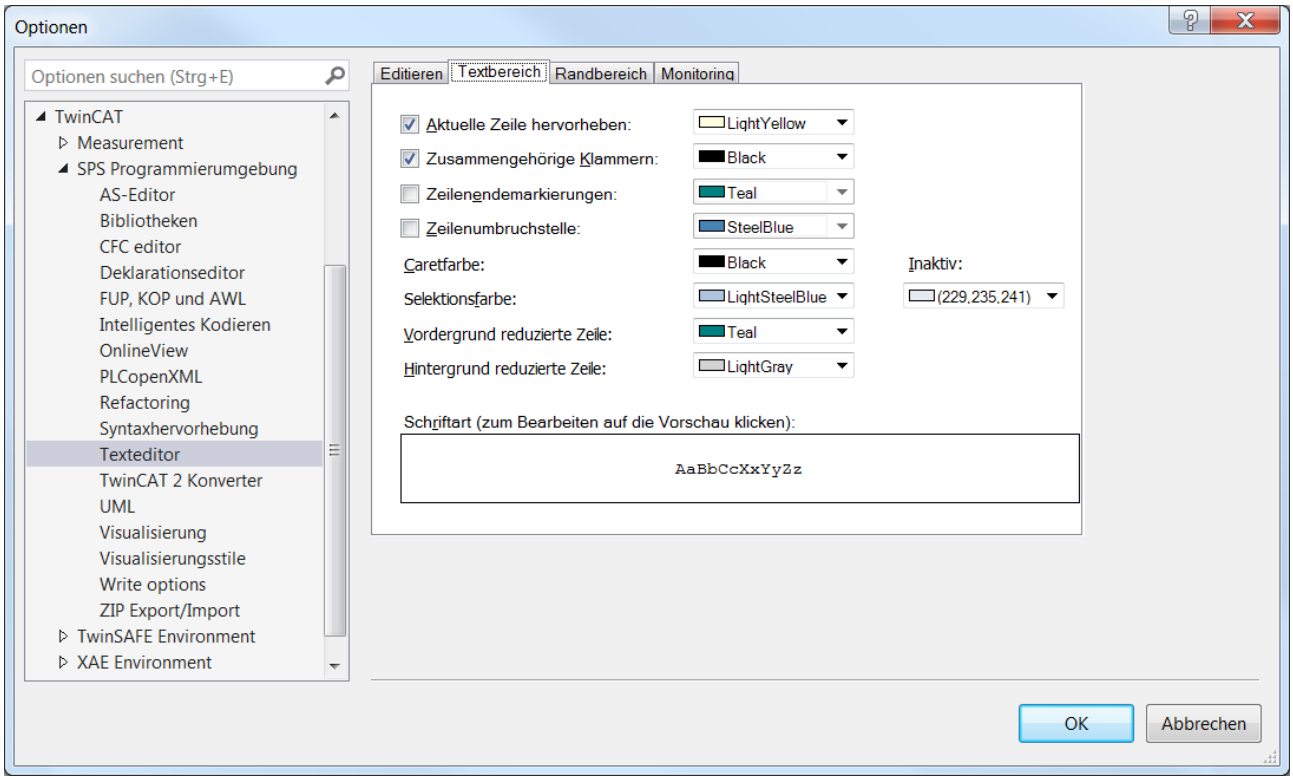


Anzahl von Rückgängig-Schritten	Maximale Anzahl der Bearbeitungsschritte, auf die Sie den Befehl **Bearbeiten > Rückgängig** ausführen können.

Gliederung	Definiert die Strukturierung des Codes durch Einrückungen.
	Wenn Sie eine Einrückung auswählen, können Sie den Einrückungsabschnitt mithilfe eines Plus- und Minuszeichens vor der ersten Zeile des jeweiligen Abschnitts auf- oder zuklappen.
	• **Einrückung**: TwinCAT fasst alle Zeilen, die gegenüber der vorausgehenden Zeile eingerückt sind, in einer Einrückungseinheit zusammen.
	• **Explizit**: Sie kennzeichnen explizit den Code-Abschnitt mit Kommentaren, der in einer Einrückungseinheit zusammengefasst werden soll: Vor dem Abschnitt muss ein Kommentar stehen, der 3 öffnende geschweifte Klammern „{{{„ enthält, nach dem Abschnitt muss ein Kommentar folgen, der 3 schließende geschweifte Klammern „}}}“ enthält. Die Kommentare können zusätzlichen Text enthalten. Beispiel:
	`IF nVar1=1`
	`///comment {{{`
	`THEN`
	`nVar2:=2;`
	`ELSE nVar2:=10;`
	`END_IF`
	`///}}}`
	`nVar1:=nVar1+1;`
	`IF nVar1=1`
	`///comment {{{ [5 lines]`
	`nVar1:=nVar1+1;`

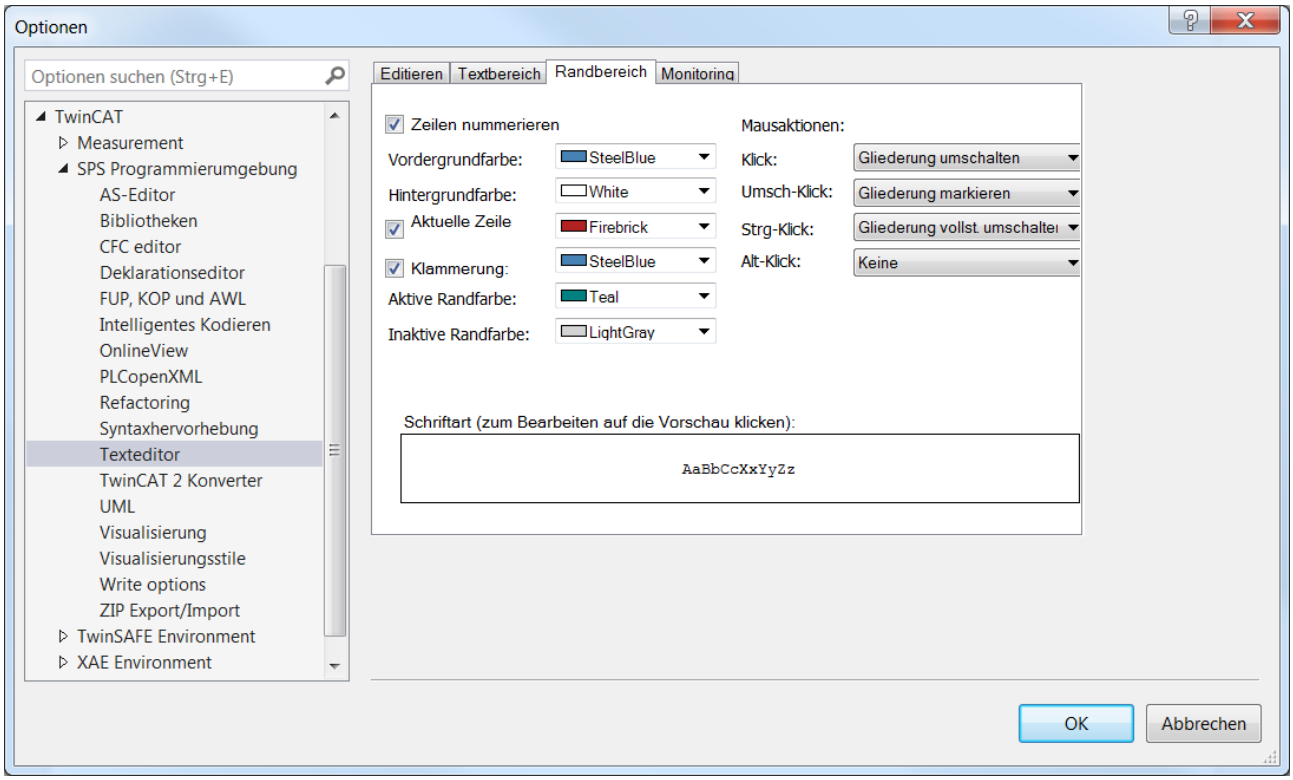
Wortumbruch	• **Weich**: Der Zeilenumbruch erfolgt am Rand des Editorfensters, wenn bei **Umbruchsstelle** 0 eingetragen ist. • **Hart**: Der Zeilenumbruch erfolgt nach der bei Umbruchsstelle angegebenen Anzahl von Zeichen.
Tabulatorgröße	Anzahl der Zeichen
Tabs beibehalten	☑: Den Leerraum, den Sie mit der [Tabulator] -Taste eingefügt haben, löst TwinCAT hinterher nicht in einzelne Leerzeichen auf.
Einzugsgröße	Wenn Sie bei der Option **Automatisch Einrücken** Intelligent oder Intelligent mit Code-Komplettierung aktiviert haben, fügt TwinCAT die Anzahl Leerzeichen am Beginn der Zeile ein.
Automatisch einrücken	• **Nicht automatisch einrücken** • **Block**: Eine neue Zeile übernimmt automatisch die Einrückung der Vorgängerzeile. • **Intelligent**: Zeilen, die einer Zeile folgen, die ein Schlüsselwort enthält (zum Beispiel VAR), rücken automatisch um die angegebene Einzugsgröße ein. • **Intelligent mit Code-Komplettierung**: Einrückung wie bei der Option **Intelligent**, zusätzlich fügt TwinCAT das abschließende Schlüsselwort ein (zum Beispiel END_VAR).

Registerkarte Textbereich



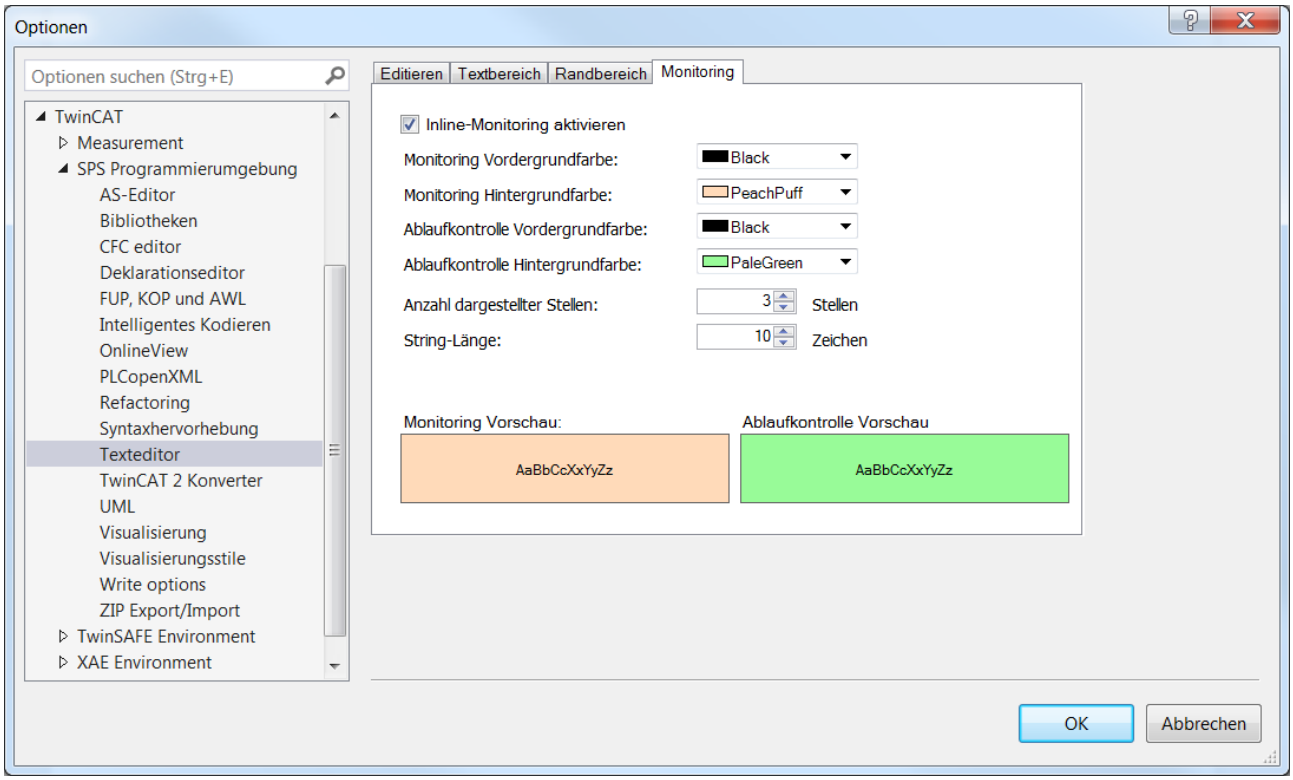
Aktuelle Zeile hervorheben	☑ : Die Zeile, in der der Cursor steht, wird farblich hinterlegt.
Zusammengehörige Klammern	☑ : Wenn der Cursor vor oder nach einer Klammer innerhalb einer Code-Zeile positioniert ist, markiert TwinCAT die zugehörige schließende oder öffnende Klammer durch einen Rahmen.
Zeilenmarkierungen	☑ : Das Ende jeder Editorzeile markiert TwinCAT durch einen kleinen Querstrich hinter dem letzten Zeichen (auch Leerzeichen) der Zeile.
Zeilenumbruchstelle:	☑ : Wenn ein weicher oder harter Zeilenumbruch aktiviert ist, wird die definierte Zeilenumbruchstelle durch eine senkrechte Linie angezeigt.
Caret-Farbe	Farbe des Cursorzeichens
Selektionsfarbe	Farbe des selektierten Textbereichs
Inaktiv	Farbe einer Selektierung, wenn das zugehörige Fenster nicht aktiv ist (Fokus liegt auf einem anderen Fenster).
Vordergrund reduzierte Zeile	Farbe der Kopfzeile eines geschlossenen, eingerückten Abschnitts im Code
Hintergrund reduzierte Zeile	Kopfzeile eines geschlossenen, eingerückten Abschnitts im Code wird in der Farbe hinterlegt.
Schriftart	Ein Klick auf das Feld öffnet den Standarddialog zum Konfigurieren der Schriftart.

Registerkarte Randbereich



Einstellungen für linken Randbereich des Texteditor-Fensters, der durch eine senkrechte Linie vom Eingabebereich abgetrennt ist:	
Zeilen nummerieren	☑ : Anzeige der Zeilennummern im Deklarations- und Implementierungsteil, jeweils mit 1 beginnend
Vordergrundfarbe	Farbe der Zeilennummern
Hintergrundfarbe	Farbe des Randbereichs
Aktuelle Zeile hervorheben	☑ : Die Zeilennummer der Zeile, in der der Cursor steht, wird farblich hervorgehoben.
Klammerung	☑ : Eine Klammerung umfasst die Zeilen zwischen den Schlüsselwörtern, die ein Konstrukt öffnen und abschließen, zum Beispiel IF und END_IF. Wenn die Option aktiviert ist und der Cursor vor, nach oder in einem der Schlüsselwörter eines Konstrukts steht, wird der Klammerungsbereich durch eine eckige Klammer im Randbereich angezeigt.
Aktive Randfarbe	Farbe der Trennlinie zwischen Rand- und Eingabebereich
Inaktive Randfarbe	Farbe der Trennlinie zwischen Rand- und Eingabebereich des gerade nicht aktiven Teils des Fensters
Mausaktionen	Eine der folgenden Aktionen können Sie jeder der angegebenen Mausaktionen oder Maus-Tastenkombinationen zuordnen. TwinCAT führt die Aktionen aus, wenn Sie die Mausaktion auf das Plus- oder Minuszeichen vor der Kopfzeile eines geklammerten Bereichs ausführen: • **Keine**: Die Mausaktion löst keine Aktion aus. • **Gliederung markieren**: TwinCAT wählt alle Zeilen des geklammerten Bereichs aus. • **Gliederung umschalten**: TwinCAT öffnet oder schließt den geklammerten Bereich, oder wenn geschachtelte Klammerungen vorliegen, die erste Ebene des geklammerten Bereichs. • **Gliederung vollst. umschalten**: TwinCAT öffnet oder schließt alle Ebenen eines geschachtelt geklammerten Bereichs.

Registerkarte Monitoring



Einstellungen für die Darstellung der Monitoring-Felder	
Inline-Monitoring aktivieren	☑ : Anzeige der Monitoring-Felder hinter den Variablen im Online-Modus
Monitoring Vordergrundfarbe	Darstellung des Werts im Monitoring-Feld
Monitoring Hintergrundfarbe	Darstellung des Hintergrunds im Monitoring-Feld
Ablaufkontrolle Vordergrundfarbe	Darstellung des Werts in den Monitoring-Feldern an den Ablaufkontroll-Positionen
Ablaufkontrolle Hintergrundfarbe	Darstellung des Hintergrunds der Monitoring-Felder an den Ablaufkontroll-Positionen
Anzahl dargestellter Stellen	Anzahl von Kommastellen im Monitoring-Feld
String-Länge	Maximale Länge von String-Variablenwerten im Monitoring-Feld

Siehe auch:

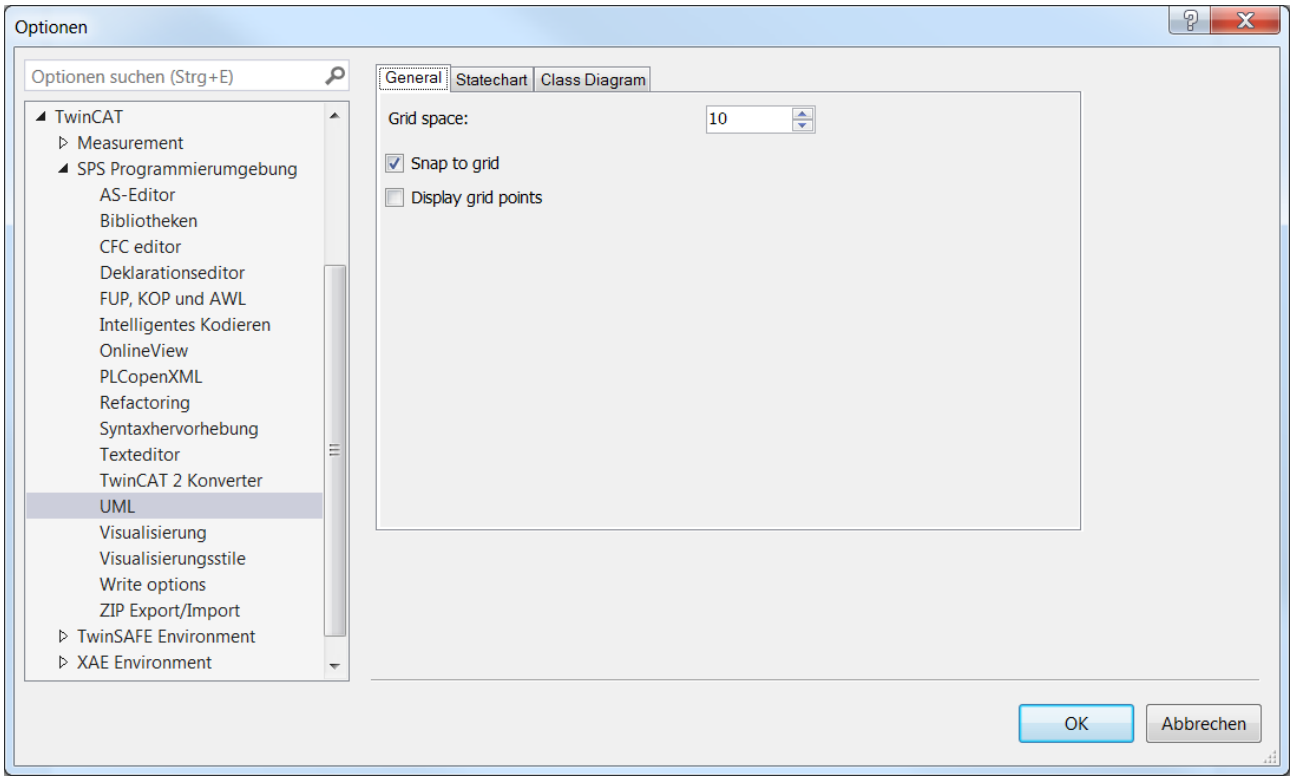
- Dokumentation PLC: Programmiersprachen und ihre Editoren [▶ 654]

17.9.1.13 Dialog Optionen - UML

Funktion: Der Dialog dient der Konfiguration des UML-Editors.

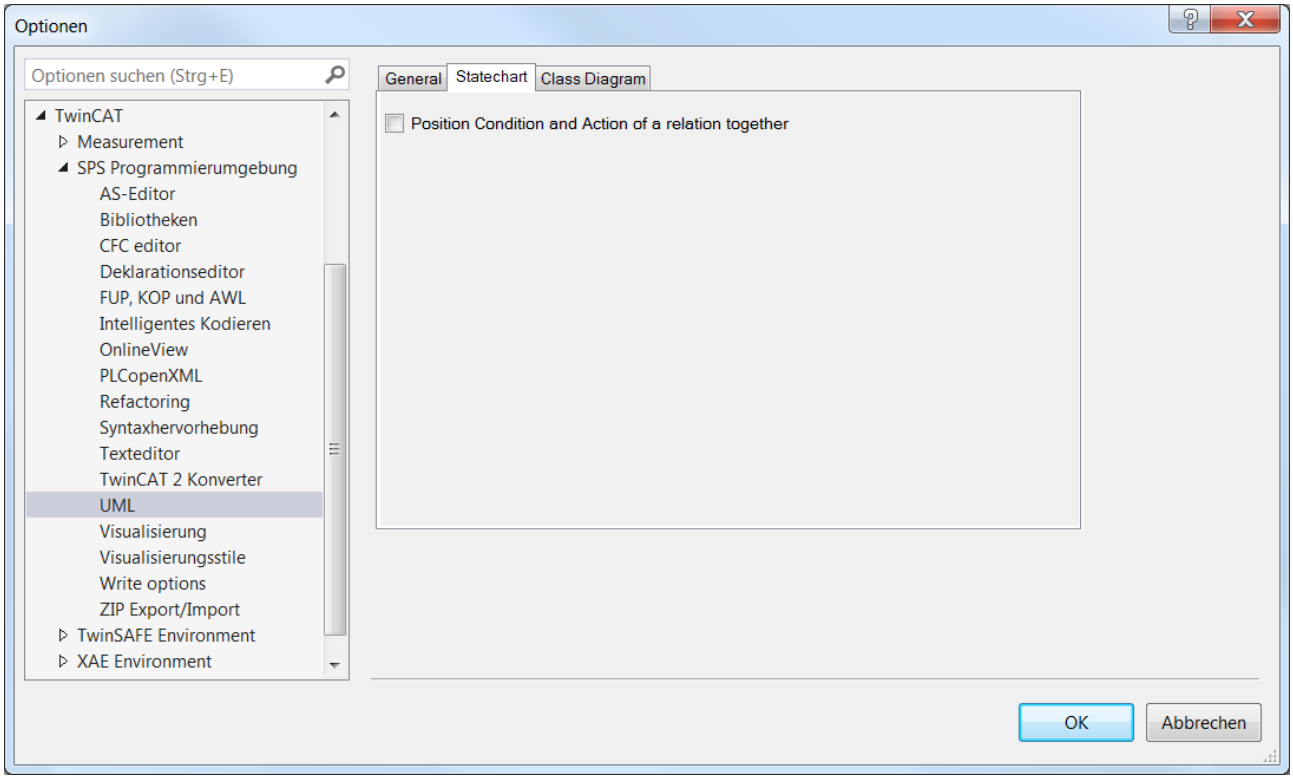
Aufruf: **TwinCAT > SPS Programmierumgebung > UML**

Registerkarte Allgemein



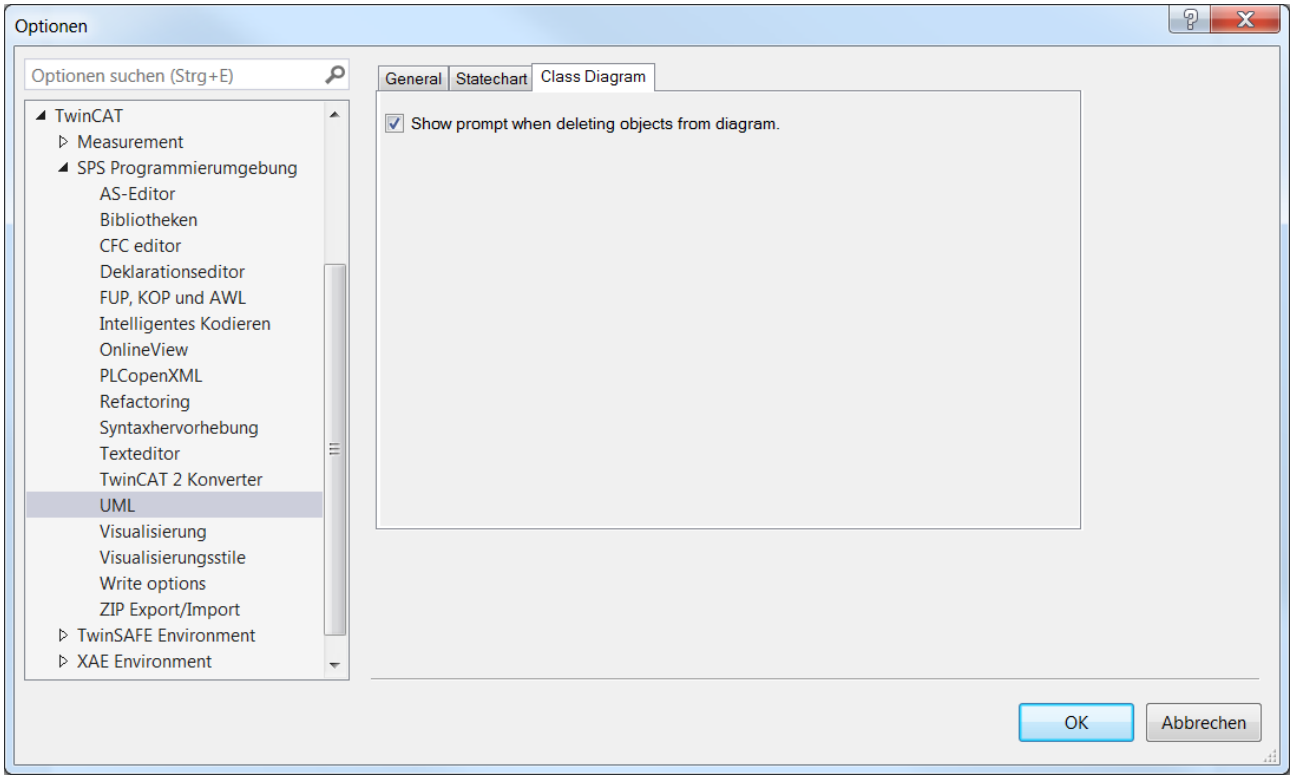
Rastermaß	Rasterlinienabstand in Pixel. Standardwert: 10
Am Raster einhängen	☑: Alle Elemente in den UML-Editoren werden am Raster ausgerichtet.
Rasterpunkte anzeigen	☑: Die Rasterpunkte werden in den UML-Editoren angezeigt.

Registerkarte Zustandsdiagramm



Bedingung und Aktion einer Beziehung gemeinsam positionieren	☑: Im Zustandsdiagramm werden eine Wächterbedingung und eine Aktion, die zur selben Transition gehören, synchron verschoben.

Registerkarte Klassendiagramm



Eingabeaufforderung, wenn Objekte aus dem Diagramm gelöscht werden	Objekte können entweder nur aus dem Diagramm oder aus dem Diagramm und aus dem Projekt gelöscht werden. ☐: Das Objekt wird standardmäßig nur aus dem Diagramm gelöscht. ☑: Beim Löschen eines Objekts erscheint ein Auswahlfenster, um zu konfigurieren, ob das Objekt nur aus dem Diagramm oder auch aus dem Projekt gelöscht werden soll.
Refactoring-Vorschau überspringen	☑: Wenn im Diagramm Refactoring angestoßen wird, wird die projektweite Änderung durchgeführt, ohne vorher den Dialog **Refactoring** mit einer Vorschau aller Änderungsstellen zu öffnen. Siehe auch: • Dokumentation PLC: Refactoring [▶ 168]

Siehe auch:

- Dokumentation UML: Überblick

17.9.1.14 Dialog Optionen - Visualisierung

Funktion: Der Dialog dient der Konfiguration des Visualisierungseditors.

Aufruf: **TwinCAT > SPS Programmierumgebung > Visualisierung**

Registerkarte Allgemein

Diese Einstellungen werden ausschließlich bei der integrierten Visualisierung, nicht aber bei den Darstellungsvarianten TwinCAT PLC HMI (TargetVisu) und TwinCAT PLC HMI Web angewendet.

Darstellungsoptionen

Fest	Die Visualisierung behält ihre Originalgröße.
Isotropisch	Die Visualisierung behält ihre Proportionen.
Anisotropisch	Die Visualisierung passt sich der Größe des Fensters im Entwicklungssystem an
Antialiased Zeichnen	Die Visualisierung zeichnet sich mithilfe von Antialiasing-Methoden, sowohl während des Editierens, als auch als integrierte Visualisierung zur Laufzeit. Tipp: Wenn auf einer konkreten Visualisierungsplattform eine horizontale oder vertikale Linie unscharf gezeichnet wird, kann dies derzeit durch eine Verschiebung um 0.5px in Richtung der Liniendicke korrigiert werden; siehe Elementeigenschaft **Absolute Bewegung**, Option **REAL-Werte verwenden**. Voraussetzung: Die verwendete Plattform unterstützt die Verwendung von REAL-Koordinaten

Bearbeitungsoptionen

Mit Umschalten-/ Tastenvariable verknüpfen	Der Platzhalter <Umschalten-/Tastenvariable> in den Visualisierungselementeigenschaften ist aktiviert. Wenn Sie ein Element, das über die Eigenschaft Farbvariablen, Farbumschlag verfügt, in die Visualisierungseditor ziehen, wird diese Eigenschaft mit dem Platzhalter <Umschalten-/Tastenvariable> konfiguriert sein. Folgende Elemente sind betroffen: Schaltfläche, Frame, Bild, Linie, Kreissektor, Polygon, Rechteck, Textfeld, Scrollbalken

Registerkarte Raster

Raster

Sichtbar	Im Visualisierungseditor sind Rasterlinien im Abstand Größe sichtbar
Aktiv	Im Visualisierungseditor sind Rasterlinien im Abstand Größe aktiv. Die Elemente sind am Raster ausgerichtet, ihre Positionswerte liegen jeweils auf einer Rasterlinie. Ein Element, das beim Aktivieren des Rasters bereits in einer Visualisierung ist, wird nicht automatisch ausgerichtet. Dazu müssen Sie es erst auf eine andere Position ziehen. Die Rasterlinien können aktiv und dabei unsichtbar sein
Größe	Abstand der Rasterlinien in Pixel

Registerkarte Dateioptionen

Textdatei für textuelles “Komponenten auflisten“	Dateiname und Speicherort einer Datei des Typs .csv. Sie enthält eine Tabelle mit Texten im Format einer Textliste. Die Einträge der Datei werden bereitgestellt, wenn die Funktion **Komponenten auflisten** als Eingabeunterstützung verwendet wird. Sie erzeugen diese Datei als Exportdatei der globalen Textliste mit dem Befehl Import/Export Textlisten.

Siehe auch:

- Dokumentation PLC: Visualisierung erstellen [▶ 393]

17.9.1.15 Dialog Optionen - Visualisierungsstile

Funktion: Der Dialog dient der Konfiguration der Visualisierungsstile.

Aufruf: **TwinCAT > SPS Programmierumgebung > Visualisierungsstile**

Stilauswahl

Alle Versionen anzeigen (nur für Experten)	☐ Zur Auswahl stehen, neben dem aktuell ausgewählten Stil, alle weiteren Stile des Repositorys, aber nur in der aktuellsten Version. Wenn für den ausgewählten Stil neuere Versionen installiert sind, werden diese auch aufgelistet. ☑ Zur Auswahl stehen alle installierten Stile in allen installierten Versionen.

Stil für neue Visualisierungsmanager

Zuletzt verwendet: <Stil, Version, Hersteller>	Stil, der automatisch als ausgewählt eingestellt wird, wenn Sie eine neue Visualisierungsapplikation hinzufügen. Gerätebedingt ist es möglich, dass eine Darstellungsvariante trotz dieser Einstellung auf andere Art und Weise dargestellt wird.
Voreinstellung: <Stil, Version, Hersteller>	Herstellerseitige Voreinstellung des Stils einstellen.
<Stil, Version, Hersteller>	Darstellungsvariante für Stil, Version und Hersteller einstellen.

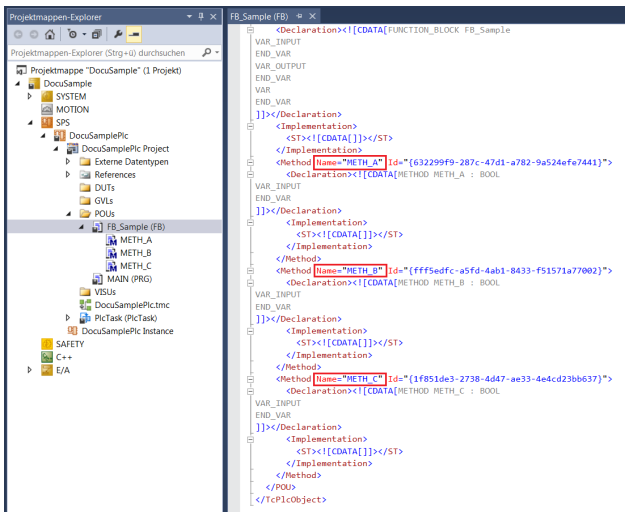
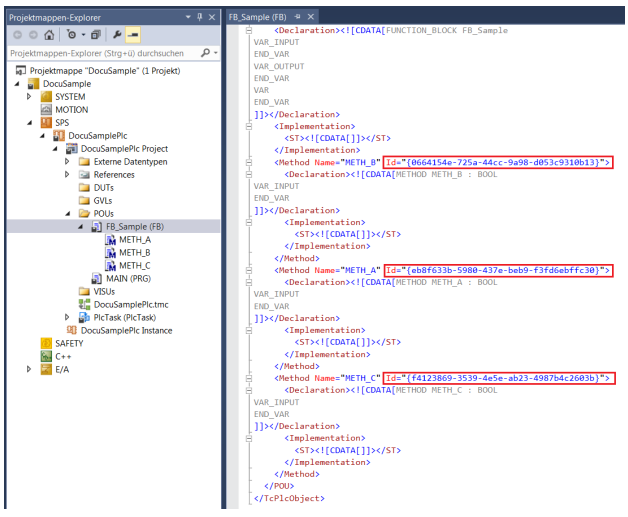
17.9.1.16 Dialog Optionen - Write Options

Write Options

Separate Line IDs	• TRUE: Die Line-IDs einer POU werden in einer separaten Datei (LineIDs.dbg) gespeichert, damit Änderungen in den Line-IDs nicht zu Änderungen in der POU führen, die dann im Source-Control-System als inhaltliche Änderung missinterpretiert werden würden. Voraussetzung dafür ist ab TC3.1 Build 4026, dass „Write Line IDs“ den Wert TRUE hat. (Standardeinstellung: FALSE) Bis TC3.1 Build 4024 werden Line-IDs u. a. für das Breakpoint-Handling benötigt und stellen sicher, dass eine Zuordnung der Code-Zeilen zu Maschinencode-Anweisungen möglich ist.
Sort by name	• TRUE (Standardeinstellung): Die Unterelemente von POUs (Aktionen, Methoden, Properties) werden nach Namen und nicht nach interner ID sortiert (siehe Beispiel [▶ 1040]).
Write Line IDs	Verfügbar ab TC3.1 Build 4026 • TRUE: In neuen Projekten werden für POUs Line-IDs erzeugt und gespeichert. (Standardeinstellung: FALSE) Bei dieser Einstellung handelt es sich um die globale Standardeinstellung. Bei der Erstellung eines neuen SPS-Projekts wird der Wert dieser Einstellung einmalig in die lokale Projekteinstellung übernommen. Diese ist in den SPS-Projekteigenschaften (Kategorie Advanced [▶ 961]) zu finden und kann dort projektbezogen angepasst werden.
Write PLC Bookmarks to File	Verfügbar ab TC3.1 Build 4026 • TRUE: In neuen Projekten werden die Lesezeichen in einer separaten .bookmarks Datei im Projektverzeichnis ablegt. (Standardeinstellung: FALSE) Bei dieser Einstellung handelt es sich um die globale Standardeinstellung. Bei der Erstellung eines neuen SPS-Projekts wird der Wert dieser Einstellung einmalig in die lokale Projekteinstellung übernommen. Diese ist in den SPS-Projekteigenschaften (Kategorie Advanced [▶ 961]) zu finden und kann dort projektbezogen angepasst werden.

Beispiel

Das Beispiel verdeutlicht die unterschiedliche Speicherreihenfolge der Methoden `METH_A`, `METH_B` und `METH_C`, je nachdem, ob die Option **Sort by name** aktiviert oder deaktiviert ist. Wenn die Option deaktiviert (FALSE) ist, steht die Methode `METH_B` nicht entsprechend ihres Namens an zweiter Stelle, sondern entsprechend ihrer internen ID an erster Stelle.

Sort by name = TRUE	Sort by name = FALSE
	

17.9.1.17 Dialog Optionen - ZIP Export/Import

Funktion: Der Dialog dient der Konfiguration der ZIP-Export- und Importeinstellungen.

Aufruf: **TwinCAT > SPS Programmierumgebung > ZIP Export/Import**

Siehe auch:

- Dokumentation PLC: SPS-Projekt exportieren und importieren [▶ 64]

17.9.2 Befehl Anpassen

Funktion: Der Befehl öffnet den Dialog **Anpassen**. Der Dialog enthält Registerkarten für die Konfiguration der Benutzeroberfläche. Hier können Sie die Menüs, die Symbolleisten und die Tastaturbelegung Ihren individuellen Anforderungen anpassen.

Aufruf: Menü **Extras**

Sie können die TwinCAT-Standardeinstellungen jederzeit über die Schaltfläche **Zurücksetzen** wiederherstellen.

Siehe auch:

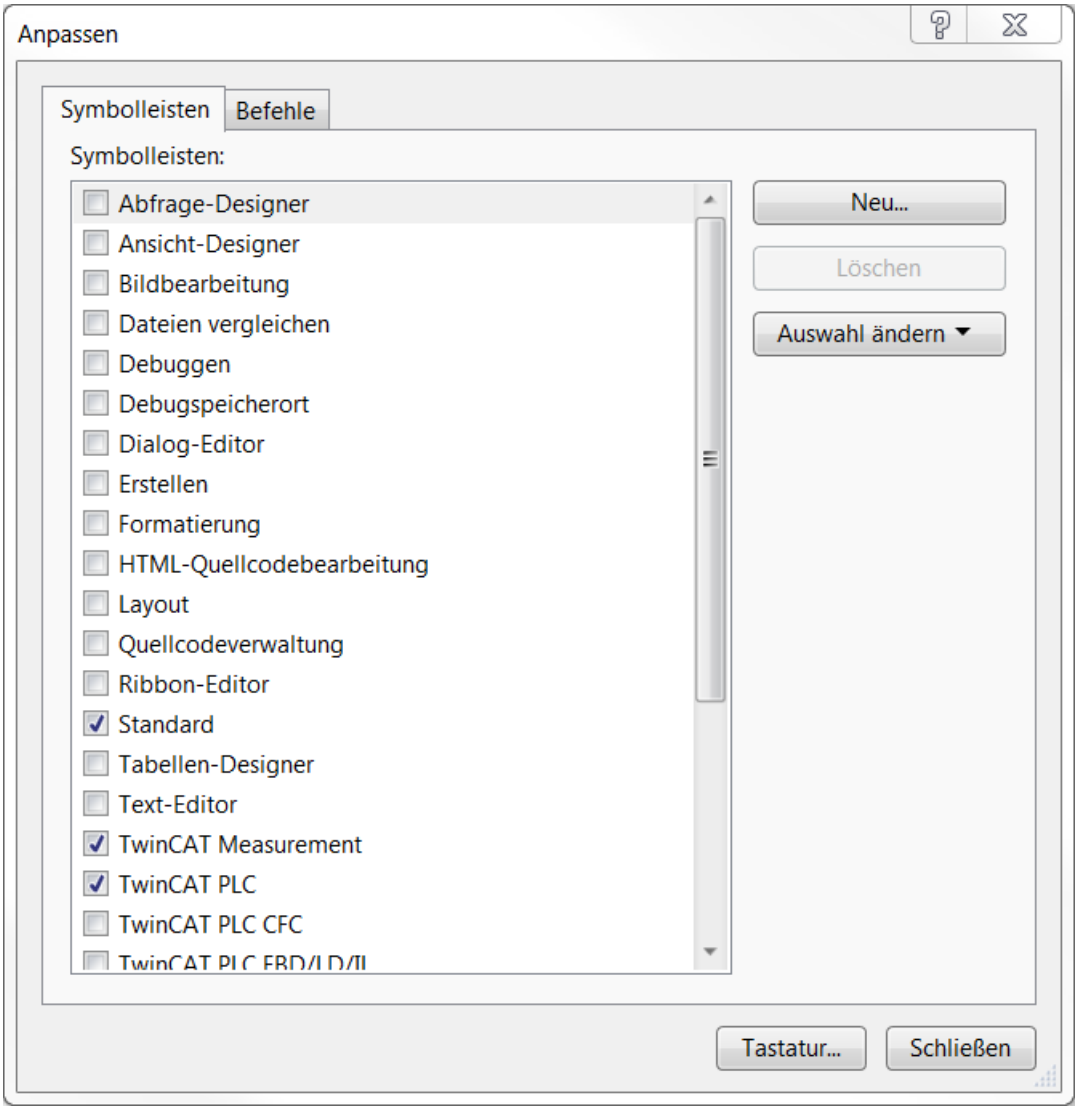
- Dokumentation TC3 User Interface > Menüs anpassen
- Dokumentation TC3 User Interface > Symbolleisten anpassen
- Dokumentation TC3 User Interface > Tastaturkürzel anpassen

17.9.2.1 Dialog Anpassen – Registerkarte Symbolleisten

Funktion: Mit dem Dialog erzeugen Sie neue Symbolleisten, oder Sie passen bestehende Symbolleisen an.

Aufruf: Menü **Extras > Anpassen**

Wenn Sie den Dialog mit **Schließen** beenden, werden die Änderungen in der Menüleiste der TwinCAT-Benutzeroberfläche sichtbar.



Neu… (Symbolleiste hinzufügen)	TwinCAT fügt oberhalb der ausgewählten Symbolleiste eine Symbolleiste hinzu. Ein Dialog öffnet, indem ein Name eingeben werden kann.
Löschen (Symbolleiste entfernen)	TwinCAT entfernt die ausgewählte Symbolleiste. Sie können nur selbst erstellte Symbolleisten entfernen.
Auswahl ändern (Symbolleiste positionieren)	TwinCAT positioniert die ausgewählte Symbolleiste am oberen, unteren, linken oder rechten Rahmen des Hauptfensters
Tastatur…	Öffnet den Dialog **Optionen**, in dem Tastaturkürzel definiert werden können.

Symbolleisten

Darstellung der aktuell definierten Symbolleisten.	
☐ (Ausblenden)	Blendet die ausgewählte Symbolleiste auf der Benutzeroberfläche aus.
☑ (Einblenden)	Blendet die ausgewählte ausgeblendete Symbolleiste in der TwinCAT-Benutzeroberfläche ein.

Siehe auch:

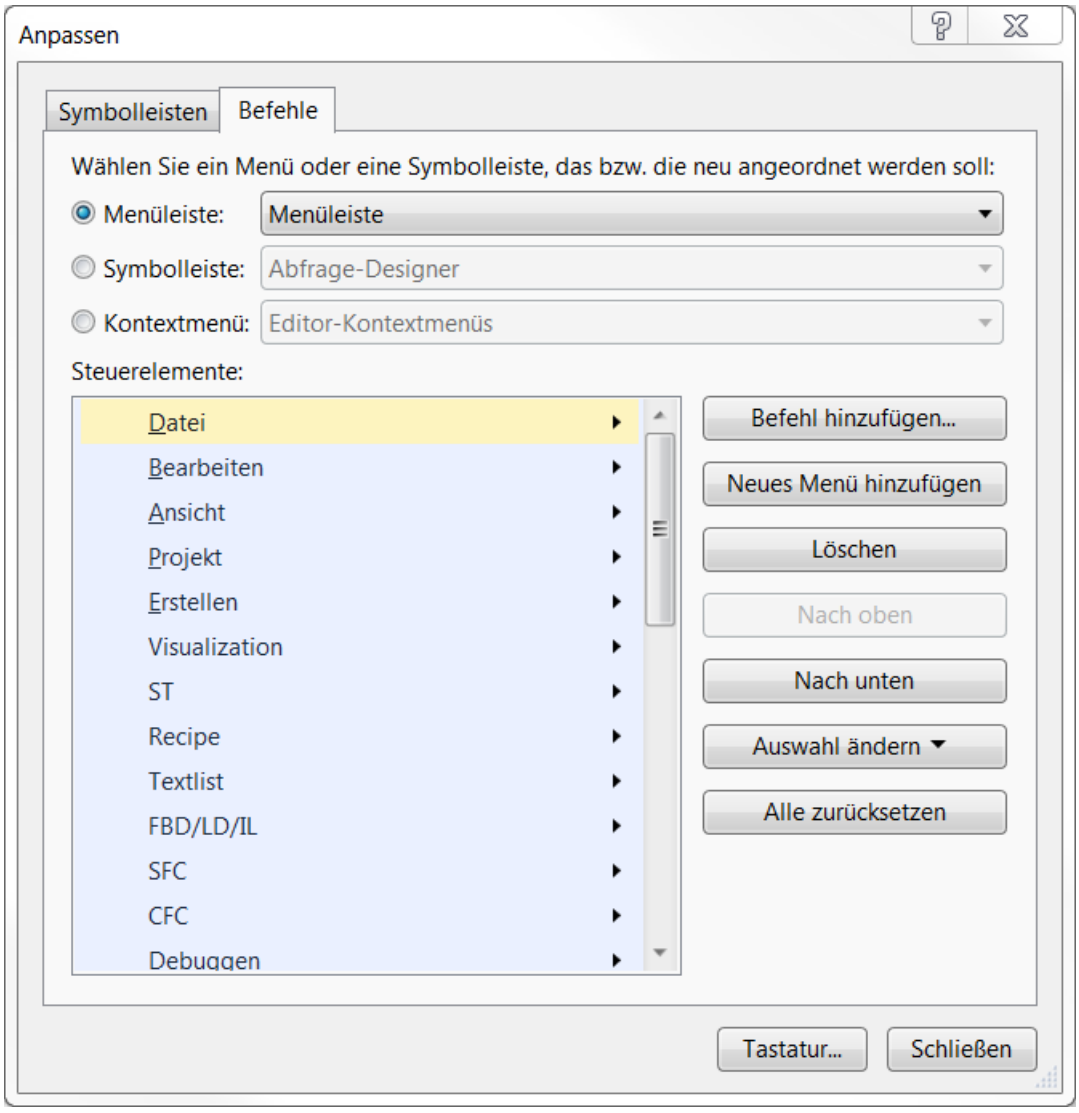
- Dokumentation TC3 User Interface > Symbolleisten anpassen

17.9.2.2 Dialog Anpassen – Registerkarte Befehle

Funktion: Mit dem Dialog definieren Sie Befehle sowie die Struktur und den Inhalt der Menüs und Symbolleisten für die Benutzeroberfläche.

Aufruf: Menü **Extras > Anpassen**

Wenn Sie den Dialog mit **Schließen** beenden, werden die Änderungen in der Menüleiste der TwinCAT-Benutzeroberfläche sichtbar.



Menüs und Symbolleisten

Darstellung der aktuell definierten Symbolleisten, Menüs, Untermenüs und der enthaltenen Befehle.	
Menüleiste	Liste der Menüs und Untermenüs
Symbolleiste	Liste der Symbolleisten
Kontextmenü	Liste der Kontextmenüs

Steuerelemente

Steuerelemente	Auflistung der in dem ausgewählten Menü bzw. der Symbolleiste enthaltenen Befehle bzw. Untermenüs. Die Anordnung von oben nach unten entspricht der später im TwinCAT-Menü bzw. in der Symbolleiste dargestellten Anordnung.

Befehl hinzufügen	Öffnet den Dialog **Befehl hinzufügen**. Der Dialog **Befehl hinzufügen** dient zur Auswahl eines oder mehrerer Befehle. Linker Teil: Auflistung der Kategorien. Rechter Teil: Auflistung der Befehle der selektierten Kategorie. Fügt oberhalb des ausgewählten Befehls einen Befehl hinzu.
Neues Menü hinzufügen	Fügt oberhalb des ausgewählten Menüs ein neues Menü hinzu.
Auswahl ändern	Öffnet ein Menü, in dem der Name eines neu hinzugefügten Menüs bestimmt werden kann.
Löschen	Entfernt das ausgewählte Menü bzw. den ausgewählten Befehl.
Nach oben	Bewegt den ausgewählten Befehl bzw. das ausgewählte Menü in der Reihenfolge der Befehle bzw. der Menüs nach oben.
Nach unten	Bewegt den ausgewählten Befehl bzw. das ausgewählte Menü in der Reihenfolge der Befehle bzw. der Menüs nach unten.
Alle Zurücksetzen	Setzt das gesamte Menü auf die Standardeinstellungen zurück.
Tastatur	Öffnet den Dialog **Optionen**, in dem Tastaturkürzel definiert werden können.

17.10 Fenster

17.10.1 Befehl Verankerung aufheben

Funktion: Der Befehl löst eine Ansicht oder ein Fenster, die/das am Rahmen der Benutzeroberfläche angedockt (fixiert) ist, vom Rahmen und platziert es als unverankertes Fenster auf dem Bildschirm.

Aufruf: Menü **Fenster**, Kontextmenü oder Schaltfläche der Kopfleiste der Ansicht oder der Registerkarte (Fenster)

Die Ansicht kann dann auch außerhalb der Benutzeroberfläche platziert werden. Um eine unverankerte Ansicht wieder an den Rahmen der Benutzeroberfläche zu binden, verwenden Sie den Befehl **Andocken**.

Siehe auch:

- Befehl Andocken [▶ 1044]
- Dokumentation TC3 User Interface: Ansichten und Fenster anordnen

17.10.2 Befehl Andocken

Funktion: Der Befehl „dockt“ eine Ansicht, die vorher mit dem Befehl **Verankerung aufheben** gelöst wurde und nun als unverankerte Ansicht auf dem Bildschirm liegt, wieder an den Rahmen der Benutzeroberfläche „an“.

Aufruf: Menü **Fenster**, Kontextmenü oder Schaltfläche der Kopfleiste der Ansicht oder der Registerkarte (Fenster)

Siehe auch:

- Befehl Verankerung aufheben [▶ 1044]
- Dokumentation TC3 User Interface: Ansichten und Fenster anordnen

17.10.3 Befehl Ausblenden

Symbol: 

Funktion: Der Befehl blendet eine Ansicht aus.

Aufruf: Menü **Fenster**, Kontextmenü oder Schaltfläche der Kopfleiste der Ansicht

Voraussetzung: Eine Ansicht ist aktiviert.

Ausblenden bedeutet, dass die Ansicht geschlossen wird. Der Befehl entspricht so dem Schließen einer Ansicht über die Schaltfläche  in der Kopfleiste der Ansicht. Um eine ausgeblendete Ansicht wieder einzublenden bzw. zu öffnen, verwenden Sie die Befehle im Menü **Ansicht**.

Siehe auch:

- Ansicht [▶ 930]
- Dokumentation TC3 User Interface: Ansichten aus-/einblenden

17.10.4 Befehl Alle automatisch ausblenden

Funktion: Der Befehl blendet alle Ansichten aus.

Aufruf: Menü **Fenster**, Kontextmenü oder Schaltfläche der Kopfleiste der Ansicht

Ausblenden bedeutet, dass TwinCAT alle Ansichten nur mehr als ein Reiter im Rahmen der Benutzeroberfläche anzeigt und es nur sichtbar werden, wenn Sie mit der Maus auf die Reiter klicken. Wenn Sie dann die Schaltfläche  der Kopfleiste der Ansicht anklicken oder den Befehl **Andocken** wählen, wird die Ansicht wieder auf der Benutzeroberfläche verankert.

Siehe auch:

- Befehl Andocken [▶ 1044]
- Ansicht [▶ 930]
- Dokumentation TC3 User Interface: Ansichten aus-/einblenden

17.10.5 Befehl Automatisch in den Hintergrund

Symbol: 

Funktion: Der Befehl setzt eine Ansicht in den Hintergrund.

Aufruf: Menü **Fenster**, Kontextmenü oder Schaltfläche der Kopfleiste der Ansicht

Voraussetzung: Eine Ansicht ist aktiviert.

In den Hintergrund setzen bedeutet, dass TwinCAT die Ansicht nur mehr als ein Reiter im Rahmen der Benutzeroberfläche anzeigt und sie nur sichtbar wird, wenn Sie mit der Maus auf den Reiter klicken. Wenn Sie dann die Schaltfläche der Kopfleiste () erneut anklicken oder den Befehl **Andocken** wählen, wird die Ansicht wieder auf der Benutzeroberfläche verankert.

Siehe auch:

- Befehl Andocken [▶ 1044]
- Dokumentation TC3 User Interface: Ansichten aus-/einblenden

17.10.6 Befehl Registerkarte anheften

Symbol: 

Funktion: Der Befehl heftet die gerade aktive Registerkarte am linken Rand des Hauptfensters an.

Aufruf: Menü **Fenster**, Kontextmenü oder Schaltfläche der Kopfleiste der Registerkarte (Fenster)

Voraussetzung: Eine Registerkarte (Fenster) ist aktiviert.

Siehe auch:

- Dokumentation TC3 User Interface: Ansichten aus-/einblenden

17.10.7 Befehl Neue horizontale Registerkartengruppe

Symbol: 

Funktion: Der Befehl verschiebt das gerade aktive Fenster in eine neue, separate Registerkartengruppe unterhalb der bereits existierenden.

Aufruf: Menü **Fenster**, Kontextmenü der Kopfleiste der Registerkarte (Fenster)

Voraussetzung: Mehrere Editorfenster sind als Registerblätter nebeneinander angeordnet.

Wenn Sie ein weiteres Objekt im Editor öffnen, wird dies automatisch in die Registerkartengruppe eingeordnet, in der der Fokus liegt.

Siehe auch:

- Dokumentation TC3 User Interface: Ansichten und Fenster anordnen
- Befehl Neue vertikale Registerkartengruppe [▶ 1046]

17.10.8 Befehl Neue vertikale Registerkartengruppe

Symbol: 

Funktion: Der Befehl verschiebt das gerade aktive Fenster in eine neue, separate Registerkarten-Gruppe rechts von der bereits existierenden.

Aufruf: Menü **Fenster**, Kontextmenü der Kopfleiste der Registerkarte (Fenster)

Voraussetzung: Mehrere Editorfenster sind als Registerblätter nebeneinander angeordnet.

Wenn Sie ein weiteres Objekt im Editor öffnen, wird dies automatisch in die Registerkartengruppe eingeordnet, in der der Fokus liegt.

Siehe auch:

- Dokumentation TC3 User Interface: Ansichten und Fenster anordnen
- Befehl Neue horizontale Registerkartengruppe [▶ 1046]

17.10.9 Befehl Fenster-Layout zurücksetzen

Funktion: Der Befehl setzt alle gerade geöffneten Fenster und Ansichten auf ihre Standardpositionen zurück. Sie müssen den Befehl vor der Ausführung bestätigen.

Aufruf: Menü **Fenster**

17.10.10 Befehl Alle Dokumente schließen

Symbol: 

Funktion: Der Befehl schließt alle gerade geöffneten Editorfenster.

Aufruf: Menü **Fenster**

Voraussetzung: Mindestens ein Editorfenster ist geöffnet.

Siehe auch:

- Dokumentation TC3 User Interface: Ansichten aus-/einblenden

17.10.11 Befehl Fenster

Funktion: Der Befehl öffnet den Dialog **Fenster**, der alle geöffneten Objekte zeigt. Sie können darin Fenster aktivieren oder schließen.

Aufruf: Menü **Fenster**

17.10.12 Befehle des Untermenüs Fenster

Funktion: Der Befehl aktiviert das ausgewählte Fenster.

Aufruf: Menü **Fenster**

Für jedes geöffnete Editorfenster enthält das Menü **Fenster** einen Befehl **<n><Objektname>**, über den Sie das Fenster aktivieren, also den Fokus dorthin setzen. Im Offlinebetrieb ergänzt TwinCAT hinter dem Befehl die Erweiterung (Offline). Bei Funktionsbausteinen wird zur Unterscheidung von Implementierung und Instanz die Erweiterung (Impl) oder <Instanzpfad> hinzugefügt.

Siehe auch:

- Befehl Fenster [▶ 1047]

17.11 AS

17.11.1 Befehl Initialschritt

Symbol: 

Funktion: Der Befehl wandelt den gerade selektierten Schritt in einen Initialschritt um.

Aufruf: Menü **AS**, Kontextmenü

Wenn Sie den Befehl ausführen, verändert sich der Rahmen des Schritt-Elements in eine Doppellinie. Der Schritt, der zuvor Initialschritt war, wird automatisch zu einem „normalen“ Schritt und wird mit einfachem Rahmen dargestellt.

Sie können die Eigenschaft **Initialschritt** auch in der Ansicht **Eigenschaften** eines Schritt-Elementes aktivieren oder deaktivieren, dabei passt TwinCAT jedoch die Einstellungen der anderen Schritte nicht automatisch an.

Der Befehl kann nützlich sein, wenn Sie das Diagramm umstellen wollen. Wenn Sie ein neues AS-Objekt anlegen, enthält es automatisch einen Initialschritt, gefolgt von einer Transition (TRUE) und einem Sprung zurück zum Initialschritt.

Beachten Sie die Möglichkeit, im Onlinebetrieb das Diagramm mithilfe der AS-Flags SFCInit bzw. SFCReset auf den Initialschritt zurückzusetzen.

Siehe auch:

- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Elementeigenschaften [▶ 683]

17.11.2 Befehl Schritt-Transition einfügen

Symbol: 

Funktion: Der Befehl fügt einen Schritt und eine Transition vor der gerade selektierten Position ein.

Aufruf: Menü **AS**, Kontextmenü

Wenn Sie einen Schritt selektiert haben, fügt TwinCAT eine neue Schritt-Transition-Kombination ein. Wenn Sie eine Transition selektiert haben, wird eine neue Transition-Schritt-Kombination eingefügt.

Der neue Schritt wird per Default Step<n> genannt. n ist eine fortlaufende Zahl, beginnend mit 0 für den ersten Schritt, der zusätzlich zum Initialschritt eingefügt wird. Die neue Transition wird entsprechend per Default Trans<n> genannt. Sie können die Default-Namen direkt durch Mausklick auf den Namen editieren.

Siehe auch:

- Befehl Schritt-Transition danach einfügen [▶ 1048]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Elemente Schritt und Transition [▶ 677]

17.11.3 Befehl Schritt-Transition danach einfügen

Symbol: 

Funktion: Der Befehl fügt einen Schritt und eine Transition nach der gerade angewählten Position ein.

Aufruf: Menü **AS**, Kontextmenü

Wenn Sie einen Schritt selektiert haben, fügt TwinCAT eine neue Transition-Schritt-Kombination ein. Wenn Sie eine Transition selektiert haben, wird eine neue Schritt-Transition-Kombination eingefügt.

Der neue Schritt wird standardmäßig Step<n> genannt. n ist eine fortlaufende Zahl, beginnend mit 0 für den ersten Schritt, der zusätzlich zum Initialschritt eingefügt wird. Die neue Transition wird entsprechend per Default Trans<n> genannt. Sie können die Default-Namen direkt nach Mausklick auf den Namen editieren.

Siehe auch:

- Befehl Schritt-Transition einfügen [▶ 1047]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Elemente 'Schritt' und 'Transition'

17.11.4 Befehl Parallel

Symbol: 

Funktion: Der Befehl wandelt die gerade selektierte alternative Verzweigung in eine parallele Verzweigung um.

Aufruf: Menü **AS**, Kontextmenü

Voraussetzung: Die horizontale Verbindungslinie einer Verzweigung ist selektiert.

Beachten Sie, dass Sie nach der Umwandlung einer Verzweigung die Anordnung der Schritte und Transitionen vor und nach der Verzweigung prüfen und anpassen müssen.

Siehe auch:

- Befehl Alternativ [▶ 1049]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]

17.11.5 Befehl Alternativ

Symbol: 

Funktion: Der Befehl wandelt die gerade selektierte parallele Verzweigung in eine alternative Verzweigung um.

Aufruf: Menü **AS**, Kontextmenü

Voraussetzung: Die horizontale Verbindungslinie einer Verzweigung ist selektiert.

Beachten Sie, dass Sie nach der Umwandlung einer Verzweigung die Anordnung der Schritte und Transitionen vor und nach der Verzweigung prüfen und anpassen müssen.

Siehe auch:

- Befehl Parallel [▶ 1048]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]

17.11.6 Befehl Verzweigung einfügen

Symbol: 

Funktion: Der Befehl fügt eine Verzweigung links von der gerade angewählten Position ein.

Aufruf: Menü **AS**, Kontextmenü

Das Verhalten des Befehls entspricht dem Befehl **Verzweigung rechts einfügen**.

Siehe auch:

- Befehl Verzweigung rechts einfügen [▶ 1049]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Element Verzweigung [▶ 681]

17.11.7 Befehl Verzweigung rechts einfügen

Symbol: 

Funktion: Der Befehl fügt eine Verzweigung rechts von der gerade angewählten Position ein.

Aufruf: Menü **AS**, Kontextmenü

Der Typ der eingefügten Verzweigung hängt von dem ausgewählten Element ab:

- Wenn das oberste Element der gerade ausgewählten Elemente eine Transition oder eine alternative Verzweigung ist, fügt TwinCAT eine alternative Verzweigung ein.
- Wenn das oberste Element der gerade ausgewählten Elemente ein Schritt, ein Makro, ein Sprung oder eine parallele Verzweigung ist, fügt TwinCAT eine parallele Verzweigung mit Sprungmarke Branch<x> ein, wobei x eine fortlaufende Zahl ist. Sie können diesen Default-Name der Sprungmarke editieren. Sie können die Sprungmarke als Ziel eines Sprungs angegeben.
- Wenn gerade ein gemeinsames Element einer bestehenden Verzweigung selektiert ist (horizontale Linie), fügt TwinCAT die neue Verzweigung ganz rechts als weiteren Zweig hinzu. Wenn gerade ein gesamter Zweig einer bestehenden Verzweigung selektiert ist, fügt TwinCAT die neue Verzweigung direkt rechts davon als neuen Zweig hinzu.

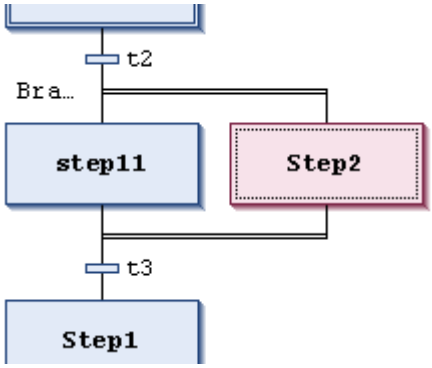
Beachten Sie, dass Sie eine Verzweigung mit den Befehlen **Alternativ** bzw. **Parallel** in den jeweils anderen Typ umwandeln können.

Beispiel: Parallele Verzweigung

In der folgenden Abbildung sehen Sie eine neu eingefügte parallele Verzweigung, erzeugt mit Befehl Verzweigung rechts einfügen, während step11 ausgewählt war. TwinCAT fügt automatisch einen Schritt (Step2 im Beispiel) ein.

Abarbeitung im Onlinebetrieb: Wenn t2 TRUE liefert, führt TwinCAT Step2 sofort nach step11 aus, bevor t3 ausgewertet wird.

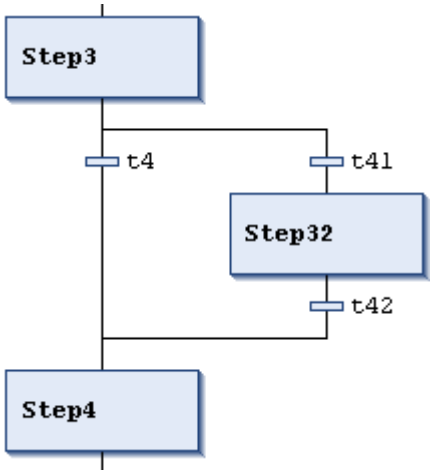
TwinCAT führt also im Gegensatz zu alternativen Verzweigungen beide Zweige aus.



Beispiel: Alternative Verzweigung

In der folgenden Abbildung sehen Sie eine neu eingefügte alternative Verzweigung, erzeugt mit Befehl Verzweigung rechts einfügen während Transition t4 ausgewählt war. TwinCAT fügt automatisch einen Schritt (Step32 im Beispiel), eine vorausgehende und eine nachfolgende Transition (t41, t42) ein.

Abarbeitung im Onlinebetrieb: Wenn Step3 aktiv ist, wertet TwinCAT die nachfolgenden Transitionen (t4, t41) von links nach rechts aus. Der erste Zweig der Verzweigung, in dem die erste Transition TRUE liefert, wird ausgeführt. Somit wird im Gegensatz zu einer parallelen Verzweigung nur ein Zweig ausgeführt.



Siehe auch:

- Befehl Alternativ [▶ 1049]
- Befehl Parallel [▶ 1048]
- Befehl Verzweigung einfügen [▶ 1049]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Element Verzweigung [▶ 681]

17.11.8 Befehl Aktionsassoziation einfügen

Symbol: 

Funktion: Der Befehl weist einem Schritt eine IEC-Aktion zu.

Aufruf: Menü **AS**, Kontextmenü

Voraussetzung: Ein Schritt ist selektiert.

TwinCAT fügt das Aktion-Element rechts neben dem gerade selektierten Schritt-Element ein.

Wenn Sie dem Schritt bereits eine oder mehrere Aktionen zugewiesen haben, werden diese in einer „Aktionenliste“ untereinander dargestellt. Die neue Aktion wird dann folgendermaßen platziert:

- Wenn Sie das Schritt-Element selektiert haben, als erste Aktion des Schrittes, d. h. an oberster Position der Aktionenliste.
- Wenn Sie eine der vorhandenen Aktionen in der Aktionenliste des Schrittes selektiert haben, direkt vor, d. h. oberhalb der Aktion.

Der linke Teil des Aktion-Elements enthält den Qualifizierer, standardmäßig N, im rechten Teil geben Sie den Aktionsnamen ein. Dazu klicken Sie in die Box, um einen Editierrahmen zu erhalten. Sie müssen diese Aktion vorher im Projekt als POU angelegt haben.

Sie können den Qualifizierer ebenfalls editieren. Eine Liste der gültigen Qualifizierer ist im Kapitel „Qualifizierer für Aktionen in AS“ beschrieben.

Siehe auch:

- Befehl Aktionsassoziation danach einfügen [▶ 1051]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: Qualifizierer für Aktionen in AS [▶ 670]

17.11.9 Befehl Aktionsassoziation danach einfügen

Symbol: 

Funktion: Der Befehl weist einem Schritt eine IEC-Aktion zu.

Aufruf: Menü **AS**, Kontextmenü

Voraussetzung: Ein Schritt ist selektiert.

Der Befehl entspricht der Beschreibung von Aktionsassoziation einfügen. Der Unterschied besteht darin, dass TwinCAT die neue Aktion nicht an der ersten, sondern an der letzten Position der Aktionenliste platziert. Wenn Sie in der Aktionsliste eine Aktion selektiert haben, platziert TwinCAT die neue Aktion nicht oberhalb, sondern unterhalb.

Siehe auch:

- Befehl Aktionsassoziation einfügen [▶ 1051]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: Qualifizierer für Aktionen in AS [▶ 670]

17.11.10 Befehl Sprung einfügen

Symbol: 

Funktion: Der Befehl fügt ein Sprung-Element vor dem gerade selektierten Element ein.

Aufruf: Menü **AS**, Kontextmenü

Voraussetzung: Ein Schritt ist selektiert.

TwinCAT fügt den Sprung automatisch mit Sprungziel Step ein. Sie müssen dieses Sprungziel anschließend noch durch ein reales Sprungziel ersetzten. Sie können das Ziel mit der Eingabehilfe auswählen.

Siehe auch:

- Befehl Sprung danach einfügen [▶ 1052]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Element Sprung [▶ 682]

17.11.11 Befehl Sprung danach einfügen

Symbol: 

Funktion: Der Befehl fügt ein Sprung-Element nach dem gerade selektierten Element ein.

Aufruf: Menü **AS**

TwinCAT fügt den Sprung automatisch mit Sprungziel Step ein. Sie müssen dieses Sprungziel anschließend noch durch ein reales Sprungziel ersetzten. Sie können das Ziel mit der Eingabehilfe auswählen.

Siehe auch:

- Befehl Sprung einfügen [▶ 1051]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Element Sprung [▶ 682]

17.11.12 Befehl Makro einfügen

Symbol: 

Funktion: Der Befehl fügt ein Makro-Element vor dem gerade selektierten Element ein.

Aufruf: Menü **AS**, Kontextmenü

Das neue Makro wird standardmäßig Macro<x> genannt. x ist eine fortlaufende Zahl, beginnend mit 0 für das erste Makro. Sie können den Default-Name direkt durch Mausklick auf den Namen editieren.

Um das Makro zu bearbeiten, öffnen Sie es mit dem Befehl **Makro anzeigen** im Makro-Editor.

Siehe auch:

- Befehl Makro anzeigen [▶ 1053]
- Befehl Makro danach einfügen [▶ 1052]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- AS-Editor [▶ 667]

17.11.13 Befehl Makro danach einfügen

Symbol: 

Funktion: Der Befehl fügt ein Makro-Element nach dem gerade selektierten Element ein.

Aufruf: Menü **AS**, Kontextmenü

Der Befehl entspricht der Beschreibung von dem Befehl **Makro einfügen**.

Siehe auch:

- Befehl Makro einfügen [▶ 1052]
- Befehl Makro anzeigen [▶ 1053]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]

17.11.14 Befehl Makro anzeigen

Symbol: 

Funktion: Der Befehl öffnet ein Makro im Makro-Editor zur Bearbeitung.

Aufruf: Menü **AS**

Voraussetzung: Ein Makro ist selektiert.

Mit dem Befehl schließt TwinCAT die Hauptansicht des AS-Editors und öffnet stattdessen den Makro-Editor. Es handelt sich ebenfalls um einen AS-Editor, in dem Sie nun den Teil des AS-Diagramms bearbeiten können, der in der Hauptansicht als Makro-Box dargestellt wird.

Mit dem Befehl **Makro verlassen** kehren Sie zur Hauptansicht zurück.

Siehe auch:

- Befehl Makro verlassen [▶ 1053]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]

17.11.15 Befehl Makro verlassen

Symbol: 

Funktion: Der Befehl schließt den Makro-Editor und kehrt zur Hauptansicht des AS-Editors zurück.

Aufruf: Menü **AS**

Voraussetzung: Ein Makro ist im Makro-Editor geöffnet.

Siehe auch:

- Befehl Makro anzeigen [▶ 1053]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]

17.11.16 Befehl Einfügen danach

Symbol: 

Funktion: Der Befehl fügt die Elemente aus der Zwischenablage nach der gerade angewählten Position ein.

Aufruf: Menü **AS**

17.11.17 Befehl Eingangsaktion hinzufügen

Symbol:

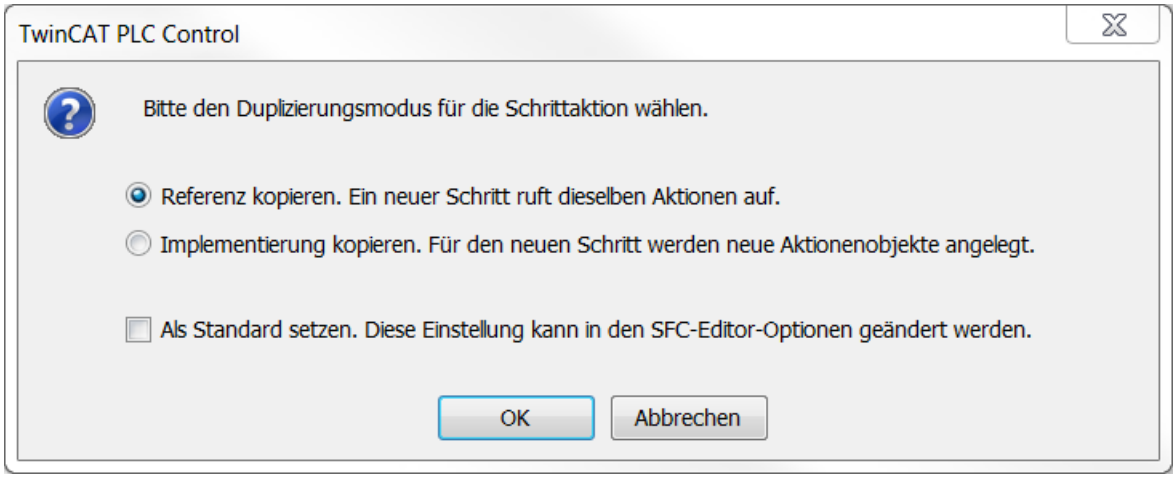
Funktion: Der Befehl führt zum Dialog **Eingangsaktion hinzufügen**, in dem Sie eine neue Schrittaktion des Typs „Eingangsaktion“ definieren. Abhängig von den AS-Optionen erscheint gegebenenfalls vorher noch eine Eingabeaufforderung zum Wählen des Duplizierungsmodus für die neue Schrittaktion.

Aufruf: Menü **AS**, Kontextmenü des ausgewählten Schrittelements

Voraussetzung: Ein Schrittelement ist ausgewählt.

Die Eingangsaktion wird automatisch im ST-Editor geöffnet. Das Schrittelement erhält ein „E“ in der linken unteren Ecke.

Abfragedialog zum Wählen des Duplizierungsmodus



Referenz kopieren. Ein neuer Schritt wird dieselben Aktionen aufrufen	Wenn der Schritt im AS kopiert wird, wird die Verknüpfung zu der/den Schrittaktionen mit kopiert. Die voneinander kopierten Schritte werden also alle dieselben Aktionen aufrufen.
Implementierung kopieren. Für den neuen Schritt werden neue Aktionsobjekte angelegt	Dies bedeutet eine „Einbettung“ der Schrittaktionen für die kopierten Schritte. Die neu erzeugten Aktionsobjekte erscheinen standardmäßig unter dem AS-Baustein im SPS-Projektbaum im Projektmappen-Explorer. Diese Objekte enthalten zunächst eine Kopie des ursprünglichen Implementierungscodes der jeweiligen Aktion.
Als Standard setzen. Diese Einstellung kann in den SFC-Editor-Optionen geändert werden.	Die Einstellungen im Dialog werden als Standardeinstellung übernommen. Sie können die Standardeinstellung in den TwinCAT-Optionen in der Kategorie **AS-Editor** ändern. Wählen Sie dazu im Gruppenfeld **Schrittaktionen** in der Drop-down-Liste **Standardeinfügemethode** den Eintrag **Immer fragen**, **Referenz kopieren** oder **Implementierung duplizieren.**

Siehe auch:

- Befehl Optionen > Dialog Optionen - AS-Editor [▶ 1022]
- Dokumentation PLC: Befehl Ausgangsaktion hinzufügen [▶ 1055]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: Programmieren in Ablaufsprache (AS) [▶ 129]

- Dokumentation PLC: AS-Element Aktion [▶ 678]

17.11.18 Befehl Ausgangsaktion hinzufügen

Symbol: 

Funktion: Der Befehl führt zum Dialog **Ausgangsaktion hinzufügen**, in dem Sie eine neue Schrittaktion des Typs Eingangsaktion definieren. Abhängig von den AS-Optionen erscheint gegebenenfalls vorher noch eine Eingabeaufforderung zum Wählen des Duplizierungsmodus für die neue Schrittaktion. Sehen Sie hierzu die Hilfeseite zum Befehl **Eingangsaktion hinzufügen**.

Aufruf: Menü **AS**, Kontextmenü des ausgewählten Schrittelements

Voraussetzung: Ein Schrittelement im AS ist ausgewählt.

Siehe auch:

- Befehl Eingangsaktion hinzufügen [▶ 1054]
- Befehl Optionen > Dialog Optionen - AS-Editor [▶ 1022]
- Dokumentation PLC: Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: Programmieren in Ablaufsprache (AS) [▶ 129]
- Dokumentation PLC: AS-Editor [▶ 667]
- Dokumentation PLC: AS-Element Aktion [▶ 678]

17.11.19 Befehl Change duplication - Setzen

Funktion: Der Befehl verknüpft jede Schrittaktion oder Transition, die von einem Schritt oder einer Transition im AS-Baustein aufgerufen wird, fest mit dem Aufrufer. Somit kann das Aktions- oder Transitionsobjekt nur noch von genau diesem einen Aufrufer aufgerufen werden (Pseudoeinbettung). Dies hat zur Folge, dass das Kopieren von Schritt- und Transitionselementen, die Aktionen oder Transitionen aufrufen, automatisch neue Aktions- oder Transitionsobjekte erzeugt. Der Implementierungscode wird jeweils kopiert.

Aufruf: Menü **AS**

Sehen Sie für Details zum Duplizierungsmodus die Hilfeseite zu den AS-Elementeigenschaften und die Anleitung zum Hinzufügen von Schrittaktionen.

Siehe auch:

- Dokumentation PLC: AS-Elementeigenschaften [▶ 683]
- Dokumentation PLC: Programmieren in Ablaufsprache (AS) [▶ 129]

17.11.20 Befehl Change duplication - Entfernen

Funktion: Der Befehl hebt die feste Verknüpfung von Aktions-, oder Transitionsobjekten mit dem Schritt oder der Transition, der/die sie aufruft, für den gesamten AS-Baustein auf. Damit wird die Pseudoeinbettung der Aktions- oder Transitionsobjekte aufgehoben. Wenn dann Schritt- und Transitionselemente, die Aktionen oder Transitionen aufrufen, kopiert werden, rufen die Kopien dieselben Aktionen und Transitionen auf wie die Quelle.

Aufruf: Menü **AS**

Sehen Sie für Details zum Duplizierungsmodus die Hilfeseite zu den AS-Elementeigenschaften und die Anleitung zum Hinzufügen von Schrittaktionen.

Siehe auch:

- Dokumentation PLC: AS-Elementeigenschaften [▶ 683]
- Dokumentation PLC: Programmieren in Ablaufsprache (AS) [▶ 129]

17.11.21 Befehl Schritt einfügen

Der Befehl ist standardmäßig nicht im Menü **AS** enthalten.

Symbol:

Funktion: Der Befehl fügt einen Schritt vor der gerade selektierten Position ein.

Aufruf: Menü **AS** , Kontextmenü im AS-Editor

Der neue Schritt wird standardmäßig Step<n> genannt. n ist eine fortlaufende Zahl, beginnend mit 0 für den ersten Schritt, der zusätzlich zum Initialschritt eingefügt wird. Der Name wird nach Mausklick auf den Namen editierbar.

Wenn Sie einen Schritt ohne Transition oder eine Transition ohne Schritt einfügen, führt dies beim Übersetzen zu einem Fehler.

Siehe auch:

- Befehl Schritt-Transition danach einfügen [▶ 1048]
- Befehl Initialschritt [▶ 1047]
- Dokumentation PLC: AS-Elemente Schritt und Transition [▶ 677]

17.11.22 Befehl Schritt danach einfügen

Der Befehl ist standardmäßig nicht im Menü **AS** enthalten.

Symbol:

Funktion: Der Befehl fügt einen Schritt nach der gerade selektierten Position ein.

Aufruf: Menü **AS**, Kontextmenü im AS-Editor

Der neue Schritt wird standardmäßig Step<n> genannt. n ist eine fortlaufende Zahl, beginnend mit 0 für den ersten Schritt, der zusätzlich zum Initialschritt eingefügt wird. Der Name wird nach Mausklick auf den Namen editierbar.

Wenn Sie einen Schritt ohne Transition oder eine Transition ohne Schritt einfügen, führt dies beim Übersetzen zu einem Fehler.

Siehe auch:

- Befehl Initialschritt [▶ 1047]
- Befehl Schritt-Transition danach einfügen [▶ 1048]
- Dokumentation PLC: AS-Element Sprung [▶ 682]

17.11.23 Befehl Transition einfügen

Der Befehl ist standardmäßig nicht im Menü **AS** enthalten.

Symbol:

Funktion: Der Befehl fügt eine Transition vor der gerade selektierten Position ein.

Aufruf: Menü **AS** , Kontextmenü im AS-Editor

Die neue Transition wird standardmäßig Trans<n> genannt. n ist eine fortlaufende Zahl, beginnend mit 0 für die erste Transition. Der Name wird nach Mausklick auf den Namen editierbar.

Wenn Sie einen Schritt ohne Transition oder eine Transition ohne Schritt einfügen, führt dies beim Übersetzen zu einem Fehler.

Siehe auch:

- Befehl Schritt-Transition danach einfügen [▶ 1048]
- Dokumentation PLC: AS-Elemente Schritt und Transition [▶ 677]

17.11.24 Befehl Transition danach einfügen

Der Befehl ist standardmäßig nicht im Menü **AS** enthalten.

Symbol: 

Funktion: Der Befehl fügt eine Transition nach der gerade selektierten Position ein.

Aufruf: Menü **AS**, Kontextmenü im AS-Editor

Die neue Transition wird standardmäßig Trans<n> genannt. n ist eine fortlaufende Zahl, beginnend mit 0 für die erste Transition. Der Name wird nach Mausklick auf den Namen editierbar.

Wenn Sie einen Schritt ohne Transition oder eine Transition ohne Schritt einfügen, führt dies beim Übersetzen zu einem Fehler.

Siehe auch:

- Befehl Schritt danach einfügen [▶ 1056]
- Dokumentation PLC: AS-Elemente Schritt und Transition [▶ 677]

17.12 CFC

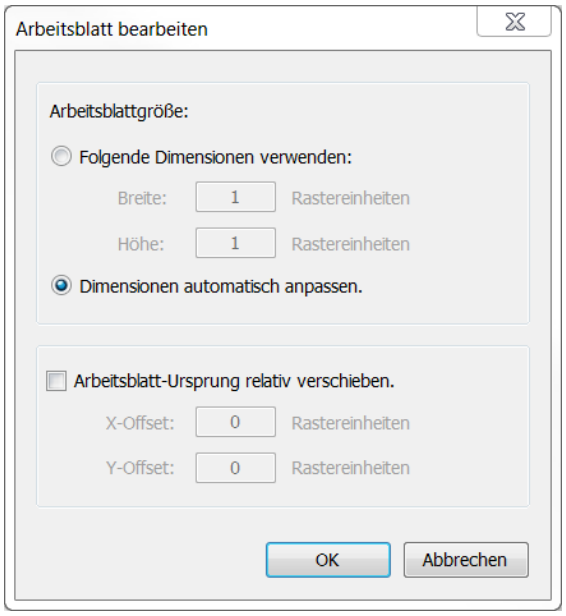
17.12.1 Befehl Arbeitsblatt bearbeiten

Funktion: Der Befehl öffnet den Dialog **Arbeitsblatt bearbeiten**, in dem Sie die Größe des Arbeitsblattes festlegen.

Aufruf: Menü **CFC**

Voraussetzungen: Ein CFC-Editor ist aktiv.

Dialog Arbeitsblatt bearbeiten



Folgende Dimensionen verwenden	Hier stellen Sie die Größe des Arbeitsblatts ein. Ihre Änderung wird nur übernommen, wenn die Größe für das bestehende Programm ausreichend ist.
Dimensionen automatisch anpassen	Passt die Größe des Arbeitsblatts automatisch an die Größe Ihres Programms an.
Arbeitsblatt Ursprung relativ verschieben	Verschiebt das Arbeitsblatt auf der x- oder y-Achse. Die Eingabe negativer Zahlen ist erlaubt.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.2 Befehl Seitengröße bearbeiten

Funktion: Der Befehl öffnet den Dialog Seitengröße bearbeiten. Darin verändern Sie die Größe des seitenorientierten CFC-Editors.

Aufruf: Menü **CFC**

Voraussetzungen: Ein seitenorientierter CFC-Editor ist aktiv.

Dialog Seitengröße bearbeiten

Breite	Breite der Seite (Minimum 24, Maximum 1024). Elemente außerhalb des Arbeitsbereiches werden rot markiert.
Höhe	Höhe der Seite (Minimum 24, Maximum 1024). Elemente außerhalb des Arbeitsbereichs werden rot markiert.
Randbreite	Breite der Randspalte (Minimum 6, Maximum 25% oder Seitenbreite).
Als Standard für neue CFC-Objekte setzen	: Die aktuellen Einstellungen werden als Standard für neue CFC-Objekte gewählt.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: CFC-Element Seite [▶ 711]

17.12.3 Befehl Negieren

Symbol: 

Funktion: Der Befehl negiert den ausgewählten Bausteineingang oder Bausteinausgang.

Aufruf: Menü **CFC**, Kontextmenü

Voraussetzungen: Ein CFC-Editor ist aktiv. Ein Bausteineingang oder Bausteinausgang ist selektiert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.4 Befehl EN/ENO

Symbol: 

Funktion: Der Befehl fügt dem selektierten Baustein einen booleschen Eingang EN (Enable) und einen booleschen Ausgang ENO (Enable Out) hinzu.

Aufruf: Menü **CFC**, Kontextmenü

Voraussetzungen: Ein CFC-Editor ist aktiv. Ein Baustein ist selektiert.

Der hinzugefügte Eingang „EN“ aktiviert den Baustein. Nur wenn er den Wert TRUE hat, wird der Baustein ausgeführt. Der Wert dieses Signals wird am Ausgang ENO ausgegeben.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.5 Befehl Kein

Symbol: 

Funktion: Der Befehl entfernt einen Reset (R), einen Set (S) oder ein REF vom Eingang des Elements „Ausgang“.

Aufruf: Menü **CFC > Set/Reset**, Kontextmenü > Set/Reset

Voraussetzungen: Ein CFC-Editor ist aktiv. Der Eingang eines Elements **Ausgang** ist selektiert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.6 Befehl R (Reset)

Symbol: 

Funktion: Der Befehl fügt dem Eingang eines booleschen Elements **Ausgang** einen Reset hinzu.

Aufruf: Menü **CFC > Set/Reset**, Kontextmenü > Set/Reset

Voraussetzungen: Ein CFC-Editor ist aktiv. Der Eingang eines Elements „Ausgang“ ist selektiert.

Wenn ein Element **Ausgang** einen Reset-Eingang hat, wird der boolesche Ausgangswert auf FALSE gesetzt, sobald der Wert des Eingangs TRUE ist. Der Wert FALSE am Ausgang bleibt erhalten, auch wenn sich der Eingangswert wieder ändert.

Siehe auch:

- Befehl S (Set) [▶ 1060]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.7 Befehl S (Set)

Symbol: 

Funktion: Der Befehl fügt dem Eingang eines booleschen Elements „Ausgang“ einen Set (S) hinzu.

Aufruf: Menü **CFC > Set/Reset**, Kontextmenü > Set/Reset

Voraussetzungen: Ein CFC-Editor ist aktiv. Der Eingang eines Elements **Ausgang** ist selektiert.

Wenn ein Element **Ausgang** einen Set-Eingang hat, wird der boolesche Ausgangswert auf TRUE gesetzt, sobald der Wert des Eingangs TRUE ist. Der Wert TRUE am Ausgang bleibt erhalten, auch wenn sich der Eingangswert wieder ändert.

Siehe auch:

- Befehl R (Reset) [▶ 1059]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.8 Befehl REF= (Reference-Zuweisung)

Symbol: 

Funktion: Der Befehl weist einem Element „Ausgang“ eine Referenz zu.

Aufruf: Menü **CFC > Set/Reset**, Kontextmenü > Set/Reset

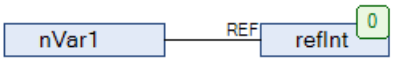
Voraussetzungen: Ein CFC-Edtior ist aktiv. Der Eingang eines Elements **Ausgang** ist selektiert.

Beispiel:

Deklaration:

```
refInt : REFERENCE TO INT;
nVar1 : INT;
```

CFC:



Dies entspricht dem ST-Code

```
refInt REF= nVar1;
```

Weitere Informationen finden Sie in der Beschreibung zum Datentyps REFERENCE TO.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Reference [▶ 807]

17.12.9 Befehl Ausführungsreihenfolge anzeigen

Symbol: 

Funktion: Der Befehl blendet für alle CFC-Elemente des Programmierobjekts kurzzeitig eine Marke mit Nummer ein.

Aufruf:

- Menü **CFC > Ausführungsreihenfolge**
- Kontextmenü im CFC-Editor > **Ausführungsreihenfolge**

Voraussetzungen: Ein CFC-Editor ist aktiv und der Automatische Datenfluss-Modus ist aktiviert.

Die Nummern geben die automatisch ermittelte Ausführungsreihenfolge wieder. Die Ausführungsreihenfolge wird nach Datenfluss und bei mehreren Netzwerken nach deren topologischer Position im Editor ermittelt.

Sobald Sie in den CFC-Editor klicken, werden die Marken ausgeblendet.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.10 Befehl Startpunkt der Rückkopplung setzen

Symbol: 

Funktion: Der Befehl definiert das selektierte Element als Startpunkt innerhalb einer Rückkopplung.

Aufruf:

- Menü **CFC > Ausführungsreihenfolge**
- Kontextmenü im CFC-Editor > **Ausführungsreihenfolge**

Voraussetzungen: Ein CFC-Editor ist aktiv und der Automatische Datenfluss-Modus ist aktiviert. Ein Netzwerk des CFC-Programmierbausteins einhält eine Rückkopplung und ein Element innerhalb dieser Rückkopplung ist selektiert.

Im CFC-Editor ist der Startpunkt innerhalb von Rückkopplungen mit dem Symbol  dekoriert. Dieses Element hat die niedrigste Nummer der Ausführungsreihenfolge innerhalb der Rückkopplungen. Zur Laufzeit beginnt die Abarbeitung der Rückkopplung mit diesem Element.

Siehe auch:

- Befehl Ausführungsreihenfolge anzeigen [▶ 1061]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.11 Befehl An den Anfang

Symbol: 

Funktion: Der Befehl nummeriert die Elemente so, dass die selektierten Elemente am Anfang der Ausführungsreihenfolge stehen.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Mindestens ein Element ist selektiert.

Die selektierten Elemente erhalten die niedrigsten Nummern beginnend mit 0 unter Beibehaltung der bisherigen Reihenfolge. Die restlichen Elemente werden so nummeriert, dass ihre Ausführungsreihenfolge bestehen bleibt. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.12 Befehl Ans Ende

Symbol: 

Funktion: Der Befehl nummeriert die Elemente so, dass die selektierten Elemente am Ende der Ausführungsreihenfolge stehen.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Mindestens ein Element ist selektiert.

Die selektierten Elemente erhalten die höchsten Nummern unter Beibehaltung der bisherigen Reihenfolge. Die restlichen Elemente werden so nummeriert, dass ihre Ausführungsreihenfolge bestehen bleibt. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.13 Befehl Eins vor

Symbol: 

Funktion: Der Befehl nummeriert die Elemente so, dass die selektierten Elemente um eins weiter vorne stehen.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Mindestens ein Element ist selektiert.

Die selektierten Elemente erhalten eine um eins niedrigere Nummerierung unter Beibehaltung der bisherigen Reihenfolge, sodass sie eine Ausführungsposition früher abgearbeitet werden. Die restlichen Elemente werden so nummeriert, dass ihre Ausführungsreihenfolge bestehen bleibt. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.14 Befehl Eins zurück

Symbol: 

Funktion: Der Befehl nummeriert die Elemente so, dass die selektierten Elemente um eins weiter hinten stehen.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Mindestens ein Element ist selektiert.

Die selektierten Elemente erhalten eine um eins höhere Nummerierung unter Beibehaltung der bisherigen Reihenfolge, sodass sie eine Ausführungsposition später abgearbeitet werden. Die restlichen Elemente werden so nummeriert, dass ihre Ausführungsreihenfolge bestehen bleibt. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.15 Befehl Ausführungsreihenfolge setzen

Funktion: Der Befehl öffnet einen Dialog zum Setzen der Nummer des selektierten Elementes auf einen beliebigen Wert.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Genau ein Element ist selektiert.

Das selektierte Element erhält die im Dialog angegebene Nummer. Die restlichen Elemente werden so nummeriert, dass ihre Ausführungsreihenfolge bestehen bleibt. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Befehl Nach Datenfluss anordnen [▶ 1063]
- Befehl Topologisch anordnen [▶ 1064]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.16 Befehl Nach Datenfluss anordnen

Funktion: Der Befehl nummeriert die Elemente im Programm nach Datenfluss und bei mehreren Netzwerken nach ihrer topologischen Position im Editor.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert.

Die Ausführungsreihenfolge wird nach dem Datenfluss und (bei mehreren Netzwerken) nach der topologischen Position der Netzwerke angeordnet. Alle nummerierten Elemente des Programmierbausteins werden entsprechend gesetzt. Die Ausführungsreihenfolge ist danach identisch zu der im automatischen Datenfluss-Modus. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Befehl Topologisch anordnen [▶ 1064]
- Befehl Ausführungsreihenfolge setzen [▶ 1063]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.17 Befehl Topologisch anordnen

Funktion: Der Befehl ordnet die Ausführungsreihenfolge der Elemente nach ihrer topologischen Position von rechts nach links und oben nach unten an.

Aufruf: Menü **CFC > Ausführungsreihenfolge**, Kontextmenü > Ausführungsreihenfolge

Voraussetzungen: Ein CFC-Editor ist aktiv und der Explizite Ausführungsreihenfolge-Modus ist aktiviert. Mindestens ein Element ist selektiert.

Der Befehl wirkt sich auf alle Elemente im Programm aus, auch wenn nicht alle Elemente beim Ausführen des Befehls selektiert sind. Die topologischen Positionen der Elemente werden nicht geändert.

Siehe auch:

- Befehl Nach Datenfluss anordnen [▶ 1063]
- Befehl Ausführungsreihenfolge setzen [▶ 1063]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: Automatische Ausführungsreihenfolge nach Datenfluss [▶ 125]
- Befehl Eigenschaften (Objekt) [▶ 946]

17.12.18 Befehl Selektierte Anschlüsse verbinden

Symbol: 

Funktion: Der Befehl stellt eine Verbindung zwischen den selektierten Anschlüssen her.

Aufruf: Menü **CFC**, Kontextmenü

Voraussetzungen: Ein CFC-Editor ist aktiv. Genau ein Ausgang und mehrere Eingänge sind selektiert.

Zum Selektieren der Anschlüsse müssen Sie die **[STRG]**-Taste gedrückt halten, während Sie die Anschlüsse anklicken. Danach führen Sie den Befehl aus.

Siehe auch:

- Befehl Verbundene Anschlüsse selektieren [▶ 1065]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.19 Befehl Verbindung lösen

Symbol: 

Funktion: Dieser Befehl löst eine gesperrte Verbindung.

Aufruf: Menü **CFC > Routing**, Kontextmenü > Routing

Voraussetzungen: Ein CFC-Editor ist aktiv. Eine Verbindung oder eine Verbindungsmarke ist selektiert.

Eine gesperrte Verbindung erhalten Sie, wenn Sie die Verbindungen des automatischen Routings verändern. Wenn Sie erneut ein automatisches Routing durchführen wollen, müssen Sie vorher eine gesperrte Verbindung lösen.

Mit einem Mausklick auf das Icon einer gesperrten Verbindung können Sie diese Verbindung ebenfalls lösen.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: CFC-Element Verbindungsmarke-Quelle/Ziel [▶ 714]

17.12.20 Befehl Nächste Kollision zeigen

Funktion: Der Befehl zeigt die nächste Kollision im Editor auf und markiert die betroffene Stelle.

Aufruf: Schaltfläche  in der rechten oberen Ecke des Editors

Voraussetzungen: Ein CFC-Editor ist aktiv und es ist mindestens eine Verbindung mit Kollision vorhanden.

Diese Funktion ist sehr nützlich, wenn Sie mit großen Netzwerken arbeiten und nur einen Teilbereich sehen. Eine Kollision wird Ihnen zusätzlich durch das rot umrandete Symbol in der rechten oberen Ecke des Editors signalisiert.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.21 Befehl Verbundene Anschlüsse selektieren

Symbol: 

Funktion: Der Befehl selektiert alle Anschlüsse, die mit der gerade selektierten Leitung oder im seitenorientierten CFC mit der gerade selektierten Verbindungsmarke verbunden sind.

Aufruf: Kontextmenü

Voraussetzungen: Ein CFC-Editor oder ein seitenorientierter CFC-Editor ist aktiv. Eine Leitung und damit genau ein Anschluss oder genau eine Verbindungsmarke ist selektiert.

Siehe auch:

- Befehl Selektierte Anschlüsse verbinden [▶ 1064]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.22 Befehl Attributierte Komponente als Eingang verwenden

Symbol: 

Funktion: Der Befehl ermöglicht das Verbinden einer Strukturkomponenten mit einem Eingang skalaren Typs.

Aufruf: Menü **CFC > Pins**, Kontextmenü > Pins

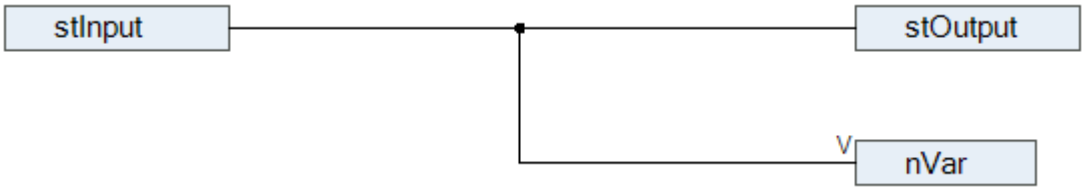
Voraussetzungen: Ein CFC-Editor ist aktiv und ein Bausteineingang ist selektiert.

Die Komponente der Struktur, die mit dem Eingang des Folgebausteins verbunden wird, muss mit dem Attribut {attribute 'ProcessValue'} versehen sein. Der Datentyp der Strukturkomponente muss zu dem Datentyp des nachfolgenden Eingangs kompatibel sein. So verbundene Eingänge werden mit dem Symbol V markiert.

Beispiel

```
TYPE ST_Sample :
STRUCT
    {attribute 'ProcessValue'}
    nVar1 : INT;
    nVar2 : INT;
END_STRUCT
END_TYPE
```

```
PROGRAM MAIN
VAR
    stInput   : ST_Sample;
    stOutput  : ST_Sample;
    nVar      : INT;
END_VAR
```



Wenn Sie bei dieser Verknüpfung den Befehl **Attributierte Komponente als Eingang verwenden** nicht ausführen, wird ein Compilerfehler erzeugt.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.23 Befehl Anschlüsse zurücksetzen

Symbol: 

Funktion: Der Befehl stellt gelöschte Anschlüsse eines Bausteins wieder her.

Aufruf: Menü **CFC > Pins**, Kontextmenü > Pins

Voraussetzungen: Ein CFC-Editor ist aktiv und ein Baustein ist selektiert.

Der Befehl stellt alle Ein- und Ausgänge des Bausteins wieder her, so wie sie in seiner Implementierung definiert sind.

Siehe auch:

- Befehl Nicht verbundene Anschlüsse entfernen [▶ 1067]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]

- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.24 Befehl Nicht verbundene Anschlüsse entfernen

Symbol: 

Funktion: Der Befehl entfernt alle ungenutzten Anschlüsse des selektierten Elements.

Aufruf: Menü **CFC > Pins**, Kontextmenü > Pins

Voraussetzungen: Ein CFC-Editor ist aktiv. Ein Element ist selektiert.

Siehe auch:

- Befehl Anschlüsse zurücksetzen [▶ 1066]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.25 Befehl Eingangsanschluss hinzufügen

Symbol: 

Funktion: Der Befehl fügt dem selektierten Baustein einen weiteren Eingang hinzu.

Aufruf: Menü **CFC > Pins**, Kontextmenü > Pins

Voraussetzungen: Ein CFC-Editor ist aktiv. Ein Baustein ist selektiert.

Siehe auch:

- Befehl Ausgangsanschluss hinzufügen [▶ 1067]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.26 Befehl Ausgangsanschluss hinzufügen

Symbol: 

Funktion: Der Befehl fügt dem selektierten Baustein einen weiteren Ausgang hinzu.

Aufruf: Menü **CFC > Pins**, Kontextmenü > Pins

Voraussetzungen: Ein CFC-Editor ist aktiv. Ein geeigneter Baustein ist selektiert.

Siehe auch:

- Befehl Eingangsanschluss hinzufügen [▶ 1067]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.27 Befehl Alle Verbindungen routen

Symbol: 

Funktion: Der Befehl macht alle manuellen Änderungen an den Verbindungen im Programm rückgängig und stellt den ursprünglichen Zustand wieder her.

Aufruf: Menü **CFC > Routing**, Kontextmenü > Routing

Voraussetzungen: Ein CFC-Editor ist aktiv.

TwinCAT kann Verbindungen, die mit Kontrollpunkten fixiert sind, nicht automatisch routen. Sie müssen die Kontrollpunkte vor dem Ausführen des Befehls entfernen. Dazu verwenden Sie den Befehl Kontrollpunkt entfernen. Weiterhin müssen Sie manuell veränderte Verbindungen, die mit dem Icon  gekennzeichnet sind, lösen. Dazu verwenden Sie den Befehl Verbindung lösen.

Siehe auch:

- Befehl Kontrollpunkt entfernen [▶ 1068]
- Befehl Verbindung lösen [▶ 1065]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.28 Befehl Kontrollpunkt erzeugen

Symbol: 

Funktion: Der Befehl erzeugt einen Kontrollpunkt an einem Verbinder.

Aufruf: Kontextmenü > Routing

Voraussetzungen: Ein CFC-Editor ist aktiv. Der Cursor befindet sich über einer Verbindung.

Der Kontrollpunkt wird an der Stelle der Verbindung erstellt, an dem sich der Cursor bei Aufruf des Befehls befindet. Der Befehl entspricht dem Element **Kontrollpunkt** im Fenster **Werkzeugkasten**.

Siehe auch:

- Befehl Kontrollpunkt entfernen [▶ 1068]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: CFC-Element Kontrollpunkt [▶ 711]

17.12.29 Befehl Kontrollpunkt entfernen

Funktion: Der Befehl entfernt einen Kontrollpunkt.

Aufruf: Kontextmenü > Routing

Voraussetzungen: Ein CFC-Editor ist aktiv. Sie haben eine Verbindungslinie selektiert.

Wenn Sie den Mauszeiger über eine selektierte Verbindungslinie bewegen, werden die vorhandenen Kontrollpunkte mit gelben Kreissymbolen angezeigt. Setzen Sie den Cursor auf den zu löschenden Kontrollpunkt und führen Sie den Befehl aus dem Kontextmenü aus.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: CFC-Element Kontrollpunkt [▶ 711]
- Befehl Kontrollpunkt erzeugen [▶ 1068]

17.12.30 Befehl Verbindungsmarke

Symbol: 

Funktion: Dieser Befehl schaltet die Darstellung der Verbindung zwischen zwei Elementen zwischen einer Verbindungslinie und der Verwendung von Verbindungsmarken hin und her.

Aufruf: Menü **CFC**, Kontextmenü

Voraussetzungen: Ein CFC-Editor ist aktiv. Eine Verbindung oder eine Verbindungsmarke ist selektiert.

Wenn Sie eine Verbindungslinie selektiert haben, entfernt der Befehl diese Linie und fügt eine Verbindungsmarke-Quelle am Ausgang des einen Elements hinzu und eine Verbindungsmarke-Ziel am Eingang des anderen. Beide bekommen standardmäßig den gleichen Namen „C-<n>“, wobei n eine laufende Nummer ist.

Wenn Sie ein Verbindungsmarkenpaar selektieren, wandelt der Befehl diese Marken in eine Verbindungslinie um.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]
- Dokumentation PLC: CFC-Element Verbindungsmarke-Quelle/Ziel [▶ 714]

17.12.31 Befehl Gruppe erzeugen

Symbol: 

Funktion: Der Befehl gruppiert die selektierten Elemente.

Aufruf: Menü **CFC > Gruppieren**, Kontextmenü > Gruppieren

Voraussetzungen: Ein CFC-Editor ist aktiv. Mehrere Elemente sind selektiert.

Gruppierte Elemente können nur noch zusammen bewegt werden. Die Position der Elemente wird durch die Gruppierung nicht beeinflusst.

Siehe auch:

- Befehl Gruppierung aufheben [▶ 1069]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.32 Befehl Gruppierung aufheben

Symbol: 

Funktion: Der Befehl entfernt eine zuvor erstellte Gruppierung.

Aufruf: Menü **CFC > Gruppieren**, Kontextmenü > Gruppieren

Voraussetzungen: Ein CFC-Editor ist aktiviert. Eine Gruppierung ist selektiert.

Siehe auch:

- Befehl Gruppe erzeugen [▶ 1069]
- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.33 Befehl Parameter bearbeiten

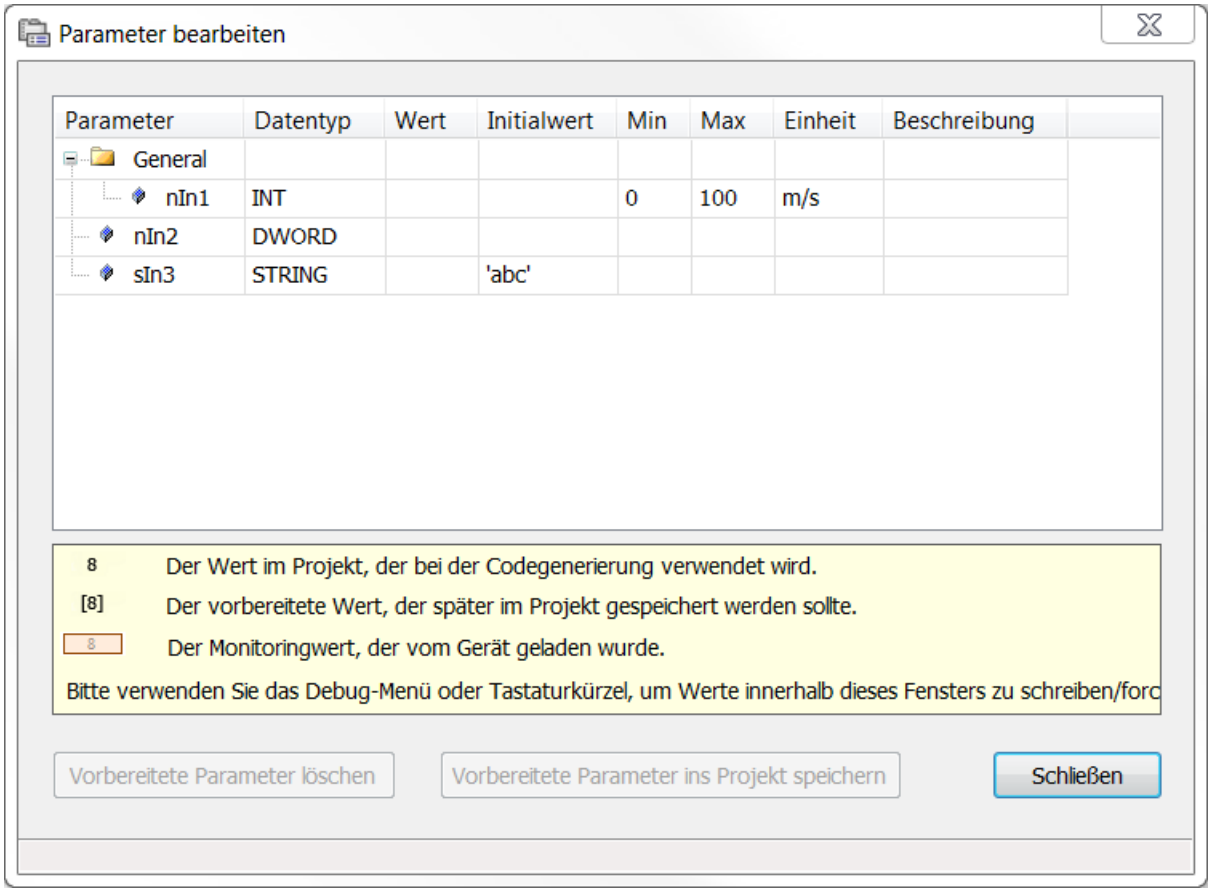
Funktion: Der Befehl öffnet den Dialog „Parameter bearbeiten“, in dem Sie die konstanten Eingangsparameter eines Funktionsbausteins verändern.

Aufruf: Menü **CFC > Parameter bearbeiten**, Kontextmenü > Parameter bearbeiten, Klick auf Funktionsbausteinfeld **Parameter**.

Voraussetzungen: Ein CFC-Editor ist aktiv. Es ist ein Funktionsbaustein instanziiert, der in seiner Deklaration VAR_INPUT CONSTANT-Variablen hat.

Ein Baustein mit VAR_INPUT CONSTANT-Variablen wird von TwinCAT durch das Wort „Parameter“ in der linken unteren Ecke des Bausteins angezeigt.

Dialog Parameter bearbeiten



Parameter	Name der Variablen
Datentyp	Datentyp der Variablen
Wert	Klicken Sie in das Feld um einen Wert einzugeben.
Initialwert	Initialisierungswert
Kategorie	Zusätzliche Informationen zu den Parametern. Diese Werte werden über Attribute definiert und können in diesem Dialog nicht verändert werden • parameterCategory • parameterUnit • parameterMinValue • parameterMaxValue
Einheit	
Min	
Max	
Vorbereitete Parameter löschen	Der Befehl ist aktiv, wenn Sie einen vorbereiteten Wert geschrieben haben (Befehl Debug > Werte schreiben

Nach Verlassen des Felds und Verlassen des Dialogs mit **OK** werden die Werteänderungen im Projekt angewendet.

Beispiel eines Bausteins mit konstanten Eingängen:

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT CONSTANT
    {attribute 'parameterCategory':='General'}
    {attribute 'parameterUnit':= 'm/s'}
    {attribute 'parameterMinValue':= '0'}
    {attribute 'parameterMaxValue':= '100'}
    nIn1 : INT;
    nIn2 : DWORD:=24354333;
    sIn3 : STRING:='abc';
END_VAR
```

Diese Funktionalität und die Deklaration von Variablen mit Schlüsselwort VAR_INPUT CONSTANT gelten nur für den CFC-Editor. Im FUP-Editor zeigt TwinCAT immer alle Eingangsparameter am Baustein an, egal ob sie als VAR_INPUT oder VAR_INPUT CONSTANT deklariert sind. Auch für Texteditoren unterscheidet TwinCAT diesbezüglich nicht.

Siehe auch:

- Befehl Vorbereitete Parameter im Projekt speichern [▶ 1072]
- Dokumentation PLC: Ändern konstanten Eingangsparametern von Funktionsbausteininstanzen [▶ 708]

17.12.34 Befehl FB-Eingang forcen

Verfügbar ab TC3.1 Build 4026

Diese Art des Forcens verwendet intern einen Haltepunkt und ist damit vom Forcen über den Befehl **Werte forcen** zu unterscheiden: Werte, die über den Befehl **FB-Eingang forcen** geforct wurden, reagieren nicht auf die Befehle **Alle Forces anzeigen** oder **Forcen für alle Werte aufheben**.

Funktion: Der Befehl öffnet einen Dialog **Wert forcen** zum Forcen des selektierten Eingangs eines Funktionsbausteins. Das Forcen kann mit dem gleichen Befehl und Dialog wieder aufgehoben werden.

Aufruf: Menü **CFC**, Kontextmenü

Voraussetzungen: Der CFC-Editor ist im Onlinebetrieb und der Eingang des Funktionsbausteins ist selektiert.

Im Dialog **Wert forcen** können Sie entweder einen Wert eingeben, mit dem der Eingang des Funktionsbausteins geforct werden soll, oder den aktuell geforcten Wert wieder entfernen.

Nach dem Forcen erscheint der Eingang grün hinterlegt. Boolesche Eingänge erhalten zudem ein kleines Monitoringfenster mit dem geforcten Wert. Der geforcte Wert wird in der Spalte **Wert** von Monitoringansichten angezeigt, also im Deklarationsteil des Programmbausteins oder in einer Überwachungsliste.

Dialog Wert forcen

Ausdruck	Name des Funktionsbausteineingangs
Typ	Datentyp des Eingangs

Was wollen Sie tun?

Einen neuen Wert zum Forcen setzen	Im Eingabefeld können Sie den gewünschten neuen Wert eintragen. Das Format muss dem Datentyp entsprechen.
Wert entfernen	Das Forcen am Eingang wird aufgehoben.

Siehe auch:

- Dokumentation PLC: Continuous Function Chart (CFC) [▶ 118]
- Dokumentation PLC: CFC-Editor [▶ 701]

17.12.35 Befehl Vorbereitete Parameter im Projekt speichern

Funktion: Der Befehl übernimmt die vorbereiteten Parameterwerte in das Projekt.

Aufruf: Menü **CFC**

Voraussetzungen: Ein CFC-Editor ist aktiv. Parameterwerte von Funktionsbausteininstanzen wurden im Onlinebetrieb geändert. Die Anwendung ist im Offlinebetrieb.

Wenn sich die Werte von Konstanten auf der Steuerung von den Werten in der Applikation unterscheiden, wird dies durch einen roten Stern rechts neben dem Parameterfeld gekennzeichnet. Durch den Befehl **Vorbereitete Parameterwerte übernehmen** übernehmen Sie die Werte der Steuerung in Ihre Applikation.

Siehe auch:

- Befehl Parameter bearbeiten [▶ 1070]
- Dokumentation PLC: Ändern konstanten Eingangsparametern von Funktionsbausteininstanzen [▶ 708]

17.13 FUP/KOP/AWL

17.13.1 Befehl Kontakt einfügen (rechts)

Symbol: 

Funktion: Der Befehl fügt einen Kontakt rechts neben dem gewählten Element ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung, ein Kontakt oder ein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.2 Befehl Netzwerk einfügen

Symbol: 

Funktion: Der Befehl fügt ein weiteres Netzwerk im FUP/KOP/AWL-Editor ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Netzwerk [▶ 694]

17.13.3 Befehl Netzwerk einfügen (unterhalb)

Symbol: 

Funktion: Der Befehl fügt ein weiteres Netzwerk im FUP/KOP/AWL-Editor unterhalb des selektieren Netzwerks ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Netzwerk ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Netzwerk [▶ 694]

17.13.4 Befehl Kommentierung ein/aus

Symbol: 

Funktion: Der Befehl kommentiert das selektierte Netzwerk ein oder aus.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP-, oder AWL-Editor ist aktiv. Ein Netzwerk ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.5 Befehl Zuweisung einfügen

Symbol:

Funktion: Der Befehl fügt eine Zuweisung im FUP oder KOP Editor ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Netzwerk ist selektiert, aber kein Baustein ist selektiert.

In AWL wird eine Zuweisung über die Operatoren LD und ST programmiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Zuweisung [▶ 695]

17.13.6 Befehl Bausteinaufruf einfügen

Symbol: 

Funktion: Der Befehl fügt einen im Projekt verfügbaren Baustein am Ende des selektierten Netzwerks ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Netzwerk ist selektieren, aber kein Baustein ist selektiert.

Wenn Sie den Befehl auswählen, öffnet sich die Eingabehilfe. Dort können Sie den gewünschten Baustein auswählen.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Baustein [▶ 694]

17.13.7 Befehl Baustein mit EN/ENO einfügen

Symbol: 

Funktion: Der Befehl fügt einen Baustein mit einem booleschen Eingang „Enable“ und einen booleschen Ausgang „Enable Out“ am Ende des selektierten Netzwerks ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Netzwerk ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Baustein mit EN/ENO [▶ 695]

17.13.8 Befehl Leeren Baustein einfügen

Symbol: 

Funktion: Der Befehl fügt am Ende des derzeit selektierten Netzwerks einen leeren Funktionsbaustein ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Netzwerk ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Baustein [▶ 694]

17.13.9 Befehl Leeren Baustein mit EN/ENO einfügen

Symbol: 

Funktion: Der Befehl fügt einen leeren Baustein mit einem booleschen Eingang „Enable“ und einen booleschen Ausgang „Enable Out“ am Ende des selektierten Netzwerks ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Ein FUP-Editor, KOP-Editor oder ein AWL-Editor ist aktiv. Ein Netzwerk muss selektiert sein. Kein anderer Baustein darf selektiert sein.

Wenn zum Zeitpunkt des Bausteinaufrufs „Enable“ den Wert FALSE hat, werden die im Baustein definierten Operationen nicht ausgeführt. Andernfalls, also wenn „Enable“ den Wert TRUE hat, werden diese Operationen ausgeführt. Der ENO-Ausgang wirkt als Repeater des EN-Eingangs.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Baustein mit EN/ENO [▶ 695]

17.13.10 Befehl Sprung einfügen

Symbol: 

Funktion: Der Befehl fügt ein Sprungelement vor dem selektierten Element ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Verbinder ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Sprung [▶ 696]

17.13.11 Befehl Sprungmarke einfügen

Symbol: 

Funktion: Der Befehl fügt eine Sprungmarke in das aktuell selektierte Netzwerk ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- AWL-Editor ist aktiv. Ein Netzwerk ist selektiert. Keine Sprungmarke ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Sprungmarke [▶ 696]

17.13.12 Befehl Return einfügen

Symbol:

Funktion: Der Befehl fügt ein Element „Return“ an der selektierten Stelle ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder AWL-Editor ist aktiv. Ein Bausteinausgang ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Return [▶ 696]

17.13.13 Befehl Bausteineingang einfügen

Symbol: 

Funktion: Der Befehl fügt einem erweiterbaren Baustein (ADD, OR, ADD, MUL, SEL) einen weiteren Eingang oberhalb des ausgewählten Eingangs hinzu.

Aufruf: Menü **FUP/KOP/AWL**

Voraussetzungen: Der FUP-, KOP-Editor ist aktiv. Ein Eingang eines Bausteins ist selektiert.

Wenn ein Baustein selektiert ist, ist im Kontextmenü der Befehl **Namen Eingang anhängen** verfügbar. Der Eingang wird am unteren Ende des Bausteins eingefügt.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.14 Befehl Baustein parallel einfügen (unterhalb)

Symbol: 

Funktion: Der Befehl fügt einen leeren Baustein parallel unterhalb des selektierten Bausteins ein.

Aufruf: Menü **FBD/IL/LD**, Kontextmenü

Voraussetzungen: Im KOP-Editor ist ein Baustein selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.15 Befehl Spule einfügen

Symbol: 

Funktion: Der Befehl fügt eine Spule im Netzwerk ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Ein Netzwerk, eine Spule oder ein Verbinder ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Spule [▶ 698]

17.13.16 Befehl Set-Spule einfügen

Symbol: 

Funktion: Der Befehl fügt eine Set-Spule im Netzwerk ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Ein Netzwerk, eine Spule oder eine Leitung ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Spule [▶ 698]

17.13.17 Befehl Reset-Spule einfügen

Symbol: 

Funktion: Der Befehl fügt eine Reset-Spule im Netzwerk ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Ein Netzwerk, eine Spule oder ein Leitung ist selektiert, aber kein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Spule [▶ 698]

17.13.18 Befehl Kontakt einfügen

Symbol: 

Funktion: Der Befehl fügt einen Kontakt links neben dem gewählten Element ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung oder ein Kontakt ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.19 Befehl Kontakt parallel einfügen (unterhalb)

Symbol: 

Funktion: Der Befehl fügt einen Kontakt mit Leitungen parallel unterhalb des gewählten Elements ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung oder ein Kontakt oder ein Baustein ist selektiert.

Geschlossene parallele Verzweigungen in einem KOP-Netzwerk können Sie als „Short Circuit Evaluation“ (SCE) oder OR-Konstrukte programmieren. SCE-Zweige werden mit doppelten vertikalen Linien dargestellt, OR-Zweige mit einfachen. Sehen Sie hierzu die Hilfeseite zu „Geschlossenen Leitungsverzweigungen [▶ 699]“.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]

- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.20 Befehl Kontakt parallel einfügen (oberhalb)

Symbol: 

Funktion: Der Befehl fügt einen Kontakt mit Leitungen parallel oberhalb des gewählten Elements ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung, in Kontakt oder ein Baustein ist selektiert.

> Geschlossene parallele Verzweigungen in einem KOP-Netzwerk können Sie als „Short Circuit Evaluation“ (SCE) oder OR-Konstrukte programmieren. SCE-Zweige werden mit doppelten vertikalen Linien dargestellt, OR-Zweige mit einfachen. Sehen Sie hierzu die Hilfeseite zu „Geschlossenen Leitungsverzweigungen [▶ 699]“.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.21 Befehl Negierten Kontakt einfügen

Symbol: 

Funktion: Der Befehl fügt einen negierten Kontakt links neben dem gewählten Element ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung oder ein Kontakt ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.22 Befehl Negierten Kontakt parallel einfügen (unterhalb)

Symbol: 

Funktion: Der Befehl fügt einen negierten Kontakt mit Leitungen parallel unterhalb des gewählten Elements ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung, ein Kontakt oder ein Baustein ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.23 Befehl Kontakte einfügen: Darunter einfügen

Funktion: Der Befehl fügt einen zuvor kopierten Kontakt mit Leitungen unterhalb des selektierten Elements ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.24 Befehl Kontakte einfügen: Darüber einfügen

Funktion: Der Befehl fügt einen zuvor kopierten Kontakt mit Leitungen oberhalb des selektierten Elements ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung oder ein Kontakt ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.25 Befehl Kontakte einfügen: Rechts einfügen (danach)

Funktion: Der Befehl fügt einen zuvor kopierten Kontakt rechts neben dem selektierten Element ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung oder ein Kontakt ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: KOP-Element Kontakt [▶ 697]

17.13.26 Befehl AWL-Zeile danach einfügen

Symbol: 

Funktion: Der Befehl fügt eine Anweisungszeile unterhalb der selektierten Zeile ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der AWL-Editor ist aktiv. Eine Zeile ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.27 Befehl AWL-Zeile löschen

Symbol: 

Funktion: Der Befehl löscht die selektierte Anweisungszeile.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der AWL-Editor ist aktiv. Eine Zeile ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.28 Befehl Negation

Symbol: 

Funktion: Der Befehl negiert folgende Elemente:

- Ein-/Ausgang eines Bausteins
- Sprung
- Return
- Spule

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, oder KOP-Editor ist aktiv. Das entsprechende Element ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.29 Befehl Flankenerkennung

Symbol FUP: 

Symbol KOP: 

Funktion: Der Befehl fügt eine Flankenerkennung vor dem selektierten Bausteineingang oder Bausteinausgang ein. Durch ein-, zwei- oder dreimaliges Ausführen des Befehls kann zwischen den folgenden Funktionalitäten unterschieden werden:

- Die Flankenerkennung, die beim einmaligen Ausführen des Befehls eingefügt wird, dient zum Erkennen einer steigenden Flanke. Dabei zeigt der Pfeil des Symbols nach rechts.
- Durch erneutes Ausführen des Befehls wird die Flankenerkennung umgedreht, sodass fallende Flanken erkannt werden. Hierbei zeigt der Pfeil des Symbols nach links.
- Wird der Befehl ein weiteres Mal ausgeführt, werden die Flankenerkennung und das Symbol entfernt.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Ein Bausteineingang oder Bausteinausgang ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]

- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.30 Befehl Set/Reset

Symbol: 

Funktion: Der Befehl wechselt bei einem Element mit booleschen Ausgang zwischen Reset, Set und keiner Auszeichnung.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Ein Element mit booleschem Ausgang ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.31 Befehl Weiterverschaltung festlegen

Symbol: 

Funktion: Der Befehl macht den selektierten Bausteinausgang zum weiterführenden Bausteinausgang.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Einer von mehreren Bausteinausgängen ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.32 Befehl Leitungsverzweigung einfügen

Symbol: 

Funktion: Der Befehl erstellt eine offene Leitungsverzweigung an der selektieren Leitung.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Ein Ein- oder Ausgang eines Bausteins ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Leitungsverzweigung [▶ 696]

17.13.33 Befehl Leitungsverzweigung oberhalb einfügen

Symbol: 

Funktion: Der Befehl erstellt eine Leitungsverzweigung oberhalb der selektierten offenen Leitungsverzweigung.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Eine offene Leitungsverzweigung ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Leitungsverzweigung [▶ 696]

17.13.34 Befehl Leitungsverzweigung unterhalb einfügen

Symbol: 

Funktion: Der Befehl erstellt eine Leitungsverzweigung unterhalb der selektierten offenen Leitungsverzweigung.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Eine offene Leitungsverzweigung ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: FUP/KOP/AWL-Element Leitungsverzweigung [▶ 696]

17.13.35 Befehl Verzweigung Startpunkt setzen

Symbol: 

Funktion: Der Befehl setzt den Startpunkt einer Leitungsverzweigung an der selektieren Leitung.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung ist selektiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: Geschlossene Leitungsverzweigung [▶ 699]

17.13.36 Befehl Verzweigung Endpunkt setzen

Symbol: 

Funktion: Der Befehl setzt den Endpunkt einer Leitungsverzweigung an der selektieren Leitung.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der KOP-Editor ist aktiv. Eine Leitung ist selektiert. Ein Startpunkt der Leitungsverzweigung wurde gesetzt.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: Geschlossene Leitungsverzweigung [▶ 699]

17.13.37 Befehl Parallelen Modus wechseln

Funktion: Der Befehl wechselt eine parallele Leitungsverzweigung zwischen einem OR-Konstrukt und der Kurzschlussauswertung („Short Circuit Evaluation“ (SCE)).

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP/KOP/AWL-Editor ist aktiv. Eine senkrechte Linie einer parallelen Leitungsverzweigung ist selektiert.

> Geschlossene parallele Verzweigungen in einem KOP-Netzwerk können Sie als „Short Circuit Evaluation“ (SCE) oder OR-Konstrukte programmieren. SCE-Zweige werden mit doppelten vertikalen Linien dargestellt, OR-Zweige mit einfachen. Sehen Sie hierzu die Hilfeseite zu „Geschlossenen Leitungsverzweigungen [▶ 699]“.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]
- Dokumentation PLC: Geschlossene Leitungsverzweigung [▶ 699]

17.13.38 Befehl Parameter aktualisieren

Funktion: Der Befehl pflegt Änderungen an der Deklaration des selektierten Elements in der Grafik ein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP-, KOP- oder CFC-Editor ist aktiv. Ein Baustein ist selektiert. Eine erweiternde Änderung an der Deklaration wurde vorgenommen.

Der Befehl prüft, ob ein Baustein und seine Deklaration im Deklarationseditor übereinstimmen. Die Änderung wird nur für den Baustein übernommen, wenn die Deklaration erweitert wurde. Löschungen und Überschreibungen werden nicht aktualisiert.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.39 Befehl Nicht verwendete FB-Aufruf-Parameter entfernen

Symbol: 

Funktion: Der Befehl löscht Eingänge und Ausgänge des selektierten Bausteins, denen keine Variable und kein Wert zugewiesen wurden. Die standardmäßigen Eingänge und Ausgänge des Bausteins bleiben aber immer erhalten.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder KOP-Editor ist aktiv. Ein Baustein ist selektiert. Der Baustein verfügt über Schnittstellen, denen kein Wert zugewiesen ist.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.40 Befehl Baustein reparieren

Symbol: 

Funktion: Der Befehl repariert interne Inkonsistenzen am selektierten Baustein.

Aufruf: Menü **FUP/KOP/AWL**, Kontextmenü

Voraussetzungen: Der FUP- oder der KOP-Editor ist aktiv. Der fehlerhafte Baustein ist selektiert. Der Editor hat interne Inkonsistenzen im Programmierbaustein entdeckt, die eventuell automatisch aufgelöst werden können. Die Inkonsistenzen meldet TwinCAT in der Ansicht **Fehlerliste**.

Diese Situation ist denkbar beim Bearbeiten eines Projekts, das mit einer älteren Programmiersystemversion erstellt wurde, die die betreffende Inkonsistenz noch nicht als Fehler behandelte.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.41 Befehl Als Funktionsbausteinsprache anzeigen

HINWEIS

Datenverlust

Eine fehlerfreie Konvertierung setzt syntaktisch korrekten Code voraus. Anderenfalls können Teile der Implementierung verloren gehen.

Funktion: Der Befehl konvertiert die aktive Anweisungsliste oder den aktiven Kontaktplan in die Funktionsbausteinsprache.

Aufruf: Menü **FUP/KOP/AWL > Ansicht**

Voraussetzungen: Der KOP- oder der AWL-Editor ist aktiv.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.42 Befehl Als Kontaktplan anzeigen

HINWEIS

Datenverlust

Eine fehlerfreie Konvertierung setzt syntaktisch korrekten Code voraus. Anderenfalls können Teile der Implementierung verloren gehen.

Funktion: Der Befehl konvertiert das aktuelle Funktionsbausteincode oder die aktive Anweisungsliste in einen Kontaktplan.

Aufruf: Menü **FUP/KOP/AWL > Ansicht**

Voraussetzungen: Der FUP- oder AWL-Editor ist aktiv.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.43 Befehl Als Anweisungsliste anzeigen

AWL kann bei Bedarf über die TwinCAT-Optionen aktiviert werden.

HINWEIS

Datenverlust

Eine fehlerfreie Konvertierung setzt syntaktisch korrekten Code voraus. Anderenfalls können Teile der Implementierung verloren gehen.

Funktion: Der Befehl konvertiert den aktiven Funktionsbausteincode oder den aktiven Kontaktplan in eine Anweisungsliste.

Aufruf: Menü **FUP/KOP/AWL > Ansicht**

Voraussetzungen: Der KOP- oder FUP-Editor ist aktiv.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.13.44 Befehl Gehe zu

Symbol: 

Funktion: Der Befehl lässt Sie zu einem beliebigen Netzwerk springen.

Aufruf: Menü **FUP/KOP/AWL**

Voraussetzungen: Der KOP-Editor, FUP-Editor oder der AWL-Editor ist aktiv. Ein Netzwerk ist selektiert.

Der Befehl öffnet einen Dialog mit einem Eingabefeld. Geben Sie die Nummer des gewünschten Netzwerks in das Eingabefeld ein.

Siehe auch:

- Dokumentation PLC: FUP/KOP/AWL [▶ 113]
- Dokumentation PLC: FUP/KOP/AWL-Editor [▶ 685]

17.14 Ladder-Editor

Die Befehle für das Arbeiten im Ladder-Editor sind im Menü **Ladder** sowie in den jeweiligen Kontextmenüs verfügbar.

17.14.1 Befehl Auskommmentiert

Symbol:

Funktion: Der Befehl schaltet ein Netzwerk [▶ 1088] zwischen den Zuständen auskommentiert und nicht auskommentiert um.

Aufruf: Menü **Ladder**, Kontextmenü

Der Netzwerkinhalt erscheint blass und die Texte in Kursivschrift. Das Netzwerk wird bei der Abarbeitung nicht berücksichtigt.

17.14.2 Befehl Negieren

Symbol:

Funktion: Der Befehl negiert folgende Elemente:

- Eingang/Ausgang eines Bausteins

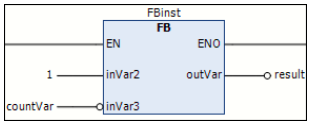
- Sprung
- Return
- Spule
- Kontakt
- Variable

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Das Element, das negiert werden soll, ist selektiert.

Beispiele:

Negierter Eingang und negierter Ausgang an einem Funktionsbaustein:



Negierter Kontakt:

Negierte Spule:

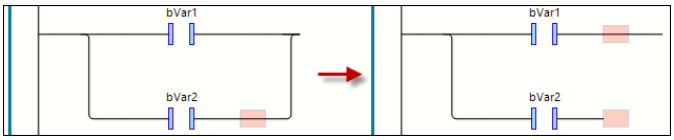
17.14.3 Befehl Parallele Verzweigung öffnen

Symbol:

Funktion: Der Befehl öffnet eine geschlossene parallele Leitungsverzweigung.

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Eine der beiden Leitungslinien der Verzweigung, die wieder geöffnet werden soll, muss selektiert sein.



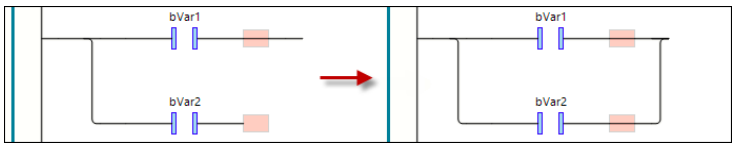
17.14.4 Befehl Parallele Verzweigung schließen

Symbol:

Funktion: Der Befehl schließt die offene parallele Leitungsverzweigung.

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Die beiden Leitungslinien der Verzweigung, die geschlossen werden soll, müssen selektiert sein:



Alternativ kann auch durch Ziehen der Selektionsmarkierung der einen Verzweigungslinie mit der Maus auf die Selektionsmarkierung der anderen Verzweigungslinie eine offene Verzweigung geschlossen werden.

17.14.5 Befehl Set/Reset - Set, Set/Reset - Reset

Symbol: Set , Reset

Funktion: Der Befehl Set versieht das im Editor selektierte Element mit einem Set-Modifizierer. Der Befehl Reset versieht das Element mit einem Reset-Modifizierer.

Beispielsweise können Sie damit eine Spule [▶ 1089] und einen nicht-primären Bausteinausgang zu einer "Set-Spule" machen. Der Befehl ist nur an den passenden Positionen verfügbar.

Aufruf: Menü **Ladder** - Untermenü **Set/Reset**, Kontextmenü

17.14.6 Befehl Flankenerkennung Steigende Flanke

Symbol:

Funktion: Der Befehl fügt eine Erkennung für eine steigende Flanke vor dem selektierten Bausteineingang oder Kontakt ein.

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Ein Bausteineingang oder Kontakt ist selektiert.

17.14.7 Befehl Flankenerkennung: Fallende Flanke

Symbol:

Funktion: Der Befehl fügt eine Erkennung für eine fallende Flanke vor dem selektierten Bausteineingang oder Kontakt ein.

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Ein Bausteineingang oder Kontakt ist selektiert.

17.14.8 Befehl EN/ENO: EN

Symbol:

Funktion: Der Befehl dient dem Hinzufügen oder Entfernen eines booleschen Eingangs "Enable" am selektierten Baustein.

Aufruf: Menü **Ladder** - Untermenü **EN/ENO**, Kontextmenü

Mit dem boolschen Eingang EN wird die Ausführung des Bausteins gesteuert: Wenn der Eingang EN zum Zeitpunkt des Bausteinaufrufs den Wert FALSE hat, werden die im Baustein definierten Operationen nicht ausgeführt. Wenn EN den Wert TRUE hat, werden diese Operationen ausgeführt.

Wenn ein Baustein einen EN-Eingang hat, kann ihm auch ein ENO-Ausgang [▶ 1087] hinzugefügt werden: ENO erhält denselben Wert wie EN.

17.14.9 Befehl EN/ENO: ENO

Symbol:

Funktion: Der Befehl fügt einem Baustein mit EN-Eingang einen ENO-Ausgang hinzu.

Aufruf: Menü **Ladder** - Untermenü **EN/ENO**, Kontextmenü

Voraussetzungen: Ein Baustein, der einen EN-Eingang hat, ist selektiert.

Der ENO-Ausgang erhält denselben Wert wie der EN-Eingang.

17.14.10 Befehl Netzwerk einfügen

Symbol:

Funktion: Der Befehl fügt ein weiteres Netzwerk im Ladder-Editor ein. Sie erkennen die möglichen

Einfügepositionen am Plussymbol am Cursor , wenn Sie das Element mit der Maus über den Implementierungsteil ziehen.

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Kein Baustein ist selektiert.

17.14.10.1 Element: Netzwerk

Ein Netzwerk ist die Basiseinheit eines KOP-Programms. Im Ladder-Editor sind die Netzwerke in einer Liste untereinander angeordnet. Jedes Netzwerk ist an der linken Seite mit einer fortlaufenden Netzwerknummer versehen und kann Folgendes enthalten: Logische und arithmetische Ausdrücke, Programm-, Funktions- und Funktionsbausteinaufrufe, eine Sprung- oder Return-Anweisung.

Sie können jedes Netzwerk mit einem Titel, einem Kommentar oder einer Sprungmarke versehen:

Netzwerktitel: Doppelklicken Sie zur Eingabe auf die erste Zeile oben in einem Netzwerk.

Netzwerkkommentar: Doppelklicken Sie zur Eingabe auf die zweite Zeile oben in einem Netzwerk.

Sprungmarke: Doppelklicken Sie zur Eingabe auf die dritte Zeile oben in einem Netzwerk. Die Sprungmarke kann dann beim Element Sprung [▶ 1089] als Ziel angegeben werden.

In den TwinCAT-Optionen, Kategorie Ladder [▶ 1028] definieren Sie, ob Netzwerktitel, Netzwerkkommentar und Trennlinien zwischen den einzelnen Netzwerken im Editor angezeigt werden.

17.14.11 Befehl Kontakt einfügen

Symbol: , im Editor

Funktion: Der Befehl fügt einen Kontakt im Netzwerk, links vom selektierten Element ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen

Einfügepositionen am Plussymbol am Cursor .

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Eine Leitung oder ein Kontakt ist selektiert.

17.14.11.1 Element: Kontakt

Ein Kontakt gibt das Signal TRUE (ON) oder FALSE (OFF) von links nach rechts weiter, bis das Signal eine Spule im rechten Teil des Netzwerks erreicht. Zu diesem Zweck wird dem Kontakt eine boolesche Variable zugewiesen, die das Signal enthält. Dazu ersetzen Sie den Platzhalter **???** oberhalb des Kontakts mit dem Namen einer booleschen Variablen.

Sie können mehrere Kontakte sowohl in Reihe als auch parallel anordnen. Bei zwei parallelen Kontakten muss nur einer den Wert TRUE erhalten, damit ON nach rechts weitergegeben wird. Wenn Kontakte hintereinandergeschaltet sind, müssen alle Kontakte den Wert TRUE erhalten, damit vom letzten Kontakt der Reihe ON nach rechts weitergegeben wird. Sie können also mit KOP elektrische Parallel- und Reihenschaltungen programmieren.

Ein negierter Kontakt gibt das Signal TRUE weiter, wenn der Variablenwert FALSE ist. Sie können die Negation eines eingefügten Kontakts mit Hilfe des Befehls **Ladder** → **Negieren** aktivieren oder deaktivieren.

17.14.12 Befehl Spule einfügen

Symbol:

Symbol im Editor:

Funktion: Der Befehl fügt eine Spule im Netzwerk ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen Einfügepositionen am Plussymbol am Cursor .

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Ein Netzwerk, eine Spule oder ein Verbinder ist selektiert. Kein Baustein ist selektiert.

17.14.12.1 Element: Spule

Eine Spule nimmt den Wert auf, der von links geliefert wird und speichert ihn in der ihr zugewiesenen booleschen Variablen. Ihr Eingang kann den Wert TRUE (ON) oder FALSE (OFF) haben.

In einer negierten [▶ 1085] Spule wird der negierte Wert des eingehenden Signals in der booleschen Variablen gespeichert, die der Spule zugewiesen ist.

Mit den Befehlen des Untermenüs Set/Reset [▶ 1087] können Sie eine "Set-Spule" oder "Reset-Spule" definieren:

Set-Spule: Wenn der Wert TRUE an einer Set-Spule ankommt, behält die Spule den Wert TRUE. Solange die Applikation läuft, kann der Wert an dieser Stelle nicht mehr überschrieben werden.

Reset-Spule: Wenn der Wert TRUE an einer Reset- Spule ankommt, behält die Spule den Wert FALSE. Solange die Applikation läuft, kann der Wert an dieser Stelle nicht mehr überschrieben werden.

17.14.13 Befehl Baustein einfügen

Symbol:

Funktion: Der Befehl fügt einen Baustein am Ende des selektierten Netzwerks ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen Einfügepositionen am Plussymbol am Cursor .

Aufruf: Menü **Ladder**, Kontextmenü

Der Befehl fügt standardmäßig einen EN/ENO-Baustein ein. Mit den Befehlen des Menüs **Ladder** EN/ENO können die Modifizierer EN [▶ 1087] und ENO [▶ 1087] entfernt und hinzugefügt werden.

AKTUELL NOCH NICHT IMPLEMENTIERT: Wenn es sich um einen im Projekt definierten Baustein handelt, muss nach einer Änderung an diesem Baustein die Verwendung im Ladder aktualisiert werden.

17.14.13.1 Element: Baustein

Ein Bausteinelement kann zusätzliche Funktionen repräsentieren: Beispielsweise IEC-Funktionsbausteine, IEC-Funktionen, Bibliotheksbausteine, Operatoren.

Wenn der Baustein eine Bilddatei mitliefert, kann das Bausteinsymbol innerhalb des Bausteins angezeigt werden. Voraussetzung: In den TwinCAT-Optionen, Kategorie **Ladder** ist die Option **Bausteinsymbol anzeigen** aktiviert.

17.14.14 Befehl Sprung einfügen

Symbol:

Funktion: Der Befehl fügt ein Sprungelement vor dem selektierten Element ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen Einfügepositionen am

Plussymbol am Cursor  .

Aufruf: Menü **Ladder**, Kontextmenü

Voraussetzungen: Ein Verbinder ist selektiert.

17.14.14.1 Element: Sprung

In KOP wird ein Sprung abhängig von der aktuellen Cursorposition entweder direkt vor einem Eingang, direkt nach einem Ausgang oder am Ende des Netzwerks eingefügt.

Direkt hinter dem Sprung-Element geben Sie die Sprungmarke eines Netzwerks als Sprungziel ein.

17.14.15 Befehl Return einfügen

Symbol:

Funktion: Der Befehl fügt ein Element **Return** an der selektierten Stelle ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen Einfügepositionen am

Plussymbol am Cursor  .

Aufruf: Menü **Ladder**, Kontextmenü

17.14.15.1 Element: Return

Das Element dient dazu, die Abarbeitung des Ladder-Bausteins sofort abzubrechen, wenn der Eingang des **Return**-Elements TRUE wird. Sie können die **Return**-Anweisung parallel zu oder anschließend an die vorausgehenden Elemente platzieren.

17.14.16 Befehl Eingang einfügen

Symbol:

Funktion: Der Befehl fügt einen Eingang an der gewählten Position ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen Einfügepositionen am

Plussymbol am Cursor  . Die zunächst angezeigten *???* können Sie mit einem Variablennamen oder einer Konstanten ersetzen.

Aufruf: Menü **Ladder**, Kontextmenü

17.14.16.1 Element: Eingang

Das Element dient der Zuweisung eines Variablenwerts zu einem anderen Element im Signalfluss. Es hat eine weiterführende Linie nach rechts.

17.14.17 Befehl Ausgang einfügen

Symbol:

Funktion: Der Befehl fügt einen Ausgang an der gewählten Position ein. Wenn Sie das Element aus der Werkzeugbox in den Implementierungsteil ziehen, erkennen Sie die möglichen Einfügepositionen am

Plussymbol am Cursor  . Die zunächst angezeigten **???** können Sie mit einem Variablennamen oder einer Konstanten ersetzen.

Aufruf: Menü **Ladder**, Kontextmenü

17.14.17.1 Element: Ausgang

Das Element dient der Zuweisung eines von links kommenden Signals zu der mit dem Element verknüpften Variablen. Es hat eine von links kommende Eingangslinie.

17.14.18 Befehl In neuen Ladder konvertieren

Verfügbar im FUP/KOP-Editor.

17.15 Deklarationen

17.15.1 Befehl Einfügen

Symbol: 

Funktion: Der Befehl fügt eine neue Zeile für eine Variablendeklaration im tabellarischen Deklarationseditor ein und das Eingabefeld für den Variablennamen öffnet sich.

Aufruf: Schaltfläche im Deklarationskopf des tabellarischen Deklarationseditors, Kontextmenü im tabellarischen Deklarationseditor

Um die anderen Felder der Deklarationszeile zu bearbeiten, doppelklicken Sie die Felder und wählen Sie die Angaben aus den Auswahllisten oder mithilfe der entsprechenden Dialoge.

Siehe auch:

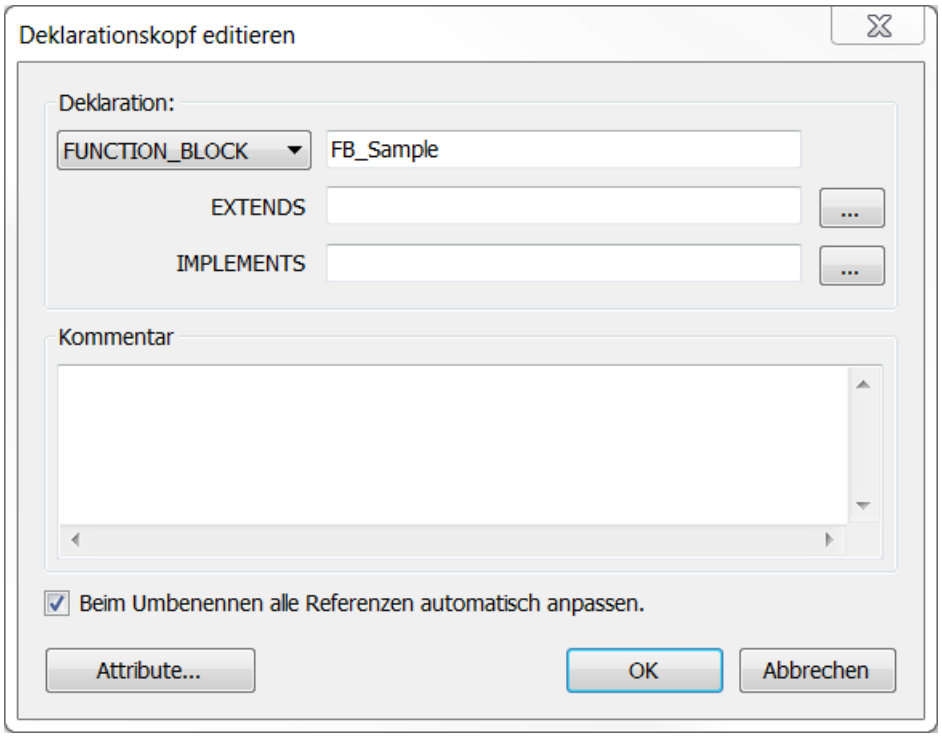
- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]

17.15.2 Befehl Deklarationskopf editieren

Funktion: Der Befehl öffnet den Dialog **Deklarationskopf editieren**, der im tabellarischen Deklarationseditor zur Konfiguration der Kopfzeile einer POU dient.

Aufruf: Mausklick auf die Kopfleiste des tabellarischen Deklarationseditors, Kontextmenü im tabellarischen Deklarationseditor

Dialog Deklarationskopf editieren



Deklaration	Auswahlliste zur Änderung des POU-Typs • PROGRAM • FUNCTION_BLOCK ◦ EXTENDS: Eingabefeld für einen Basis-Funktionsbaustein ◦ IMPLEMENTS: Eingabefeld für eine Schnittstelle • FUNCTION ◦ Rückgabetyp Eingabefeld mit aktuellem POU-Namen: Sie können den Namen der POU ändern.
Beim Umbenennen alle Referenzen automatisch anpassen	☑: Dialog **Refactoring** öffnet sich. ☐: Die Umbenennung wird nur im Deklarationskopf der POU wirksam.
Attribute	Der Dialog **Attribute** öffnet sich zur Eingabe von Attributen und Pragmas.

Siehe auch:

- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]
- Dokumentation PLC: Pragmas [▶ 828]
- Dokumentation PLC: Refactoring [▶ 168]

17.15.3 Befehl Nach unten verschieben

Symbol: 

Funktion: Der Befehl verschiebt eine Variablendeklaration um eine Zeile nach unten.

Aufruf: Schaltfläche im Deklarationskopf des tabellarischen Deklarationseditors, Kontextmenü im tabellarischen Deklarationseditor

Voraussetzung: Im tabellarischen Deklarationseditor ist eine Zeile mit einer Variablendeklaration selektiert.

Siehe auch:

- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]

17.15.4 Befehl Nach oben verschieben

Symbol: 

Funktion: Der Befehl verschiebt eine Variablendeklaration um eine Zeile nach oben.

Aufruf: Schaltfläche im Deklarationskopf des tabellarischen Deklarationseditors, Kontextmenü im tabellarischen Deklarationseditor

Voraussetzung: Im tabellarischen Deklarationseditor ist eine Zeile mit einer Variablendeklaration selektiert.

Siehe auch:

- Dokumentation PLC: Deklarationseditor verwenden [▶ 72]

17.16 Textliste

17.16.1 Befehl Sprache einfügen

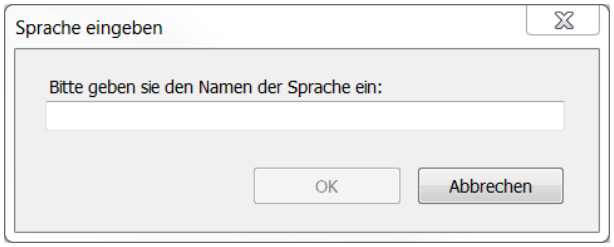
Symbol: 

Funktion: Der Befehl fügt eine weitere Sprachspalte der Textliste hinzu.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung: Eine Textliste oder eine globale Textliste ist geöffnet und aktiv.

Geben Sie im Dialog **Sprache eingeben** ein Kürzel für die neue Sprache ein, zum Beispiel „en-US“. TwinCAT fügt das Kürzel als Spaltenüberschrift ein.



Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.2 Befehl Sprache entfernen

Symbol: 

Funktion: Der Befehl entfernt die ausgewählte Sprachspalte aus der Textliste.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung: Eine Textliste oder eine globale Textliste ist geöffnet und aktiv. Ein Feld in der Spalte der Sprache, die Sie entfernen wollen, ist selektiert.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.3 Befehl Text einfügen

Symbol:

Funktion: Der Befehl fügt eine neue Zeile oberhalb der selektieren Zeile in die Textliste ein. Ein Eingabefeld unter Standard öffnet sich, in dem Sie den Ausgangstext eingeben.

Aufruf: Menü **Textliste**

Voraussetzung: Eine Textliste, keine GlobalTextList, ist geöffnet und aktiv. Ein Feld in der Tabelle ist selektiert.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.4 Befehl Import/Export Textlisten

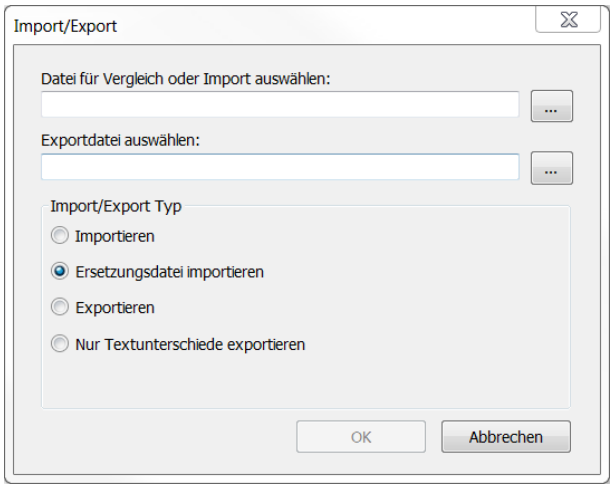
Symbol:

Funktion: Der Befehl exportiert eine aktive Textliste, importiert eine Datei oder gleicht eine Textliste mit einer Datei ab. Die Datei hat CSV-Format. Der Dialog **Import/Export** bietet dafür Optionen an.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung: Eine Textliste oder eine globale Textliste ist geöffnet und aktiv.

Dialog Import/Export



Datei für Vergleich oder Import auswählen	Datei, die TwinCAT ausliest. ... öffnet den Dialog **Textlistendatei auswählen**, in dem Sie eine Datei auswählen können.
Exportdatei auswählen	Datei, in die TwinCAT schreibt. ... öffnet den Dialog **Textlistendatei auswählen**, in dem Sie ein Datei und Verzeichnis auswählen können.

Import/Export Typ:

Importieren	Voraussetzung: In **Datei für Vergleich oder Import auswählen** ist eine Datei ausgewählt. Die Datei kann Textlisteneinträge sowohl für die globale Textliste als auch für Textlisten enthalten. Globale Textliste: • TwinCAT liest die Datei, vergleicht für gleichen Ausgangstext die Textlisteneinträge und übernimmt Unterschiede bei den Übersetzungen. TwinCAT überschreibt gegebenenfalls die Übersetzungen im Projekt. Textlisten: • TwinCAT liest die Datei, vergleicht für gleiche IDs die Textlisteneinträge und übernimmt Unterschiede im Ausgangstext und den Übersetzungen in das Projekt. TwinCAT überschreibt gegebenenfalls die Textlisteneinträge im Projekt. • Wenn die Datei eine neue ID enthält, wird der Textlisteneintrag in die Textliste des Projekts importiert und die Textliste ergänzt.
Ersetzungsdatei importieren	Voraussetzung: In **Datei für Vergleich oder Import auswählen** ist eine Ersetzungsdatei ausgewählt. Die Ersetzungsdatei enthält Ersetzungen für die globale Textliste. TwinCAT arbeitet die Ersetzungsdatei zeilenweise ab und führt die spezifizierten Ersetzungen in der globalen Textliste durch. Der Aufbau der Ersetzungsdatei ist im Abschnitt Statischen Text in einer globalen Textliste verwalten [▶ 148] beschrieben.
Exportieren	Voraussetzung: In „Exportdatei auswählen“ ist die Datei, in die TwinCAT schreibt, ausgewählt. TwinCAT exportiert alle Texte aus allen Textlisten des aktuellen Projekts. In der Exportdatei werden alle im Projekt vorhandenen Sprachen als Spalten eingefügt. Die Datei kann dazu verwendet werden die sprachenabhängigen Texte extern übersetzen zu lassen.
Nur Textunterschiede exportieren	Voraussetzung: In **Datei für Vergleich oder Import auswählen** ist eine Importdatei für den Vergleich ausgewählt. In **In Exportdatei auswählen** ist eine Exportdatei, in die TwinCAT schreibt, ausgewählt. TwinCAT liest die Importdatei, vergleicht die Zeilen der aktiven Textliste damit. Wenn Zeilen übereinstimmen, ignoriert TwinCAT diese. Wenn Zeilen sich unterscheiden, schreibt TwinCAT die Zeile in die Exportdatei und übernimmt dabei gegebenenfalls Übersetzungen aus der Textliste. TwinCAT übernimmt die Übersetzungen aus der Importdatei und überschreibt Sie gegebenenfalls.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.5 Befehl Nicht verwendete Textlisteneinträge entfernen

Symbol: 

Funktion: Der Befehl prüft, ob ein Textlisteneintrag im Projekt als statischer Text verwendet wird. Wenn nicht, entfernt TwinCAT ihn aus der Textliste.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung: Die globale Textliste ist geöffnet und aktiv. Ein Feld in der Tabelle ist selektiert.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.6 Befehl Visualisierungstext-IDs prüfen

Symbol: 

Funktion: Der Befehl prüft, ob die ID eines Textlisteneintrags im Projekt korrekt ist und meldet das Ergebnis.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung: Die globale Textliste ist geöffnet und aktiv. Ein Feld in der Tabelle ist selektiert.

Wenn TwinCAT beim Prüfen feststellt, dass die globale Textliste und die statischen Texte der Visualisierungen nicht übereinstimmen, kann das daran liegen, dass die globale Textliste schreibgeschützt ist oder war. Voraussetzung dafür ist, dass Sie eine Benutzerverwaltung im Projekt eingerichtet haben.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.7 Befehl Visualisierungstext-IDs aktualisieren

Symbol: 

Funktion: Der Befehl aktualisiert alle inkonsistenten IDs in einer statischen Textliste.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung: Die globale Textliste ist geöffnet und aktiv. Ein Feld in der Tabelle ist selektiert. Das Objekt ist schreibgeschützt.

Wenn TwinCAT beim Prüfen feststellt, dass die globale Textliste und die statischen Texte der Visualisierungen nicht übereinstimmen, kann das daran liegen, dass die globale Textliste unter Schreibschutz steht oder stand. Voraussetzung dafür ist, dass Sie eine Benutzerverwaltung im Projekt eingerichtet haben.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.8 Befehl Alles als Text exportieren

Symbol: 

Funktion: Der Befehl exportiert alle Textlisten des Projekts.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung:

- Eine Textliste oder eine globale Textliste ist geöffnet und aktiv.
- Die Visualisierung codiert die Zeichen der Texte nicht in Unicode.

TwinCAT erstellt dabei für jede Textliste eine Datei als einfachen Text im Format .txt. Der Name der Textliste wird zum Namen der Datei. Die Datei wird in das Verzeichnis des TwinCAT Projekts gespeichert.

Eine Steuerung kann dieses Format lesen und verwenden. Sie können die Datei zum Beispiel auf eine Steuerung kopieren und über eine Einstellung im Visualisierungsmanager konfigurieren, dass beim Laden des SPS-Projekts die Textlisten nicht nochmals übertragen werden.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.9 Befehl Alles als Unicode-Text exportieren

Symbol: 

Funktion: Der Befehl exportiert alle Textlisten des Projekts.

Aufruf: Menü **Textliste**, Kontextmenü

Voraussetzung:

- Eine Textliste oder eine globale Textliste ist geöffnet und aktiv.
- Die Visualisierung codiert die Zeichen der Texte in Unicode.
  - Die Option **Unicode-Zeichenfolgen verwenden** im Visualisierungsmanager ist aktiviert.

TwinCAT erstellt dabei für jede Textliste eine Datei als einfachen Text im Format .txt. Der Name der Textliste wird zum Namen der Datei. Die Datei wird in das Verzeichnis des TwinCAT Projekts gespeichert.

Eine Steuerung kann dieses Format lesen und verwenden. Sie können die Datei zum Beispiel auf eine Steuerung kopieren und über eine Einstellung im Visualisierungsmanager konfigurieren, dass beim Laden des SPS-Projekts die Textlisten nicht nochmals übertragen werden.

Siehe auch:

- Dokumentation PLC: Text in einer Textliste verwalten [▶ 145]

17.16.10 Befehl Textlistenunterstützung hinzufügen

Symbol: 

Funktion: Der Befehl fügt dem selektierten DUT-Objekt des Typs **Enumeration** eine Textlistenunterstützung hinzu.

Aufruf: Kontextmenü eines Standard-DUT-Objekts vom Typ **Enumeration** ()

Die Textlistenunterstützung ermöglicht eine Lokalisierung der Enumerationskomponentenbezeichner und eine Darstellung des symbolischen Komponentenwerts in einer Textausgabe in der Visualisierung.

Siehe auch:

- Dokumentation PLC: Objekt DUT [▶ 77]
- Befehl Textlistenunterstützung entfernen [▶ 1097]

17.16.11 Befehl Textlistenunterstützung entfernen

Symbol: 

Funktion: Der Befehl entfernt die Textlistenunterstützung vom selektierten Enumerationsobjekt.

Aufruf: Kontextmenü eines Objekts einer textlistenunterstützten Enumeration ().

Die Textlistenunterstützung ermöglicht eine Lokalisierung der Enumerationskomponentenbezeichner und eine Darstellung des symbolischen Komponentenwerts in einer Textausgabe in der Visualisierung.

17.17 Rezepturen

17.17.1 Befehl Rezeptur einfügen

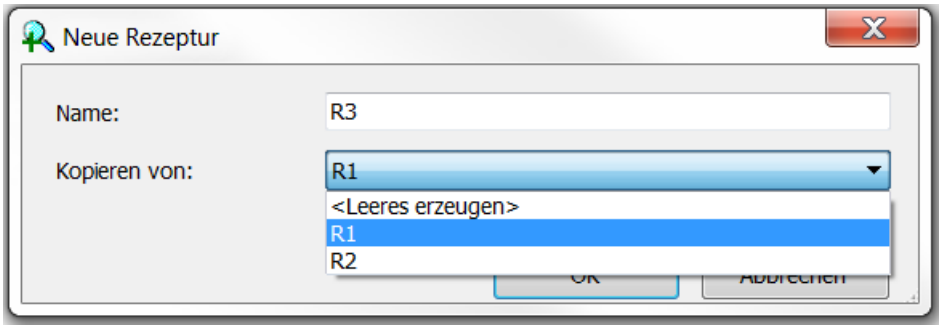
Symbol: 

Funktion: Der Befehl öffnet einen Dialog zum Einfügen einer neuen Rezeptur (einer neue Spalte) in die Rezepturdefinition.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Eine Rezepturdefinition ist im Editor geöffnet.

Wenn Sie den Befehl ausführen, öffnet sich ein Dialog, in dem Sie den Namen der neuen Rezeptur festlegen. Der Dialog bietet Ihnen auch die Möglichkeit, bestehende Rezepturen in die neue Rezeptur zu kopieren.



Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.2 Befehl Rezeptur entfernen

Symbol: 

Funktion: Der Befehl löscht eine Rezeptur aus der gerade geöffneten Rezepturdefinition.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Ein Feld in der Rezepturspalte einer Rezepturdefinition ist selektiert.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.3 Befehl Rezeptur laden

Symbol: 

Funktion: Der Befehl lädt eine Rezeptur aus einer Datei.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Ein Feld in der Rezepturspalte einer Rezepturdefinition ist selektiert.

Wenn Sie den Befehl ausführen, öffnet sich der Standarddialog zur Auswahl einer Datei. Der Filter ist automatisch auf die Dateierweiterung *.txtrecipe eingestellt. Nach dem Laden werden die Werte der selektierten Rezeptur der Rezepturdefinition überschrieben und die Anzeige aktualisiert.

Wenn Sie nur einzelne Variablen der Rezeptur mit neuen Werten überschreiben möchten, entfernen Sie vor dem Laden in der Rezepturdatei die Werte für die restlichen Variablen. Einträge ohne Wertangabe werden nicht eingelesen und somit bleiben diese Variablen auf der Steuerung und im Projekt durch die Aktualisierung unberührt. Im Folgenden sehen Sie das Beispiel für die Einträge einer Rezepturdatei, bei deren Laden nur MAIN.nVar mit einem neuen Wert (6) geschrieben wird:

```
MAIN.nVar1:=
MAIN.nVar2:=6
MAIN.nVar3:=
MAIN.sVar4:=
MAIN.wsVar5:=
```

Bei Werten vom Datentyp REAL/LREAL wird in manchen Fällen auch der Hexadezimalwert in die Rezepturdatei geschrieben. Dies ist notwendig, damit bei der Rückkonvertierung der exakt identische Wert wiederhergestellt wird. In diesem Fall ändern Sie den Dezimalwert und löschen Sie den Hexadezimalwert.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.4 Befehl Rezeptur speichern

Symbol: 

Funktion: Der Befehl speichert die Werte der Variablen einer Rezeptur in eine Datei.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Ein Wert der Rezeptur in der Rezepturdefinition ist selektiert.

Beim Ausführen des Befehls speichert TwinCAT die Werte der selektierten Rezeptur in eine Datei mit der Erweiterung *.txtrecipe, deren Name definiert werden muss. Dazu öffnet der Standarddialog zum Speichern einer Datei. Das Format ergibt sich aus den Einstellungen des Rezepturverwalters in der Registerkarte **Speicherung**.

Überschreiben der impliziten Rezepturdatei

Die implizit verwendeten Rezepturdateien, die als Zwischenablage beim Lesen und Schreiben von Rezepturen nötig sind, dürfen nicht überschrieben werden. D. h. der Name für die Datei muss anders lauten als <Rezepturname>.<Rezepturdefiinitionsname>.txtrecipe

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.5 Befehl Rezeptur lesen

Symbol: 

Funktion: Der Befehl liest die Werte der Variablen einer Rezeptur aus der Steuerung.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und ein Wert der Rezeptur in der Rezepturdefinition ist selektiert.

Beim Ausführen des Befehls überschreibt TwinCAT die Werte der selektierten Rezeptur mit den gelesenen Werten aus der Steuerung. Dabei werden die Werte implizit gespeichert (in einer Datei auf der Steuerung) und gleichzeitig in der Tabelle der Rezepturdefinition angezeigt.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.6 Befehl Rezeptur schreiben

Symbol: 

Funktion: Der Befehl schreibt die Werte einer Rezeptur in die Variablen in der Steuerung.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und ein Wert der Rezeptur in der Rezepturdefinition ist selektiert.

Beim Ausführen des Befehls überschreibt TwinCAT die Werte in der Steuerung mit den Werten der selektierten Rezeptur.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.7 Befehl Rezeptur laden und schreiben

Symbol: 

Funktion: Der Befehl lädt eine Rezeptur aus einer Datei und schreibt die Werte auf die Variablen in der Steuerung.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und ein Wert der Rezeptur in der Rezepturdefinition ist selektiert.

Nach dem Ausführen des Befehls werden Sie gefragt, ob die Werte aus der Datei auch in die Rezeptur im Projekt, oder nur zur SPS geschrieben werden sollen. Ein Aktualisieren der Werte in der Rezeptur kann bei erneutem Einloggen einen Online Change notwendig machen.

Wenn Sie den Befehl ausführen, überschreibt TwinCAT die Werte der selektierten Rezeptur der Rezepturdefinition. Weiterhin werden die Werte der Variablen in der Steuerung mit diesen Rezepturwerten überschrieben.

Wenn Sie nur einzelne Variablen der Rezeptur mit neuen Werten überschreiben möchten, entfernen Sie vor dem Laden in der Rezepturdatei die Werte für die restlichen Variablen. Einträge ohne Wertangabe werden nicht eingelesen und somit bleiben diese Variablen auf der Steuerung und im Projekt durch die Aktualisierung unberührt. Im Folgenden sehen Sie das Beispiel für die Einträge einer Rezepturdatei, bei deren Laden nur MAIN.nVar1 mit einem neuen Wert (6) geschrieben wird.

```
MAIN.nVar1:=
MAIN.nVar2:=6
MAIN.nVar3:=
MAIN.sVar4:=
MAIN.wstVar5:=
```

Bei Werten vom Datentyp REAL/LREAL wird in manchen Fällen auch der Hexadezimalwert in die Rezepturdatei geschrieben. Dies ist notwendig, damit bei der Rückkonvertierung der exakt identische Wert wiederhergestellt wird. In diesem Fall ändern Sie den Dezimalwert und löschen Sie den Hexadezimalwert.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.8 Befehl Rezeptur lesen und speichern

Symbol: 

Funktion: Der Befehl liest die Werte der Variablen einer Rezeptur aus der Steuerung und speichert sie in eine Datei.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und ein Wert der Rezeptur in der Rezepturdefinition ist selektiert.

Nach dem Ausführen des Befehls werden Sie gefragt, ob die Variablenwerte in die Rezeptur im Projekt eingelesen oder nur gespeichert werden sollen. Ein Aktualisieren der Werte in der Rezeptur könnte bei erneutem Einloggen einen Online-Change notwendig machen.

Die Werte werden gemäß den Einstellungen des Rezepturverwalters (Registerkarte **Speicherung**) mit dem Standardnamen für Rezepturdateien abgespeichert.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.9 Befehl Variable einfügen

Symbol: 

Funktion: Der Befehl fügt eine Variable in die gerade geöffnete Rezepturdefinition vor der ausgewählten Position ein.

Aufruf: Menü **Rezepturen**, Kontextmenü

Voraussetzung: Eine Rezepturdefinition ist im Editor geöffnet ist und die einfache Ansicht gewählt.

Dabei fügt TwinCAT in der Spalte **Variable** den Standardtext „NeueVariable“ ein. Diesen Namen müssen Sie durch den entsprechenden gültigen Variablennamen ersetzen werden. Dazu öffnen Sie die Eingabehilfe über Schaltfläche  oder tragen Sie den Variablennamen direkt in das Tabellenfeld ein.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.10 Befehl Variablen entfernen

Symbol 

Funktion: Der Befehl entfernt in einer Rezepturdefinition die selektierte(n) Variablen.

Aufruf: Taste **[Entf]**, Kontextmenü

Voraussetzung: Sie haben eine Variable selektiert.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.11 Befehl Strukturierte Variablen aktualisieren

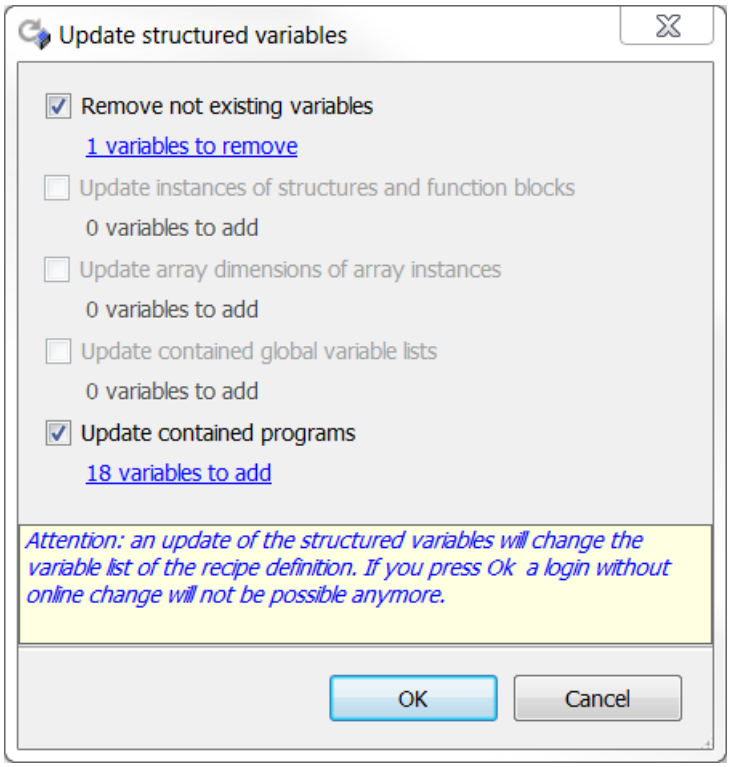
Symbol: 

Funktion: Der Befehl öffnet den Dialog **Strukturierte Variablen aktualisieren**.

Aufruf: Menü **Rezepturen**

In dem Dialog können Sie Rezepturdefinitionen aktualisieren, wenn sich die Deklaration einer strukturierten Variablen oder eines Bausteins geändert hat. Wenn beispielsweise die Dimension eines Arrays verändert wurde, können Sie die Einträge in der Rezepturdefinition automatisch entsprechend entfernen oder hinzufügen.

Dialog Strukturierte Variablen aktualisieren



Nicht existierende Variablen entfernen	☑: Variablen, die es auf Grund der Änderung eines strukturieren Elements im Projekt nicht länger gibt, werden aus der Rezepturdefinition entfernt.
Instanzen von Strukturen und Funktionsbausteinen aktualisieren	☑: Wenn die Deklaration einer Struktur oder eines Funktionsbausteins erweitert wird, die/der mit einer Instanz in der Rezepturdefinition vertreten ist, werden der Rezepturdefinition die entsprechenden Variablen hinzugefügt.
Array-Dimensionen von Array-Instanzen aktualisieren	☑: Wenn die Dimension eines Arrays erweitert wird, das mit einer Instanz in der Rezepturdefinition vertreten ist, werden der Rezepturdefinition die entsprechenden Variablen hinzugefügt.
Enthaltene globale Variablenlisten aktualisieren	☑: Wenn die Deklaration einer globalen Variablenliste erweitert wird, die mit einer Instanz in der Rezepturdefinition vertreten sind, werden der Rezepturdefinition die entsprechenden Variablen hinzugefügt.
Enthaltene Programme aktualisieren	☑: Wenn die Deklaration eines Programms erweitert wird, das in der Rezepturdefinition instanziiert wurde, werden der Rezepturdefinition die entsprechenden Variablen hinzugefügt.

Siehe auch:

- Dokumentation PLC: Werte ändern mit Rezepturen [▶ 240]

17.17.12 Befehl Rezepturen vom Gerät laden

Symbol: 

Funktion: Der Befehl stößt die Synchronisierung der Rezepturen der gerade geöffneten Rezepturdefinition im Projekt mit den auf dem Gerät in Form von Rezepturdateien liegenden Rezepturen an.

Aufruf: Menü **Rezepturen**

Voraussetzung: Das SPS-Projekt ist im Onlinebetrieb und Sie haben eine Rezepturdefinition im Editor geöffnet. Im Detail bedeutet die Synchronisierung Folgendes:

- Die aktuellen Werte für die im Projekt vorliegenden Rezepturvariablen werden mit den Werten aus den Rezepturen auf der Steuerung überschrieben. Dadurch wird beim nächsten Einloggen möglicherweise ein Online-Change ausgelöst.
- Wenn in den Rezepturdateien auf der Steuerung Rezepturvariablen definiert sind, die in der Rezepturdefinition im Projekt fehlen, werden diese Variablen beim Laden ignoriert. Zuvor erscheint pro Rezepturdatei eine Meldung mit den betroffenen Variablen.
- Wenn in den Rezepturdateien auf der Steuerung Rezepturvariablen fehlen, die in der Rezepturdefinition im Projekt enthalten sind, erscheint pro Rezepturdatei eine Meldung mit den betroffenen Variablen.
- Wenn auf der Steuerung zusätzliche Rezepturen für diese Variablen angelegt wurden, werden diese der Rezepturdefinition im Projekt hinzugefügt.

17.18 Bibliothek

Befehl	Weiterführende Informationen in der PLC-Dokumentation
Bibliothekserstellung	
Befehl Als Bibliothek speichern...	Befehl Als Bibliothek speichern [▶ 282]
Befehl Als Bibliothek speichern und installieren...	Befehl Als Bibliothek speichern und installieren [▶ 284]
Bibliotheksinstallation	
Befehl Bibliotheksrepository	Bibliotheksrepository [▶ 285]
Bibliotheksverwaltung	
Dialog Bibliotheksverwalter	Bibliotheksverwalter [▶ 289]
Weitere Befehle und Dialoge	
Befehl Bibliothek hinzufügen	Befehl Bibliothek hinzufügen [▶ 373]
Befehl Bibliothek ohne Auflösung hinzufügen	Befehl Bibliothek ohne Platzhalterauflösung hinzufügen [▶ 374]
Befehl Bibliothek erneut laden	Befehl Bibliothek erneut laden [▶ 375]
Befehl Bibliothek entfernen	Befehl Bibliothek entfernen [▶ 376]
Befehl Details	Befehl Details [▶ 376]
Befehl Abhängigkeiten	Befehl Abhängigkeiten [▶ 377]
Befehl Eigenschaften	Befehl Eigenschaften [▶ 377]
Befehl Platzhalter	Platzhalter [▶ 294]
Befehl Effektive Version verwenden	Befehl Effektive Version verwenden [▶ 380]
Befehl Immer neueste Version verwenden	Befehl Immer neueste Version verwenden [▶ 381]

Siehe auch:

- Dokumentation PLC: Bibliotheken verwenden > Weitere Befehle und Dialoge

17.19 Visualisierung

Die Visualisierungsbefehle werden vom Plug-in **Visual Editor** für die Menükategorie Visualisierungskommandos zur Verfügung gestellt, die im Dialog **Extras > Anpassen** zu finden sind. Sie ermöglichen es, ein Visualisierungsobjekt im Visualisierungseditor zu editieren.

Die meisten sind standardmäßig Teil des Menüs **Visualisierung** und deswegen auch im Kontextmenü des Visualisierungseditors vorhanden. Wenn nötig, öffnen Sie den Dialog **Extras > Anpassen**, um die Menükonfiguration der Kategorie **Visualisierung** anzusehen oder zu ändern.

17.19.1 Befehl Schnittstellen-Editor

Symbol: 

Funktion: Dieser Befehl öffnet den Schnittstelleneditor, um Frameparameter in der Visualisierung zu definieren, die dazu bestimmt sind, im Element **Frame** einer anderen Visualisierung referenziert zu werden. Er wird in einer Registerblattansicht im oberen Teil des Visualisierungseditors angezeigt.

Aufruf: Menü **Visualisierung**

17.19.2 Befehl Tastaturkonfiguration

Symbol: 

Funktion: Dieser Befehl öffnet den Konfigurationseditor für die Tastaturbedienung für die aktuelle Visualisierung. Er wird in einer Registerblattansicht im oberen Teil des Visualisierungseditors angezeigt.

Aufruf: Menü **Visualisierung**

17.19.3 Befehl Elementliste

Symbol: 

Funktion: Dieser Befehl öffnet den Elementlisteneditor für die aktuelle Visualisierung. Er wird in einer Registerblattansicht im oberen Teil des Visualisierungseditors angezeigt.

Aufruf: Menü **Visualisierung**

17.19.4 Befehl Links ausrichten

Symbol: 

Funktion: Dieser Befehl bewirkt, dass alle ausgewählten Visualisierungselemente am linken Rand ihres am weitesten links positionierten Elements ausgerichtet werden.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.5 Befehl Oben ausrichten

Symbol: 

Funktion: Dieser Befehl bewirkt, dass alle ausgewählten Visualisierungselemente am oberen Rand ihres am weitesten oben positionierten Elements ausgerichtet werden.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.6 Befehl Rechts ausrichten

Symbol: 

Funktion: Dieser Befehl bewirkt, dass alle ausgewählten Visualisierungselemente am rechten Rand ihres am weitesten rechts positionierten Elements ausgerichtet werden.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.7 Befehl Unten ausrichten

Symbol: 

Funktion: Dieser Befehl bewirkt, dass alle ausgewählten Visualisierungselemente am unteren Rand ihres am weitesten unten positionierten Elements ausgerichtet werden.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.8 Befehl Vertikal zentrieren

Symbol: 

Funktion: Dieser Befehl bewirkt, dass alle ausgewählten Visualisierungselemente an ihrer gemeinsamen vertikalen Mitte ausgerichtet werden.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.9 Befehl Horizontal zentrieren

Symbol: 

Funktion: Dieser Befehl bewirkt, dass alle ausgewählten Visualisierungselemente an ihrer gemeinsamen horizontalen Mitte ausgerichtet werden.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.10 Befehl Horizontalen Abstand gleichmachen

Symbol: 

Der Befehl wird aktiv, wenn drei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die horizontal mit gleichem Abstand positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Horizontalen Abstand gleichmachen** aus.

⇨ Die Elemente werden so positioniert, dass das linkeste Element und das rechteste Element seine Position behalten und die Elemente dazwischen horizontal mit gleichem Abstand ausgerichtet werden.

17.19.11 Befehl Horizontalen Abstand vergrößern

Symbol: 

Der Befehl wird aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die horizontal mit mehr Abstand positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.

2. Führen Sie den Befehl **Horizontalen Abstand vergrößern** aus.

⇨ Die Elemente werden so positioniert, dass das blaue Element seine Position behält und die weiteren Elemente horizontal mit mehr Abstand zwischen den Elementen ausgerichtet werden. Der Abstand vergrößert sich um 1 Pixel.

17.19.12 Befehl Horizontalen Abstand verkleinern

Symbol: 

Der Befehl wird aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die horizontal mit weniger Abstand positioniert werden sollen.
2. Führen Sie den Befehl **Horizontalen Abstand verkleinern** aus.

⇨ Die Elemente werden so positioniert, dass das blaue Element seine Position behält und die weiteren Elemente horizontal mit weniger Abstand zwischen den Elementen ausgerichtet werden. Der Abstand verkleinert sich um 1 Pixel.

17.19.13 Befehl Horizontalen Abstand entfernen

Symbol: 

Der Befehl wird aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die horizontal ohne Abstand zueinander positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Horizontalen Abstand entfernen** aus.

⇨ Die Elemente werden so positioniert, dass das blaue Element seine Position behält und die weiteren Elemente horizontal ohne Abstand zueinander ausgerichtet werden.

17.19.14 Befehl Vertikalen Abstand gleichmachen

Symbol: 

Der Befehl wird aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die vertikal mit gleichem Abstand positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Vertikalen Abstand gleichmachen** aus.

⇨ Die Elemente werden so positioniert, dass das oberste Element und das unterste Element seine Position behalten und die Elemente dazwischen vertikal mit gleichem Abstand ausgerichtet werden.

17.19.15 Befehl Vertikalen Abstand vergrößern

Symbol: 

Der Befehl wird aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die vertikal mit mehr Abstand positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Vertikalen Abstand vergrößern** aus.

⇨ Die Elemente werden so positioniert, dass das blaue Element seine Position behält und die weiteren Elemente vertikal mit mehr Abstand zwischen den Elementen ausgerichtet werden. Der Abstand vergrößert sich um 1 Pixel.

17.19.16 Befehl Vertikalen Abstand verkleinern

Symbol: 

Der Befehl ist aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die vertikal mit weniger Abstand positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Vertikalen Abstand verkleinern** aus.

⇨ Die Elemente werden so positioniert, dass das blaue Element seine Position behält und die weiteren Elemente vertikal mit weniger Abstand zwischen den Elementen ausgerichtet werden. Der Abstand verkleinert sich um 1 Pixel.

17.19.17 Befehl Vertikalen Abstand entfernen

Symbol: 

Der Befehl wird aktiv, wenn zwei oder mehr Elemente selektiert sind.

1. Selektieren Sie alle Elemente, die vertikal ohne Abstand zueinander positioniert werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Vertikalen Abstand** entfernen aus.

⇨ Die Elemente werden so positioniert, dass das blaue Element seine Position behält und die weiteren Elemente vertikal ohne Abstand zueinander ausgerichtet werden.

17.19.18 Befehl Breite gleichmachen

Symbol: 

Der Befehl wird aktiv, wenn mehr als ein Element selektiert ist, außer ein Linien- oder Polygonelement ist selektiert.

1. Selektieren Sie alle Elemente, die die gleiche Breite bekommen sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Breite gleichmachen** aus.

⇨ Alle Elemente bekommen die Breite des blau markierten Elements.

17.19.19 Befehl Höhe gleichmachen

Symbol: 

Der Befehl wird aktiv, wenn mehr als ein Element selektiert ist, außer ein Linien- oder Polygonelement ist selektiert.

1. Selektieren Sie alle Elemente, die die gleiche Höhe bekommen sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Höhe gleichmachen** aus.

⇨ Alle Elemente bekommen die Höhe des blau markierten Elements.

17.19.20 Befehl Größe gleichmachen

Symbol: 

Der Befehl ist aktiv, wenn mehr als ein Element selektiert ist, außer ein Linien- oder Polygonelement ist selektiert.

1. Selektieren Sie alle Elemente, die die gleiche Größe bekommen sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Größe gleichmachen** aus.

⇨ Alle Elemente bekommen die Größe des blau markierten Elements.

17.19.21 Befehl Größe an Raster anpassen

Symbol: 

Der Befehl wird aktiv, wenn mehr als ein Element selektiert ist, außer ein Linien- oder Polygonelement ist selektiert.

1. Selektieren Sie alle Elemente, die in Position und Größe am Raster ausgerichtet werden sollen.
   ⇨ Das erste Element ist blau markiert, die weiteren Elemente sind grau markiert.
2. Führen Sie den Befehl **Größe an Raster** anpassen aus.

⇨ Alle Elemente werden in Größe und Position am Raster ausgerichtet.

17.19.22 Befehl Um Eins nach vorn legen

Symbol: 

Funktion: Dieser Befehl positioniert das ausgewählte Element eine Ebene höher, d. h. weiter in den Vordergrund der Visualisierung. Elemente auf tieferen Ebenen werden von denen auf höheren verdeckt.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.23 Befehl Nach vorn legen

Symbol: 

Funktion: Dieser Befehl positioniert das ausgewählte Element im Vordergrund der Visualisierung, d. h. auf höchster Ebene. Elemente auf tieferen Ebenen werden von denen auf höheren verdeckt.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.24 Befehl Um Eins nach hinten legen

Symbol: 

Funktion: Dieser Befehl positioniert das ausgewählte Element eine Ebene tiefer, d. h. weiter in den Hintergrund der Visualisierung. Elemente auf tieferen Ebenen werden von denen auf höheren verdeckt.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.25 Befehl Nach hinten legen

Symbol: 

Funktion: Dieser Befehl positioniert das ausgewählte Element im Hintergrund der Visualisierung, d. h. auf tiefster Ebene. Elemente auf tieferen Ebenen werden von denen auf höheren verdeckt.

Aufruf: Menü **Visualisierung**, Kontextmenü

17.19.26 Befehl Gruppieren

Symbol: 

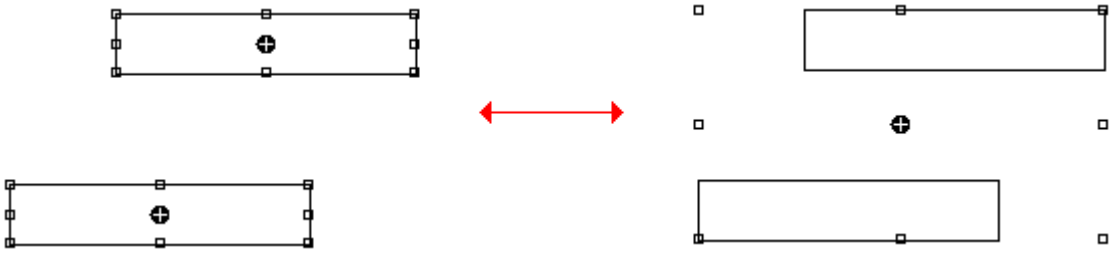
Funktion: Mit diesem Befehl werden die gerade ausgewählten Visualisierungselemente gruppiert und die Gruppe als Einzelobjekt selektiert dargestellt.

Aufruf: Menü **Visualisierung**, Kontextmenü

Voraussetzung: Mehrere Visualisierungselemente sind ausgewählt. Für die Mehrfachauswahl halten Sie die **[<Umschalt>]**-Taste gedrückt, während Sie die gewünschten Elemente anklicken. Alternativ können Sie außerhalb eines Elements ins Editorfenster klicken und bei gedrückter Maustaste ein Rechteck um die gewünschten Elemente ziehen.

Zum Auflösen der Gruppe verwenden Sie Befehl **Gruppierung aufheben**.

Die folgende Abbildung zeigt die Gruppierung (von links nach rechts) bzw. das Aufheben der Gruppierung (von rechts nach links) zweier Rechteck-Elemente:



- Befehl Gruppierung aufheben [▶ 1109]

17.19.27 Befehl Gruppierung aufheben

Symbol: 

Funktion: Mit diesem Befehl wird eine Gruppe von Visualisierungselementen wieder aufgelöst. Die einzelnen Elemente werden jeweils wieder einzeln selektiert dargestellt.

Aufruf: Menü **Visualisierung**, Kontextmenü

Siehe auch:

- Gruppieren [▶ 1109]

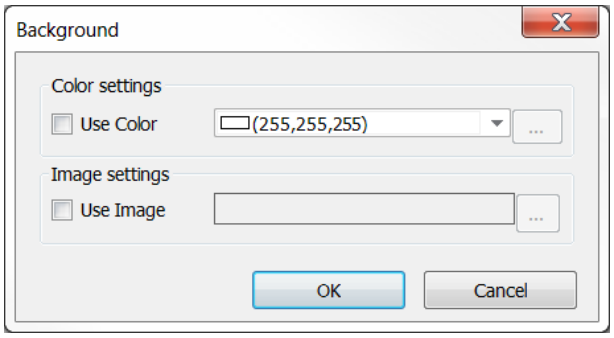
17.19.28 Befehl Hintergrund

Symbol: 

Funktion: Dieser Befehl öffnet den Dialog **Konfiguration der Framevisualisierung**, in dem eine Farbe und/oder eine Abbildung für den Visualisierungshintergrund ausgewählt werden kann.

Aufruf: Menü **Visualisierung**, Kontextmenü

Dialog Hintergrund



Aktivieren Sie die gewünschte(n) Option(en):

- Bitmap: Um ein Hintergrundbild zu definieren, muss hier der Pfad einer Bilddatei, die in einer Bildersammlung im Projekt verfügbar ist, eingetragen werden. Geben Sie dazu den Namen der Bildersammlung und die ID der Bilddatei - getrennt durch einen Punkt „.“ - ein: <bildersammlung>.<ID> (z. B. „Images_1.drive_icon>“, „Images_1.43“).
- Farbe: Um die Hintergrundfarbe der Visualisierung zu definieren, wählen Sie aus der Farbauswahlliste die gewünschte Farbe aus.

17.19.29 Befehl Alles auswählen

Symbol: /

Funktion: Dieser Befehl wählt alle Elemente der gerade im Editor geöffneten Visualisierung aus.

Aufruf: Menü **Visualisierung**, Kontextmenü

Siehe auch:

- Alles deselektieren [▶ 1110]

17.19.30 Befehl Alles deselektieren

Symbol:

Funktion: Dieser Befehl hebt die aktuelle Selektion von Visualisierungselementen auf.

Aufruf: Menü **Visualisierung**

Siehe auch:

- Befehl Alles auswählen [▶ 1110]

17.19.31 Befehl Visualisierungselement vervielfachen

Symbol:

Funktion: Der Befehl öffnet den Dialog **Visualisierungselement vervielfachen**, mit dem Sie das Vervielfachen konfigurieren können.

Aufruf: Menü **Visualisierung** , Kontextmenü

Voraussetzung: Die Visualisierung ist aktiv und ein Vorlagenelement ist selektiert.

Dialog Visualisierungselement vervielfachen

TwinCAT fügt beim Vervielfachen weitere Elemente ein, die dem Vorlagenelement gleichen. Sie können in diesem Dialog die Anzahl und die Anordnung, sowie die Ersetzung der Indizes konfigurieren.

Registerkarte Grundeinstellung

Gesamtzahl Elemente:

TwinCAT fügt die neuen Elemente als Tabelle ein. Die Zeilenanzahl ist in **Horizontal** eingestellt, die Spaltenanzahl in **Vertikal**. Das Produkt der beiden bestimmt die tatsächliche Gesamtzahl der einzufügenden Elemente.	
Horizontal	Anzahl an Elementen pro Zeile Voreinstellung: Entspricht der Anzahl der Komponenten in $FIRSTDIM$
Vertikal	Anzahl an Elementen pro Spalte Voreinstellung: Entspricht der Anzahl der Komponenten in $SECONDDIM$

Offset zwischen den Elementen:

TwinCAT ordnet die Elemente in der Visualisierung als Tabelle an. Wenn Sie einen Offset angeben, wird zwischen den Elementen ein Abstand eingefügt. • 0 : Die Rahmen der Elemente überlappen um ein Pixel • 1 : Die Elemente stoßen aneinander • <n> : Zwischen den Elementen ist ein sichtbarer Abstand von n-1 Pixel	
Horizontal	Zeilenabstand der Elemente in Pixel
Vertikal	Spaltenabstand der Elemente in Pixel

Registerkarte Erweiterte Einstellung

Erste Dimension:

Startindex	Startindex für $FIRSTDIM$ Voreinstellung: 1, bedeutet, das der konkrete Index für $FIRSTDIM$ mit 1 beginnt. Beispiel Array[1, <$SECONDDIM$>]
Hochzählen	Inkrement, um das der Index hochgezählt wird Voreinstellung: 1

Zweite Dimension:

Startindex	Startindex für $SECONDDIM$ Voreinstellung: 1, bedeutet, das der konkrete Index für $SECONDDIM$ mit 1 beginnt . Beispiel Array[<$FIRSTDIM$>, 1] relevant sind
Hochzählen	Inkrement, um das der Index hochgezählt wird Voreinstellung: 1

17.19.32 Befehl Tastaturbedienung aktivieren

Symbol: 

Funktion: Dieser Befehl ist in der Menüleiste für eine integrierte Visualisierung (Diagnosevisualisierung) verfügbar. Er aktiviert bzw. deaktiviert die Tastaturbedienung im Onlinebetrieb einer Visualisierung.

Aufruf: Menü **Visualisierung**

Wenn die Tastaturbedienung aktiviert ist, können Eingaben auf Elemente und die Auswahl der Elemente über bestimmte Tastenkombinationen vorgenommen werden. In diesem Fall werden andere über Tastatur gegebene Befehle nicht ausgeführt, solange der Visualisierungseditor aktiv und im Onlinebtrieb ist.

17.20 Sonstiges

17.20.1 Befehl Schnittstellen implementieren

Funktion: Der Befehl aktualisiert für einen Funktionsbaustein die implementierten Schnittstellen, indem die Schnittstellenelemente hinzugefügt werden, die der Funktionsbaustein aktuell nicht enthält.

Aufruf: Kontextmenü, wenn der Funktionsbaustein im SPS-Projektbaum selektiert ist.

Voraussetzung: Der Funktionsbaustein implementiert eine Schnittstelle, die Sie verändert haben. Beispielsweise haben Sie der Schnittstelle eine weitere Methode hinzugefügt.

Anwendungsfälle

Bei Ausführung dieses Befehls werden die automatisch angelegten Methoden oder Eigenschaften mit einem Pragmaattribut versehen, welches Übersetzungsfehler oder -warnungen provoziert. Dadurch werden Sie dahingehend unterstützt, dass automatisch angelegte Elemente nicht unbeabsichtigt leer bleiben. Ob ein error- oder warning-Attribut verwendet wird, hängt vom Anwendungsfall ab.

Fall 1:

Situation: Der Funktionsbaustein, für den der Befehl **Schnittstellen implementieren** ausgeführt wird, ist **nicht** von einem anderen Funktionsbaustein abgeleitet.

Konsequenz: Bei Ausführung des Befehls werden die Schnittstellenelemente in dem Funktionsbaustein ohne Implementierung angelegt („Stubs“) und mit einem warning-Attribut versehen (in der ersten Zeile der Methoden-/Eigenschaftendeklaration). Durch die Warnungen, die beim Kompilieren erzeugt werden, werden Sie darauf aufmerksam gemacht, dass diese Elemente automatisch erzeugt wurden und der gewünschte Implementierungscode hinzugefügt werden muss.

```
{warning 'add method/property implementation'}
```

Vorgehen: Fügen Sie dem jeweiligen Schnittstellenelement (Methode oder Eigenschaft) den gewünschten Implementierungscode hinzu. Entfernen Sie anschließend das warning-Attribut aus der Methoden- bzw. Eigenschaftendeklaration.

Fall 2:

Situation: Der Funktionsbaustein, für den der Befehl Schnittstellen implementieren ausgeführt wird, ist von einem anderen Funktionsbaustein abgeleitet. Das Element (Methode oder Eigenschaft), das bei Ausführung des Befehls im abgeleiteten Funktionsbaustein angelegt wird, wird **nicht** über Vererbung von dem Basis-Funktionsbaustein zur Verfügung gestellt (d. h. das Element ist nicht unterhalb des Basis-Funktionsbausteins oder einer höheren Elternklasse vorhanden).

Konsequenz/Vorgehen: s. Fall 1.

Fall 3:

Situation: Der Funktionsbaustein, für den der Befehl **Schnittstellen implementieren** ausgeführt wird, ist von einem anderen Funktionsbaustein abgeleitet. Das Element (Methode oder Eigenschaft), das bei Ausführung des Befehls im abgeleiteten Funktionsbaustein angelegt wird, wird bereits über Vererbung von dem Basis-Funktionsbaustein zur Verfügung gestellt (d. h. das Element ist unterhalb des Basis-Funktionsbausteins oder einer höheren Elternklasse vorhanden).

Konsequenz: Bei Ausführung des Befehls wird das Schnittstellenelement in dem abgeleiteten Funktionsbaustein ohne Implementierung angelegt („Stub“) und mit einem error-Attribut versehen (in der ersten Zeile der Methoden-/Eigenschaftendeklaration). Durch den Fehler, der beim Kompilieren erzeugt wird, werden Sie darauf aufmerksam gemacht, dass dieses Schnittstellenelement automatisch erzeugt wurde und dass diese Methode oder Eigenschaft das entsprechende Element des Basis-Funktionsbausteins überschreibt.

```
{error 'add method/property implementation or delete method/property to use base implementation'}
```

Vorgehen: Falls Sie die Methode oder Eigenschaft des Basis-Funktionsbausteins überschreiben oder erweitern möchten, fügen Sie dem Element unterhalb des abgeleiteten Bausteins den gewünschten Implementierungscode hinzu. Entfernen Sie anschließend das error-Attribut aus der Methoden- bzw. Eigenschaftendeklaration. Falls Sie die Methode oder Eigenschaft des Basis-Funktionsbausteins hingegen **nicht** überschreiben möchten, löschen Sie die Methode oder Eigenschaft unterhalb des abgeleiteten Funktionsbausteins. Dadurch wird die Methoden- bzw. Eigenschaftenimplementierung des Basis-Funktionsbausteins verwendet.

17.21 Kontextmenü TwinCAT Projekt

17.21.1 Befehl Sichern <TwinCAT-Projektname> als Archiv…

Symbol: 

Funktion: Der Befehl öffnet den Standarddialog zum Speichern einer Datei als Archiv. Das Projekt kann unter dem gewünschten Speicherpfad als ***.tszip**-Archiv abgelegt werden.

Aufruf: Menü **Datei**, Kontextmenü

Voraussetzung: Das TwinCAT-Projekt ist **Projektmappen-Explorer** ausgewählt.

Inhalt vom *.tszip	Der Archivordner *.tszip enthält das TwinCAT-Projekt, welches archiviert wird.
Befehl zum Öffnen	Ein tszip-Archiv kann über den folgenden Befehl wieder geöffnet werden: Befehl Projekt/Projektmappe (Projekt/Projektmappe öffnen) [▶ 901]
Hinweis zu SPS-Projekten	Falls das TwinCAT-Projekt ein oder mehrere SPS-Projekte enthält, sind die Dateien und Ordner, die bezüglich dieser SPS-Projekte in dem Archivordner gespeichert werden, abhängig von den SPS-Projekteinstellungen des jeweiligen SPS-Projekts. Registerkarte Settings [▶ 969]

17.21.2 Befehl Sende per E-Mail…/Send by E-Mail...

Symbol: 

Funktion: Der Befehl startet das aktuell im System eingestellte E-Mail-Programm und öffnet eine neue E-Mail. Diese enthält im Anhang die Archivdatei des ausgewählten Projekts.

Aufruf: Menü **Datei**, Kontextmenü

17.21.3 Befehl Sichere <TwinCAT-Projektnamen> automatisch auf dem Zielsystem

Funktion: Über diesen Befehl können Sie aktivieren oder deaktivieren, dass das TwinCAT Projekt beim Aktivieren automatisch als .tszip auf dem Zielsystem abgelegt wird. Dies ist notwendig, wenn Sie das Projekt zu einem späteren Zeitpunkt mit dem Befehl Open Project from Target [▶ 909] vom Zielsystem laden und öffnen wollen.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

17.21.4 Befehl <TwinCAT-Projektname> mit dem Zielsystem vergleichen...

Funktion: Dieser Befehl ermöglicht, das selektierte TwinCAT Projekt mit dem Projektstand auf dem Zielsystem mit des TwinCAT Project Compare Tools zu vergleichen.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert und ein Zielsystem ist ausgewählt, auf welches bereits ein TwinCAT Projekt heruntergeladen worden ist.

Siehe auch:

- Befehl Projekt mit Zielsystem aktualisieren... [▶ 1114]

17.21.5 Befehl Projekt mit Zielsystem aktualisieren…

Funktion: Dieser Befehl ermöglicht, das selektierte TwinCAT Projekt auf den Projektstand des verbundenen Zielsystems zu aktualisieren. Dafür wird ein Popup-Dialog mit einer Liste der geänderten Dateien geöffnet, der bestätigt werden muss, um die Aktion erfolgreich durchzuführen. Ein detaillierter Vergleich ist hier nicht vorgesehen.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

Siehe auch:

- Befehl <TwinCAT-Projektname> mit dem Zielsystem vergleichen... [▶ 1114]

17.21.6 Befehl Projekt mit TwinCAT 2.xx Version laden…

Funktion: Dieser Befehl öffnet den Standard-Browserdialog, über den eine TwinCAT 2 System Manager-Datei ausgewählt und importiert werden kann. Auf diese Weise können Sie ein bestehendes TwinCAT 2 Projekt inklusive der System Manager-Einstellungen und des SPS-Projektes nach TwinCAT 3 konvertieren.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist ohne Änderungen neu angelegt worden und im Projektmappen-Explorer selektiert.

Siehe auch:

- TwinCAT-2-SPS-Projekt öffnen [▶ 58]

17.21.7 Befehl Verborgene Konfigurationen anzeigen

Funktion: Dieser Befehl öffnet ein Detailmenü, in dem die verborgenen Konfigurationen aufgelistet werden und wieder angezeigt werden können.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

17.21.8 Befehl Aus Projektmappe entfernen

Symbol: 

Funktion: Der Befehl ermöglicht es, das TwinCAT Projekt aus der Projektmappe zu löschen.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

17.21.9 Befehl Umbenennen

Symbol: 

Funktion: Der Befehl ermöglicht das Umbenennen des TwinCAT Projektes im **Projektmappen-Explorer**.

Aufruf: Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

17.21.10 Befehl TwinCAT-Projekt erstellen

Symbol: 

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codeerzeugung für das gerade aktive TwinCAT-Projekt.

Aufruf: Menü **Erstellen**, wenn aktuell ein TwinCAT-Projekt selektiert ist, oder Kontextmenü des TwinCAT-Projekts

Voraussetzung: Das TwinCAT-Projekt ist selektiert.

Alle in dem TwinCAT-Projekt enthaltenen Projekte (SPS, C++, etc.) werden der Reihe nach übersetzt. Die dabei für ein SPS-Projekt durchgeführten Schritte werden im Abschnitt Befehl SPS-Projekt erstellen [▶ 973] beschrieben.

Siehe auch:

- Befehl TwinCAT-Projekt neu erstellen [▶ 1115]

17.21.11 Befehl TwinCAT-Projekt neu erstellen

Funktion: Der Befehl startet den Übersetzungsprozess bzw. die Codeerzeugung für das gerade aktive TwinCAT-Projekt, auch wenn es zuletzt fehlerfrei übersetzt wurde.

Aufruf: Menü **Erstellen**, wenn aktuell ein TwinCAT-Projekt selektiert ist, oder Kontextmenü des TwinCAT-Projekts

Voraussetzung: Das TwinCAT-Projekt ist selektiert.

Bei einer Neuerstellung des Projekts wird das TwinCAT-Projekt zunächst bereinigt (siehe auch: Befehl TwinCAT-Projekt bereinigen [▶ 1115]) und anschließend erstellt (siehe auch: Befehl TwinCAT-Projekt erstellen [▶ 1115]).

17.21.12 Befehl TwinCAT-Projekt bereinigen

Funktion: Der Befehl löscht die lokale Übersetzungsinformation für das gerade aktive SPS-Projekt und aktualisiert das Sprachmodell aller Objekte.

Aufruf: Menü **Erstellen**, wenn aktuell ein TwinCAT-Projekt selektiert ist, oder Kontextmenü des TwinCAT-Projekts

Voraussetzung: Das TwinCAT-Projekt ist selektiert.

Alle in dem TwinCAT-Projekt enthaltenen Projekte (SPS, C++, etc.) werden der Reihe nach bereinigt. Die dabei für ein SPS-Projekt durchgeführten Schritte werden im Abschnitt Befehl SPS-Projekt bereinigen [▶ 973] beschrieben.

Siehe auch:

- Befehl TwinCAT-Projekt neu erstellen [▶ 1115]

17.21.13 Befehl Projekt entladen

Funktion: Dieser Befehl entlädt das TwinCAT Projekt, sodass alle Dateien dieses TwinCAT Projekts freigegeben sind.

Aufruf: Menü Projekt oder Kontextmenü des TwinCAT Projektes

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

17.21.14 Import AutomationML via AML DataExchange...

Funktion: Der Befehl öffnet den Standard-Browse-Dialog, über den eine Datei im AutomationML-Format gesucht und importiert werden kann.

Aufruf: Der Befehl kann über das Kontextmenü des TwinCAT Projekts unter **Import AutomationML** oder über den **TwinCAT** Eintrag in der Menüleiste unter **AutomationML** und **Import AutomationML** aufgerufen werden.

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

Siehe auch

- Befehl: Open AML DataExchange Log (local)

17.21.15 Export AutomationML...

Funktion: Der Befehl öffnet den Standard-Browserdialog zum Speichern einer Datei im AutomationML-Format, um die unter dem Knoten E/A vorhandene Topologie zu exportieren.

Aufruf: Der Befehl kann über das Kontextmenü des TwinCAT Projekts oder über den **TwinCAT** Eintrag in der Menüleiste unter **AutomationML** aufgerufen werden.

Voraussetzung: Das TwinCAT Projekt ist im Projektmappen-Explorer selektiert.

18 SPS-Programmierkonventionen

In der Welt der Programmiersprachen gibt es verschiedene Konventionen, die abhängig vom Tooling, Fokus und Ursprung der Programmiersprache sowie von der Branche, in der die Anwendung verwendet wird, Beachtung finden.

In Anlehnung an allgemein anerkannte Programmierrichtlinien (zum Beispiel die MISRA C++ 2008 – Guidelines[…], die Java Code Conventions, den C# Programming Guide oder PLCopen) finden Sie im Folgenden Programmierhinweise, die die Erstellung von optimalem und funktionstüchtigem SPS-Programmcode erleichtern.

Beachten Sie die SPS-Programmierkonventionen, um die folgenden Vorteile zu erreichen:

- einheitlicher Aufbau der SPS-Programme
- konsistente Benennung von Objekten, Variablen und Instanzen
- leichte Lesbarkeit und intuitive Verständlichkeit von Code und insbesondere von den Schnittstellen von Funktionsbausteinen, Methoden und Funktionen
- vereinfachte Entwicklung, Nutzung und Wartung der Programme
- erhöhte Codequalität durch die systematische und frühzeitige Vermeidung von Fehlerquellen

Neue Beckhoff TwinCAT 3 SPS Bibliotheken, Programmbeispiele und Applikationen richten sich nach diesen „TwinCAT 3 SPS-Programmierkonventionen“.

Ausnahmen:

- Tc2_MC2_xxx Bibliotheken entsprechen der PLCopen-Programmierkonvention.
- Tc2_xxx Bibliotheken entsprechen den ursprünglichen TwinCAT-2-Bibliotheken.
- Branchenspezifische Bibliotheken richten sich im Einzelfall nach dem Standard der Branche.
- Bausteine aus der IEC 61131-Norm, die Bestandteil der Bibliothek Tc2_Standard sind.

Klassifizierung

Die SPS-Programmierkonventionen sind in die folgenden Kapitel unterteilt:

- Programmierstil [▶ 1119]
- Namenskonventionen [▶ 1129]
- Programmierung [▶ 1139]

Innerhalb jedes Kapitels gibt es Unterkapitel, welche wiederum eine Auflistung von Themenpunkten beinhalten.

Die einzelnen Themenpunkte sind klassifiziert als

- Vorschrift, gekennzeichnet mit [++], oder als
- Empfehlung, gekennzeichnet mit [+].

Als Vorschrift klassifizierte Themenpunkte müssen Sie zwingend umsetzen. Empfehlungen sollten möglichst immer umgesetzt werden, weil sie aus verschiedenen Gründen ratsam sind.

Statische Codeanalyse

Um das Risiko von Laufzeitfehlern zu reduzieren, sollte der Code mindestens mithilfe der kostenfreien Variante der statischen Codeanalyse (Static Analysis Light [▶ 158]) überprüft werden. Diese ist ab TwinCAT 3.1 Build 4018 enthalten und enthält einen minimalen Umfang zur Codeüberprüfung. Mit Hilfe des vollen Funktionsumfangs von TE1200 PLC Static Analysis ist darüber hinaus eine detailliertere Überprüfung der Programmierkonventionen möglich (siehe folgenden Abschnitt).

Überprüfung der Programmierkonventionen mit Hilfe von TE1200 PLC Static Analysis

TE1200 PLC Static Analysis bietet die Möglichkeit zu vielseitigen Prüfungen des SPS-Programmcodes. Mit Hilfe dieser Funktionalitäten kann die Einhaltung vieler Vorschriften, Empfehlungen und Namenskonventionen der SPS-Programmierkonventionen überprüft werden.

Unterkapitel Programmierung

Soweit vorhanden wird zu jeder Vorschrift/Empfehlung die entsprechende Regel seitens Static Analysis genannt, mit Hilfe derer die Einhaltung der Vorschrift/Empfehlung überprüft werden kann.

Siehe auch:

- Unterkapitel Programmierung [▶ 1139]
- Funktionalität Regeln vom TE1200 PLC Static Analysis

Unterkapitel Namenskonventionen

Soweit vorhanden wird zu jeder Namenskonvention die ID des Static Analysis Konfigurationsfeldes genannt, in das der jeweilige empfohlene Namenspräfix eingetragen werden muss. Dadurch kann ein Großteil der empfohlenen Präfixe für POUs, DUTs, Variablen und Instanzen vom Static Analysis überprüft werden.

Siehe auch:

- Unterkapitel Namenskonventionen [▶ 1129]
- Funktionalität Namenskonventionen vom TE1200 PLC Static Analysis

Pragmas und Attribute

Beachten Sie auch die Möglichkeit, Static Analysis Regeln und Namenskonventionen für überprüfte Stellen, an denen eine bestimmte Regel/Namenskonvention nicht mehr gemeldet werden soll, lokal per Pragma oder Attribut zu deaktivieren (siehe auch: Kapitel Pragmas und Attribute in TE1200 PLC Static Analysis). Eine lokale Deaktivierung kommentieren Sie optimalerweise mit entsprechender Begründung.

Download Static Analysis Konfigurationsdatei

1) Vorschriften und Empfehlungen

Über den folgenden Link können Sie eine vorgefertigte Static Analysis Konfigurationsdatei herunterladen, in der die Vorschriften [++] und Empfehlungen [+] der Programmierkonventionen als Regeln aktiviert sind. Ein Verstoß gegen eine Vorschrift wird nach Ausführung des Static Analysis als Fehler ausgegeben, ein Verstoß gegen eine Empfehlung ist als Warnung klassifiziert.

https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/12663400331.zip

2) Vorschriften, Empfehlungen und Namenskonventionen

Über den folgenden Link können Sie eine vorgefertigte Static Analysis Konfigurationsdatei herunterladen, in der die Vorschriften [++], Empfehlungen [+] und Namenskonventionen der Programmierkonventionen als Regeln aktiviert sind. Ein Verstoß gegen eine Vorschrift oder eine Namenskonvention wird nach Ausführung des Static Analysis als Fehler ausgegeben, ein Verstoß gegen eine Empfehlung ist als Warnung klassifiziert.

https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/12663402507.zip

Weitere Regeln

Zusätzlich zu den Regeln, die innerhalb der folgenden Programmierkonventionen explizit erwähnt werden, bietet das TE1200 viele weitere nützliche Regeln, mit denen Flüchtigkeitsfehler oder allgemeine Programmierfehler frühzeitig aufgedeckt werden können. Dazu zählen u.a. die folgenden Regeln.

- SA0008: Unterbereichstypen prüfen
- SA0041: Möglicherweise schleifeninvarianter Code
- SA0044: Deklarationen mit Schnittstellenreferenz
- SA0056: Konstante außerhalb des gültigen Bereichs
- SA0059: Vergleichsoperationen, die immer TRUE oder FALSE liefern
- SA0060: Null als ungültiger Operand
- SA0062: Verwendung von TRUE und FALSE in Ausdrücken
- SA0140: Auskommentierte Anweisungen

18.1 Programmierstil

In diesem Abschnitt der SPS-Programmierkonventionen werden die folgenden Themenpunkte behandelt.

Schrift- und Editoreinstellungen

1. Gleiche Schrifteinstellungen verwenden [▶ 1119] [+]

Sprache

1. Englische Sprache verwenden [▶ 1120] [+]
2. Gültige Zeichen beachten [▶ 1120] [+]

Projektstruktur

1. Modularer Projektaufbau [▶ 1120] [+]
2. Ordnerstruktur [▶ 1120] [+]
3. Sortierungsschema im Projektbaum [▶ 1121] [+]
4. Keine unbenutzten Deklarationen/Objekte oder unnützen Code [▶ 1121] [+]

Programmstruktur

1. Objektorientierte Programmierung [▶ 1122] [+]
2. Strukturierung von Programmelementen [▶ 1122] [+]
3. Strukturierung von Textblöcken [▶ 1122] [+]
4. Kommentare [▶ 1122] [+]

18.1.1 Schrift- und Editoreinstellungen

Themenpunkte:

1. Gleiche Schrifteinstellungen verwenden [▶ 1119] [+]

Gleiche Schrifteinstellungen verwenden

Um in unterschiedlichen Programmen ein einheitliches Schriftbild zu erreichen, sollten identische Schrifteinstellungen verwendet werden. Bei der Integration von TwinCAT 3 ins Visual Studio werden u.a. die Schriftart und die Tabulatorgröße standardmäßig voreingestellt.

Einstellungen:

Extras > Optionen > Umgebung > Schriftarten und Farben	
Schriftart	Consolas
Schriftgrad	10
Extras > Optionen > TwinCAT > SPS Programmierumgebung > Texteditor	
Gliederung	Einrückung
Wortumbruch	Keine Umbrüche
Tabulatorgröße	4
Tabs beibehalten	Ja
Einzugsgröße	4
Automatisch einrücken	Intelligent mit Code-Komplettierung

18.1.2 Sprache

Themenpunkte:

1. Englische Sprache verwenden [▶ 1120] [+]
2. Gültige Zeichen beachten [▶ 1120] [+]

Englische Sprache verwenden

Ein SPS-Programm (Programmcode, Variablennamen, Kommentare etc.) sollte vollständig in Englisch verfasst werden. Empfohlen wird hierbei amerikanisches Englisch.

Gültige Zeichen beachten

Die Namen von Programmelementen und Variablen dürfen lediglich die folgenden Buchstaben, Zahlen und Sonderzeichen enthalten:

- 0...9
- A...Z
- a...z
- Unterstrichzeichen '_' als einzig gültiges Trennzeichen (weitere Hinweise siehe unten)

Doppelte Unterstriche ('__') dürfen generell nicht verwendet werden, da sie für interne Variablen genutzt werden.

Verwendung des Unterstrichzeichens '_':

Das Unterstrichzeichen '_' als einzig gültiges Trennzeichen ist für folgende Benennungen vorgesehen:

- Bei Objekten zwischen Präfix und Objektnamen. Beispiele:

```
FB_GetData, ST_BufferEntry, I_CylinderControl, E_StandardColor
```

- Optional zur Umsetzung eines Sortierungsschemas [▶ 1121] von Objekten im Projektbaum

18.1.3 Projektstruktur

Themenpunkte:

1. Modularer Projektaufbau [▶ 1120] [+]
2. Ordnerstruktur [▶ 1120] [+]
3. Sortierungsschema im Projektbaum [▶ 1121] [+]
4. Keine unbenutzten Deklarationen/Objekte oder unnützen Code [▶ 1121] [+]

Modularer Projektaufbau

Ein TwinCAT 3 SPS-Projekt bauen Sie modular auf. Die Ordnerstruktur sollte sich dabei an den verschiedenen Funktionalitäten und Objekten eines SPS-Projekts orientieren. Die Programmelemente sortieren Sie bei Bedarf nach einem festen Schema.

Ordnerstruktur

Die Ordnerstruktur eines TwinCAT 3 SPS-Projekts sollte modular aufgebaut sein und sich an den verschiedenen Funktionalitäten/Objekten eines SPS-Projekts orientieren. Das SPS-Projekt teilen Sie auf der ersten Ebene in Modulordner ein. Auf der zweiten Ebene können Sie je nach Komplexität der Module eine feinere Modularisierung oder eine Ordnung nach Elementart (DUTs, ITFs, POUs etc.) vornehmen.

- Ordnernamen der ersten Ebene: Beschreibung der Funktionalität/des Moduls
- Ordnernamen der zweiten Ebene: z.B. weitere Modularisierung oder Beschreibung der Programmelemente

Beispiel:

Tc3_Plc_Project

+ [Topic X]
  + DUTs
  + ITFs
  + POUs
+ [Topic Y]
  + [Topic Y, Part a]
    + DUTs
    + POUs
  + [Topic Y, Part b]
    + ITFs
    + POUs

Sortierungsschema im Projektbaum

Die Objekte im Projektbaum werden von TwinCAT 3 alphabetisch sortiert. Ein optionales Schema zur Sortierung der Objekte im Projektbaum ergibt sich daher aus den Bezeichnungen der Objekte.

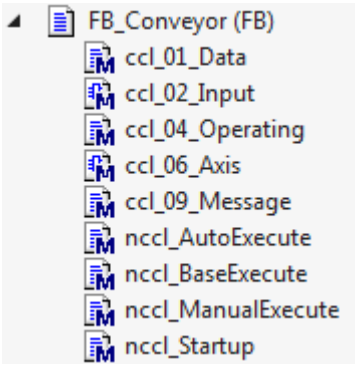
Das Sortierungsschema kann sich aus einer Kombination aus Zahlen, Stichwörtern oder deren Abkürzungen zusammensetzen. Es befindet sich zwischen dem Präfix und dem eigentlichen Namen des Objekts und wird jeweils durch ein Unterstrichzeichen vom Präfix und vom Objektnamen abgegrenzt. Wenn das Sortierungsschema aus mehreren Elementen (z. B. Zahlen mit Stichwörtern oder deren Abkürzungen) besteht, können Sie die jeweiligen Unterelemente mit einem Unterstrichzeichen voneinander trennen.

Das Sortierungsschema sollte an einer entsprechenden Stelle im Programmelement kommentiert werden.

Alternativ können POU-Unterordner verwendet werden, um die Unterelemente einer POU thematisch zu sortieren.

Beispiel für ein Sortierungsschema:

Die Methoden des folgenden Funktionsbausteins werden mithilfe von Abkürzungen und Zahlen sortiert. Dabei ordnen die Abkürzungen die Methoden gemäß ihres Aufrufintervalls ('ccl' = cyclical / zyklisch; 'nccl' = noncyclical / azyklisch, z. B. ereignisgesteuerter Aufruf) und die Zahlen sortieren die zyklischen Methoden nach ihrer vorgesehenen Aufrufreihenfolge. Bei diesen Methoden handelt es sich um ein Beispiel. Die Stichwörter zur Sortierung sind frei wählbar und die Unterstriche zwischen Stichwörtern und Zahlen sind optional (z. B. 'ccl01_Data' oder 'ccl_01_Data').



Keine unbenutzten Deklarationen/Objekte oder unnützer Code

Nicht verwendete Programmelemente führen in einem Projekt schnell zu unübersichtlichen Programmbäumen/Code-Strukturen. Bei Wartungsarbeiten und Ergänzungen kann die Lesbarkeit des Codes stark erhöht werden, wenn das Projekt nur verwendete Programmelemente enthält.

Sehen Sie dazu auch den Themenpunkt Keine unbenutzten Deklarationen/Objekte oder unnützen Code [▶ 1147] im Abschnitt Programmierung [▶ 1139].

18.1.4 Programmstruktur

Themenpunkte:

1. Objektorientierte Programmierung [▶ 1122] [+]
2. Strukturierung von Programmelementen [▶ 1122] [+]
3. Strukturierung von Textblöcken [▶ 1122] [+]
4. Kommentare [▶ 1122] [+]

Objektorientierte Programmierung

Um die Vorteile der objektorientierten Programmierung zu nutzen, strukturieren Sie das SPS-Programm in Klassen. Anstelle einer Vielzahl von Funktionsbausteinen wird dann eine Auswahl an Klassen mit entsprechenden Methoden verwendet.

Nutzen und Anwenderfreundlichkeit einer objektorientierten Ausführung beurteilen Sie im Einzelfall.

Strukturierung von Programmelementen

Siehe: Strukturierung von Programmelementen [▶ 1123]

Strukturierung von Textblöcken

Siehe: Strukturierung von Textblöcken [▶ 1125]

Kommentare

Vermeiden Sie unnötige Kommentare so weit wie möglich. Stattdessen sollten die Benennungen und der Programmcode nach Möglichkeit selbsterklärend sein, sodass keine weitere Kommentierung notwendig ist. Sofern ein Kommentar erforderlich ist, weil dieser erheblich zum Verständnis beiträgt (z. B. Einheiten), beachten Sie die folgenden Punkte:

Kommentare sollen die Verwendung, den Nutzen und den Inhalt von Programmelementen und ihren Bestandteilen beschreiben und so Programme schnell und einfach verständlich gestalten. Kommentare sind zum einen bei Ansicht des Programmelements ersichtlich und werden zum anderen per Tooltip angezeigt.

Kommentarstelle

Wenn ein Kommentar vorgesehen ist, befindet sich dieser an den folgenden Stellen:

- Kommentare für Programmelemente: über der ersten Zeile der Elementdeklaration (z. B. oberhalb des Schlüsselworts FUNCTION_BLOCK)
- Kommentare für Variablen: in der gleichen Zeile im Anschluss an die Deklaration
- Deklarations-/Programmblock: in der Zeile über dem Block

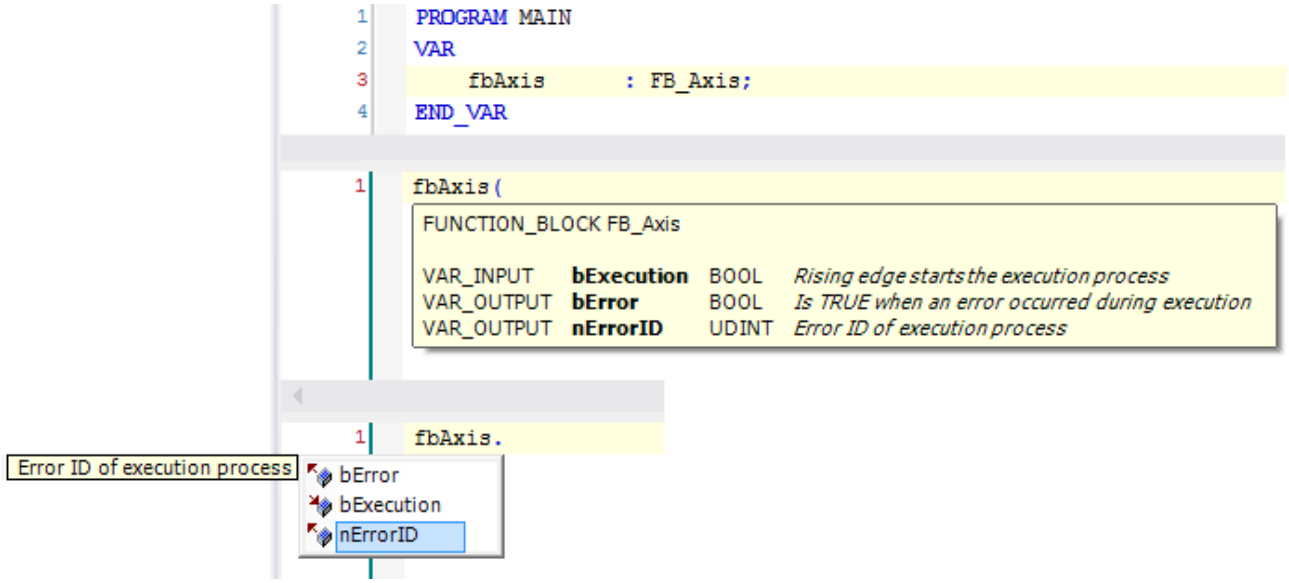
Kommentaroperator

- Für einzeilige Kommentare wird der Kommentaroperator '//' verwendet.
- Bei Kommentaren, die mehrere Zeilen betreffen, können der Kommentaroperator '//' oder der Operator '(* ... *)' zum Einsatz kommen.

Beispiel

```
// This function block represents an axis
FUNCTION_BLOCK FB_Axis
VAR_INPUT
     bExecution    : BOOL;     // Rising edge starts the execution process
END_VAR
VAR_OUTPUT
```

```
    bError        : BOOL;     // Is TRUE when an error occurred during execution
    nErrorID      : UDINT;    // Error ID of execution process
END_VAR
```



18.1.4.1 Strukturierung von Programmelementen

Jede neue Anweisung/Deklaration sollte in einer neuen Zeile beginnen.

Im Folgenden wird die Strukturierung von Programmelementen (u.a. POUs, DUTs, GVLs) mithilfe von Textblöcken, Textabschnitten und Textregionen erläutert.

Textblöcke

- In Textblöcken werden thematisch zusammenhängende Deklarationen, Zuordnungen oder Aufrufe mit jeweils einem bzw. keinem Parameter gebündelt.
- Aufrufe mit mehr als einem Parameter, Anweisungen (wie z. B. IF oder CASE) und Schleifen werden jeweils in separaten Textblöcken dargestellt.
- Jeder Textblock beginnt und endet mit einer Leerzeile.
- Textblöcke können innerhalb einer Textregion mithilfe der Tabulatortaste um eine Ebene eingerückt werden, sofern Sie den Vorteil nutzen möchten, den Code unterhalb des Regionenkommentars einklappen zu können. Falls Sie die Einrückung verwenden, sollte dies innerhalb der POU durchgängig umgesetzt werden.

Textabschnitte

- In Textabschnitten werden thematisch zusammenhängende Textblöcke gebündelt.
- Jeder Textabschnitt beginnt und endet mit einer Leerzeile.
- Die erste Zeile eines Textabschnitts kommentiert die folgenden Textblöcke.
- Textabschnitte können innerhalb einer Textregion mithilfe der Tabulatortaste um eine Ebene eingerückt werden, sofern Sie den Vorteil nutzen möchten, den Code unterhalb des Regionenkommentars einklappen zu können. Falls Sie die Einrückung verwenden, sollte dies innerhalb der POU durchgängig umgesetzt werden.

Textregionen

- In Textregionen werden thematisch verwandte Textabschnitte gebündelt.
- Jede Textregion beginnt und endet mit einer Trennlinie innerhalb eines Kommentars.
- Die erste Zeile unterhalb der Trennlinie kommentiert die folgenden Textabschnitte und damit die Textregion. Dieser Regionenkommentar befindet sich auf einer Ebene mit den darüber liegenden Kommentarzeichen.

- Regionenkommentare werden nicht eingerückt. Falls die in der Region enthaltenen Textblöcke und -abschnitte eingerückt sind, können so bis auf den Regionenkommentar alle Zeilen der Textregion ein- bzw. ausgeklappt werden.

Beispiel

Es gibt 2 Textregionen („Drill settings“ und „Conveyor settings“) mit jeweils 2 Textabschnitten (jeweils: „Positions“ und „Velocities“). In der Textregion „Drill settings“ besteht der Textabschnitt „Velocities“ aus 2 Textblöcken (thematisch zusammenhängende Zuordnungen und eine IF-Anweisung).

Programmcode:

```
//=========================================================
// Drill settings

    // Positions
    fbDrill.nPositionLower  := 100;
    fbDrill.nPositionTop    := 500;

    // Velocities
    fbDrill.fVelocityRated  := 40.0;
    fbDrill.fVelocityMax    := 100.0;

    IF fbDrill.fVelocityAverage > fbDrill.fVelocityRated THEN
        bWarning := TRUE;
        sWarning := 'Drill velocity: Average exceeded rated';
    END_IF

//=========================================================
// Conveyor settings

    // Positions
    fbConveyor.nPositionFilling    := 50;
    fbConveyor.nPositionDrill      := 200;
    fbConveyor.nPositionDischarge  := 300;

    // Velocities
    fbConveyor.fVelocityRated      := 75.5;

//=========================================================
```

Ausgeklappte Textregionen:

```
1  //=========================================================
2  // Drill settings
3
4      // Positions
5      fbDrill.nPositionLower  := 100;
6      fbDrill.nPositionTop    := 500;
7
8      // Velocities
9      fbDrill.fVelocityRated  := 40.0;
10     fbDrill.fVelocityMax    := 100.0;
11
12     IF fbDrill.fVelocityAverage > fbDrill.fVelocityRated THEN
13         bWarning    := TRUE;
14         sWarning    := 'Drill velocity: Average exceeded rated';
15     END_IF
16
17 //=========================================================
18 // Conveyor settings
19
20     // Positions
21     fbConveyor.nPositionFilling     := 50;
22     fbConveyor.nPositionDrill       := 200;
23     fbConveyor.nPositionDischarge   := 300;
24
25     // Velocities
26     fbConveyor.fVelocityRated       := 75.5;
27
28 //=========================================================
```

Eingeklappte Textregionen:

```
1  //=========================================================
2  // Drill settings
3  [13 lines]
17 //=========================================================
18 // Conveyor settings
19 [8 lines]
28 //=========================================================
```

Siehe auch:

- Alternative Möglichkeit zur Realisierung einer Einklappbarkeit: Region-Pragma [▶ 881]

18.1.4.2 Strukturierung von Textblöcken

Textblöcke innerhalb von DUTs, GVLs und POUs bestehen aus Deklarationen, Ausdrücken, Zuordnungen, Aufrufen, Anweisungen oder Schleifen.

Deklarationen

- Deklarationen bestehen aus 3 bis 5 Spalten:
  - Variablenname
  - [Lokierung (z. B. AT %Q*)]
  - Datentyp
  - [Initialisierung]

    - [Kommentar]
- Ob die Spalten für die Lokierung, die Initialisierung und den Kommentar existieren, hängt vom Programmierfall ab.
- Jede Deklarationsspalte innerhalb eines Programmelements beginnt auf gleicher Zeilenhöhe (Beispiel: alle Datentypen beginnen auf gleicher Zeilenhöhe).
- Diese Variablendeklarationen werden, soweit vorhanden, in einheitlicher Reihenfolge angelegt:

```
VAR_INPUT
END_VAR
VAR_IN_OUT CONSTANT
END_VAR
VAR_IN_OUT
END_VAR
VAR_OUTPUT
END_VAR

VAR
END_VAR
VAR PERSISTENT
END_VAR
VAR CONSTANT
END_VAR
```

- Jede deklarierte Variable erhält einen möglichst selbsterklärenden Bezeichner. Falls weitere Informationen notwendig sind, um den Zweck bzw. die Funktionalität der Variablen zu verdeutlichen, wird diese/r mittels eines Kommentars im Anschluss an die Deklaration beschrieben (in der gleichen Zeile).
- Thematisch zusammenhängende Deklarationen werden
    - in einem Deklarationsblock gebündelt,
    - mithilfe eines Kommentars beschrieben
    - und durch eine Leerzeile von anderen Deklarationsblöcken getrennt.

Beispiel:

```
VAR
    // Movement control
    bExecuteMovements     AT%I*   : BOOL;             (* Controls the movement
                                                         execution of car and bike *)
    bMovementsDone        AT%Q*   : BOOL;             (* Indicates if the car and
                                                         bike finished their movement *)

    // General distance variables
    nMovementTime                 : INT;              // Elapsed movement time in seconds
    nDistanceTotal                : INT;              // Total distance being covered by car and bike
    nStartPosition                : INT       := 100; // General start position where car and bike be
gin to move

    // Car
    fbCar                         : FB_Car;           // Car-FB whose movement is controlled
    nDistanceCar                  : INT;              // Distance being covered by car

    // Bike
    fbBike                        : FB_Bike;          // Bike-FB whose movement is controlled
    nDistanceBike                 : INT;              // Distance being covered by bike
END_VAR
```

Ausdrücke

- Bei Ausdrücken wird jeder Operator durch Leerzeichen separiert.
- Je nach Anzahl und Komplexität der Teilausdrücke können Klammern hinzugezogen werden, um die Abarbeitungsreihenfolge visuell zu verdeutlichen und die Lesbarkeit zu erhöhen.

Beispiele:

```
nDistanceCar   := fbCar.nStartPosition + (fbCar.nVelocity * nMovementTime);
nDistanceBike  := fbBike.nStartPosition + (fbBike.nVelocity * nMovementTime);
nDistanceTotal := nDistanceCar + nDistanceBike;
```

Zuordnungen

- Zuweisungsoperatoren eines Zuordnungsblocks beginnen auf gleicher Zeilenhöhe.

- Gleiche Zeilenhöhen werden durch die Tabulatortaste erreicht.
- Hinter einem Zuweisungsoperator befindet sich ein Leerzeichen.
- Sehr lange Zuordnungen, die z. B. über die Bildschirmbreite hinausgehen, werden durch einen Zeilenumbruch aufgeteilt. Der Textteil, der sich dadurch in einer neuen Zeile befindet, wird durch die Tabulatortaste auf eine Höhe nach dem Zuweisungsoperator eingerückt, z. B.:

```
bExpressionResult := bCondition1 AND (bCondition2 OR bCondition3)
                     AND bCondition4 AND (bCondition5 OR bCondition6);
```

- Thematisch zusammenhängende Zuordnungen werden
  - in einem Zuordnungsblock gebündelt
  - und durch eine Leerzeile von anderen Blöcken getrennt.

Beispiel:

```
//=========================================================
// Submodule enable

    // Sensors
    fbSensorEStop.bEnable    := bGeneralEnable AND bSensorEnable;
    fbSensorStartPos.bEnable := bGeneralEnable AND bSensorEnable;

    // Cylinder
    fbCylinderSystem1Main.bEnable := bGeneralEnable AND bCylinderEnable;
    fbCylinderSystem1Sub.bEnable  := bGeneralEnable AND bCylinderEnable;
    fbCylinderSystem2Main.bEnable := bGeneralEnable AND bCylinderEnable;
    fbCylinderSystem2Sub.bEnable  := bGeneralEnable AND bCylinderEnable;

    // Axes
    fbAxisSubfoil.bEnable := bGeneralEnable AND bAxisEnable AND NOT bStop;
    fbAxisMain.bEnable    := bGeneralEnable AND bAxisEnable AND NOT bStop;

//=========================================================
```

Aufruf von Methoden/Funktionen/Funktionsblöcken

- Aufrufe von Methoden/Funktionen/Funktionsblöcken mit bis zu drei Parametern können einzeilig geschrieben werden.
- Mehrere einzeilige Aufrufe können in einem Textblock gebündelt werden.
- Besitzen die Programmelemente mehr als drei Parameter, stellen sie jeweils separate Textblöcke dar und werden durch eine Leerzeile von anderen Blöcken getrennt. Dabei befindet sich der erste Parameter in der Zeile des Aufrufs oder in der nachfolgenden Zeile. Die weiteren Parameter beginnen jeweils in einer neuen Zeile.
- Die Parameteranfänge werden mithilfe der Tabulatortaste auf eine Ebene eingerückt.
- Auch die Zuweisungsoperatoren beginnen auf einer Zeilenhöhe.
- Hinter einem Zuweisungsoperator befindet sich ein Leerzeichen.
- Werden einem Funktionsblock an der gleichen Programmstelle, an der die Instanz aufgerufen wird, Parameter übergeben oder werden die Parameter des Funktionsblocks gelesen, geschieht dies beim Aufruf des Funktionsblock und nicht unmittelbar vor bzw. nach dem Instanzaufruf.
- Um die Lesbarkeit des Programmcodes zu erhöhen, wird ein qualifizierter Aufruf von Methoden/Funktionen/Funktionsblöcken empfohlen. Dabei wird die Zuweisung ausformuliert, indem der Parametername beim Aufruf explizit genannt wird.

Negatives Beispiel:

```
fbTimerStart.IN := TRUE;
fbTimerStart.PT := T#10S;
fbTimerStart();
bStartExecution := fbTimerStart.Q;
```

Positives Beispiel:

```
fbTimerStart( IN := TRUE,
              PT := T#10S,
              Q  => bStartExecution);
```

Ausführliches Beispiel:

```
//=======================================================
// Stop

    // Cylinder
    fbCylinderPos1.Stop(bExecute := TRUE);
    fbCylinderPos2.Stop(bExecute := TRUE);
    fbCylinderPos3.Stop(bExecute := TRUE);

    // Axes
    fbAxis1.Stop(bExecute := TRUE, bDone => bAxis1Stopped);
    fbAxis2.Stop(bExecute := TRUE, bDone => bAxis2Stopped);

//=======================================================
```

Anweisungen

- RETURN, CONTINUE, EXIT, JMP, IF, CASE
- JMP-Anweisung: Sprunganweisungen können und sollten vermieden werden, da sie die Lesbarkeit des Codes vermindern.
- IF-Anweisung:
  - Eine IF-Anweisung stellt einen separaten Textblock dar und wird durch eine Leerzeile von anderen Blöcken getrennt.
  - Verschachtelte IF-Anweisungen sollten zum besseren Verständnis und zur Erhöhung der Softwarequalität vermieden werden. Stattdessen sollte immer ein definierter Zustand gelten, der mithilfe einer CASE-Anweisung implementiert wird.
  - Alle IF-ELSIF-Anweisungen sollten einen ELSE-Zweig beinhalten.
    Sehen Sie dazu auch den Themenpunkt IF-ELSIF-Anweisungen mit ELSE-Zweig [▶ 1144] im Abschnitt Programmierung [▶ 1139].
  - Die Anweisungen werden um eine Ebene eingerückt, sodass sich nur die Schlüsselwörter IF/ELSIF/ELSE/END_IF auf einer Ebene befinden. Das Schlüsselwort THEN erhält keine eigene Zeile.
  - Sehr lange Bedingungen, die über die Bildschirmbreite hinausgehen, werden durch einen Zeilenumbruch aufgeteilt. Der Textteil, der sich dadurch in einer neuen Zeile befindet, wird durch die Tabulatortaste auf eine passende Höhe der oberen Zeile eingerückt: auf die Höhe des IF-Schlüsselwortes oder bei verschachtelten Bedingungen auf die Höhe der dazugehörigen/gleichwertigen Teilbedingung, z. B.:

Beispiel:

```
IF a > 10 THEN
   …
ELSIF a < 10 THEN
   …
ELSE
   …
END_IF

IF bCondition1 AND bCondition2 AND (bCondition3 OR bCondition4)
   AND bCondition5 AND bCondition6
   AND (bCondition7 OR bCondition8 OR bCondition9 OR bCondition10) THEN
   …
END_IF
```

- CASE-Anweisung:
  - Eine CASE-Anweisung stellt einen separaten Textblock dar und wird durch eine Leerzeile von anderen Blöcken getrennt.
  - Die Enumerationswerte werden jeweils durch eine Leerzeile voneinander getrennt und im Verhältnis zur CASE-Anweisung um eine Ebene eingerückt, sodass sich nur die Schlüsselwörter CASE/ELSE/END_CASE auf einer Ebene befinden.
  - Zusätzlich kann über jedem Werteblock ein Kommentar hilfreich sein.
  - Die Anweisungen zu einem Enumerationswert beginnen 1-2 Zeilen unterhalb des Wertes (nicht direkt neben dem Wert) und werden im Verhältnis zu diesem Wert um eine Ebene eingerückt.

Beispiel:

```
//=========================================================
// Cylinder sorting process

CASE eSortingState OF
    E_SortingState.DetectionOfBox:
        fbCylinder.MoveToBase();
        sVisuMessage := 'System in detection mode.';

        IF fbSensorDelay.bOut THEN
            eSortingState := eSorting_MoveCylToWorkPos;
        END_IF

    E_SortingState.MoveCylToWorkPos:
        fbCylinder.MoveToWork();

        IF fbCylinder.bAtWorkPos THEN
            eSortingState := eSorting_MoveCylToBasePos;
            sVisuMessage  := '';
        ELSE
            sVisuMessage  := 'Waiting for cylinder in work pos.';
        END_IF

    E_SortingState.MoveCylToBasePos:
        fbCylinder.MoveToBase();

        IF fbCylinder.bAtBasePos THEN
            eSortingState := eSorting_DetectionOfBox;
            sVisuMessage  := '';
        ELSE
            sVisuMessage  := 'Waiting for cylinder in base pos.';
        END_IF

ELSE
    sVisuMessage := 'System in non-existent state. Please check application.';
END_CASE

//=========================================================
```

Schleifen (FOR, WHILE, REPEAT)

- Die Schleifen-Parametrierungen befinden sich jeweils in einer Zeile (FOR ... DO, WHILE ... DO, REPEAT).
- Die Anweisungen werden um eine Ebene eingerückt, sodass sich nur die Schlüsselwörter FOR/ END_FOR bzw. WHILE/END_WHILE bzw. REPEAT/UNTIL/END_REPEAT auf einer Ebene befinden.

Beispiel:

```
//=========================================================
// Initialize storage areas with FOR loop

FOR nAreaCounter := cStorageStart TO cStorageEnd BY 1 DO
    aStorageAreas[nAreaCounter] := nAreaCounter;
END_FOR

//=========================================================
// Initialize delivery areas with WHILE loop

nAreaCounter := cDeliveryStart;

WHILE nAreaCounter <= cDeliveryEnd DO
    aDeliveryAreas[nAreaCounter] := nAreaCounter;
    nAreaCounter                 := nAreaCounter + 1;
END_WHILE

//=========================================================
```

18.2 Namenskonventionen

In diesem Abschnitt der SPS-Programmierkonventionen werden die folgenden Themenpunkte behandelt.

Allgemeines

1. Allgemeine Namensvorgaben [▶ 1130] [+]
2. Übliche Abkürzungen [▶ 1130] [+]

Bezeichner

1. Bezeichner für POUs und DUTs [▶ 1133] [+]
2. Bezeichner für Variablen und Instanzen [▶ 1134] [+]

Globale Variablenlisten und Parameterlisten

1. Globale Variablenlisten [▶ 1138] [+]
2. Parameterlisten [▶ 1138] [+]
3. Attribut 'qualified_only' bei GVL verwenden [▶ 1138] [+]

Beispiele

Siehe: Beispiele [▶ 1138]

18.2.1 Allgemeines

Themenpunkte:

1. Allgemeine Namensvorgaben [▶ 1130] [+]
2. Übliche Abkürzungen [▶ 1130] [+]

Allgemeine Namensvorgaben

- Namen sind aussagekräftig, um den Zweck des Objekts leicht verstehen zu können.
- Vermeiden Sie mehrfaches Nutzen gleicher Namen. Sehen Sie dazu auch den Themenpunkt Kein mehrfaches Nutzen gleicher Namen [▶ 1148] im Abschnitt Programmierung [▶ 1139].
- Besteht ein Bezeichner aus mehreren Wörtern, wird der erste Buchstabe eines jeden Wortes großgeschrieben (CamelCase). In der Regel werden keine Trennzeichen, wie z. B. '_', zwischen den Wörtern verwendet. Sollte in Ausnahmefällen eine gute Lesbarkeit mittels CamelCase nicht erreicht werden können, wird die Verwendung des Unterstrich-Zeichens als Trennzeichen empfohlen. Weitere Ausnahmen für die Verwendung des Unterstrich-Zeichens bei Objektnamen finden Sie unter dem Themenpunkt Gültige Zeichen beachten [▶ 1120].
- Bezeichner haben ein konsistentes Präfix, sodass der Objekttyp leicht zu erkennen ist. Wird für einen Bezeichner ein Präfix vorgeschrieben, beachten Sie die entsprechende Groß- bzw. Kleinschreibung. Sehen Sie dazu auch: Bezeichner [▶ 1133]
- Bezeichner beginnen immer mit einem Buchstaben.
- Abkürzungen werden ebenfalls in CamelCase geschrieben. (Ausnahmen sind eigenständige Begriffe wie ID, CRLF, PC, PLC, …. Diese können in UpperCase geschrieben werden.)

Negative Beispiele:

```
tADSTimeout    : TIME;
eCMDType       : E_CommandType;
```

Positive Beispiele:

```
nAddr          : UDINT;
nMsgCtrlMask   : DWORD;
cMaxCharacters : UDINT;
tAdsTimeout    : TIME;
eCmdType       : E_CommandType;
stRemotePCInfo : ST_RemotePCInfo;
```

Übliche Abkürzungen

Folgende übliche Abkürzungen können verwendet werden. Es darf für den entsprechenden Begriff keine andere Abkürzung gewählt werden.

Ebenso finden sich in der Liste Begriffe, welche nicht abgekürzt werden sollen. Verzichten Sie allgemein auf Abkürzungen, welche den Begriff um weniger als 3 Buchstaben kürzen würden, weil dies die Lesbarkeit nicht verbessert. (Ausnahmen sind die Abkürzungen Cnt und Idx.)

Abkürzung	Begriff
Abs	Absolute
Ack	Acknowledge
Act	Actual / Active (Current)
Addr	Address
/	Alarm
Auto	Automatic
Avg	Average
Bwd	Backward
/	Buffer
Calc	Calculation / Calculate
Char	Character
Cmd	Command
Com	Communication
Config	Configuration
Cnt	Count (Number of)
Dst	Destination
Diag	Diagnostic(s)
Diff	Difference
Dim	Dimension
/	Error
/	Execute
Fwd	Forward
Hdl	Handle
ID	Identifier
Idx	Index
In	Input
Info	Information
Init	Initialization / Initialize
Itf	Interface
Len	Length
Lib	Library
Max	Maximum
Mem	Memory
Min	Minimum
Msg	Message
Obj	Object
Op	Operation
Out	Output
Ptr	Pointer
Pos	Position
Prev	Previous
Rcv	Receive / Received
Ref	Reference
Src	Source
Srv	Server
Sync	Synchonize / Synchronization
Temp	Temperature
Tmp	Temporary

Abkürzung	Begriff
/	Timeout
Velo	Velocity
Visu	Visualization
/	Warning

18.2.2 Bezeichner

Themenpunkte:

1. Bezeichner für POUs und DUTs [▶ 1133] [+]
2. Bezeichner für Variablen und Instanzen [▶ 1134] [+]

Bezeichner für POUs und DUTs

Siehe: Bezeichner für POUs und DUTs [▶ 1133]

Bezeichner für Variablen und Instanzen

Siehe: Bezeichner für Variablen und Instanzen [▶ 1134]

18.2.2.1 Bezeichner für POUs und DUTs

Bei der Bezeichnung von POUs und benutzerdefinierte Datentypen (DUTs) sollten Sie die folgenden Punkte beachten.

- Präfixe von Objekten werden grundsätzlich großgeschrieben.
- Trennzeichen zwischen Präfix und eigentlichem Objektnamen ist das Unterstrich-Zeichen '_'.
- Der eigentliche Objektname beginnt stets mit einem Großbuchstaben.
- Wenn Schnittstellen von TcCOM-Objekten direkt eingebettet sind, dann haben sie lediglich das Präfix 'I', z. B. ITcUnknown.

Static Analysis:

Beachten Sie auch die Möglichkeit zur Überprüfung der Programmierkonventionen mit Hilfe von TE1200 PLC Static Analysis [▶ 1117].

Präfixe:

Objekt	Präfix	Beschreibung	Beispiel	Static Analysis Namenskonventions-ID
FUNCTION_BLOCK	FB_	Funktionsbaustein	FB_WritePersistentData	NC0103
ACTION		Aktion (von einem Funktionsbaustein oder einem Programm)	MoveAbsolute	NC0106
METHOD		Methode (von einem Funktionsbaustein oder einer Schnittstelle)	Reset	NC0105
PROPERTY	entsprechend dem Rückgabetyp (siehe Bezeichner für Variablen und Instanzen [▶ 1134])	Eigenschaft (von einem Funktionsbaustein, einem Programm oder einer Schnittstelle)	nErrorID fMotorTempC	NC0107 Siehe auch: Platzhalter {datatype}
PROGRAM		Programm	ModuleControl	NC0102
FUNCTION	F_	Funktion	F_MeterToInch	NC0104
STRUCT	ST_	Struktur	ST_BufferEntry	NC0151
ENUM	E_	Aufzählungstyp	E_MachineState E_Quality	NC0152
TYPE	T_	Aliastyp	T_Nibble	NC0154
UNION	U_	Union	U_Control	NC0153
INTERFACE	I_	Schnittstelle	I_Cylinder	NC0108
GVL	GVL als Name oder GVL_ als Präfix	Globale Variablenliste	GVL GVL_Axis GVL_Subsystem	
	GCL	Globale Konstantenliste	GCL	
Param	Param als Name oder Param_ als Präfix	Globale Parameterliste	Param Param_Subsystem	

Falls ein Property <name> durch eine Variable des Funktionsbausteins direkt repräsentiert wird, wird diese Variable als _<name> bezeichnet.

Enumeration:
Bei der Definition einer Enumeration verwenden Sie das Attribut `{attribute 'qualified_only'}`, womit die Verwendung der Enumeration vereinfacht und zugleich eine Abkürzung des Aufzählungstyps unnötig wird. Sehen Sie dazu auch die Themenpunkte Attribute 'qualified_only' und 'strict' bei Enumeration verwenden [▶ 1152] im Abschnitt Programmierung [▶ 1139].

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalStates :
(
     Red    := 0,
     Yellow,
     Green
);
END_TYPE
```

18.2.2.2 Bezeichner für Variablen und Instanzen

Bei der Bezeichnung von Variablen und Instanzen beachten Sie die folgenden Punkte:

- Präfixe von Variablen- und Instanznamen werden kleingeschrieben.

- Das erste Zeichen nach dem Präfix wird großgeschrieben (Option **Erstes Zeichen nach Präfix soll ein Großbuchstabe sein** der Namenskonventionen vom Static Analysis aktivieren, siehe folgenden Hinweis zum Static Analysis).

Static Analysis:

Beachten Sie auch die Möglichkeit zur Überprüfung der Programmierkonventionen mit Hilfe von TE1200 PLC Static Analysis [▶ 1117].

Präfixe:

1) Variablen elementarer und dynamischer Datentypen:

Typ	Präfix	Beschreibung	Deklarations-beispiel	Aufrufbeispiel	Static Analysis Namenskonventions-ID

SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, BYTE, WORD, DWORD, LWORD, XINT, UXINT, XWORD	n	Integer, Bitbasierte Zahl	nErrorID : UDINT;
			nSize : UINT;
			nMask : WORD;
			nMessages : UINT;
				nErrorID := 16#745;
				nSize := SIZEOF(ST_BufferEntry);
				nMask := nMask & 16#0FF0;
				nMessages := 3;
					NC0005-NC0016, NC0035, NC0037, NC0038, NC0160

BOOL, BIT	b	boolean (bit)	bConfigured bReset		NC0003, NC0004
REAL, LREAL	f	float	fPosition		NC0017, NC0018
STRING	s	string	sNetID		NC0019
WSTRING	ws	wide string (unicode)	wsRouteName		NC0020
TIME, LTIME	t	time	tDelay		NC0021, NC0022
TIME_OF_DAY	td	time of day	tdClockTime		NC0025
DATE	d	date	dProductionStart		NC0023
DATE_AND_TIME	dt	date and time	dtProductionStart		NC0024
ARRAY[...] OF ...	a	array	aMessages		NC0030
POINTER TO ...	p	pointer	pData pNextProduct	pData := ADR(aBuffer);	NC0026
POINTER TO POINTER TO ...	pp	Verschachtelter Zeiger	ppData		
POINTER TO INTERFACE	pip	Zeiger auf eine Schnittstelle	pipCylinder		
REFERENCE TO ... 1) als Methodeneingang 2) in sonstigen Fällen	1) entsprechend dem Basisdatentyp der Referenz 2) ref	Referenz	1) nData (bei REFERENCE TO INT) 2) refSize	1) SampleMethod(nData := <…>) 2) refSize REF= nSize;	NC0027

- **Zeiger / Pointer:**
  - Typisierte Pointer aus Sicherheitsgründen untypisierten Pointern vorziehen.
  - Pointer als POINTER TO <Datentyp> deklarieren, wobei der <Datentyp> dem Datentyp der Variablen entspricht, auf den der Pointer verweisen soll.

- Ein Pointer, der auf ein Array zeigen soll, als POINTER TO ARRAY[...] OF <Datentyp> deklarieren, wobei der <Datentyp> dem Datentyp des Arrays entspricht, auf das verwiesen werden soll.
- Ist der Datentyp der Variablen, auf die ein Pointer zeigen soll, unbekannt und damit untypisierbar, sollte er als POINTER TO BYTE (oder PVOID) deklariert werden. Eine Verwendung des Datentyps DWORD als Zeiger ist aus Gründen der 64bit-Kompatibilität nicht zulässig.
- Die Dereferenzierung eines Pointers erfolgt über den Inhaltsoperator '^'.
- Achten Sie besonders auf die Gültigkeit der Adresse, auf die ein Pointer zeigt.

Beispiel 1:

```
nSample       : INT;
pSampleToInt  : POINTER TO INT;
```

```
pSampleToInt  := ADR(nSample)
pSampleToInt^ := 123;              // Result: nSample = 123
```

Beispiel 2:

```
aSample                   : ARRAY[1..3] OF LREAL;
pSampleToArrayOfLreal     : POINTER TO ARRAY[1..3] OF LREAL;
```

```
pSampleToArrayOfLreal     := ADR(aSample);
pSampleToArrayOfLreal^[2] := 4.56; // Result: aSample[2] = 4.56
```

- **Pointerarithmetiken / Pointervergleiche:**
  - Wenn möglich, verzichten Sie aus Sicherheitsgründen auf Pointerarithmetiken und die Anwendung von Vergleichsoperatoren auf Pointer (z. B. Inkrementierungen wie [pSampleToLreal := pSampleToLreal + SIZEOF(LREAL)] oder Pointervergleiche mittels >, >=, <, <=).
  - Falls Sie auf diese Operationen nicht verzichten können, verwenden Sie sie höchstens innerhalb eines Arrays.
  - Achten Sie dabei ist besonders darauf, dass die Pointer auf Elemente des gleichen Arrays zeigen und dass der Adressbereich des Arrays nicht verlassen wird.

2) Instanzen von Objekten und benutzerdefinierter Datentypen:

Instanzen von Funktionsbausteinen, die in der IEC61131-Norm definiert und von grundlegender Bedeutung sind (R_TRIG, TON...), besitzen kein anderes Präfix. Beispiel für eine korrekte Benennung:

```
fbAdsTimer : TON;
```

Typ	Präfix	Beschreibung	Deklarationsbeispiel	Aufrufbeispiel	Static Analysis Namenskonventions-ID
FUNCTION_BLOCK	fb	Instanz eines Funktionsbausteins	fbWritePersData : FB_WritePersistentData;	fbWritePersData();	NC0031
STRUCT	st	Instanz einer Struktur	stBufferEntry : ST_BufferEntry;	stBufferEntry.nCounter := 5;	NC0032
ENUM	e	Instanz einer Aufzählung	eMachineState : E_MachineState; eQuality : E_Quality;	eMachineState := E_MachineState.Stop; eQuality := E_Quality.Good;	NC0029
TYPE (Alias)	entsprechend dem internen Datentyp	Instanz eines Aliastyps	nNibble : T_Nibble; sNetId : T_AmsNetId;	nNibble := 16#1; sNetId := '1.2.3.4.5.6';	NC0033 Siehe auch: Platzhalter {datatype}
INTERFACE	ip	Instanz einer Schnittstelle	ipCylinder : I_Cylinder;	ipCylinder := fbCylinderBase;	NC0036
UNION	u	Instanz einer Union	uCtrl : U_Control;	uCtrl.aRegister[1] := 16#02;	NC0034

3) Sonstiges:

Typ	Präfix	Beschreibung	Deklarationsbeispiel	Static Analysis Hinweis
Verschachtelte Typen	Lediglich entsprechend dem äußersten Typ Ausnahme 1: 'pp' für POINTER TO POINTER TO Ausnahme 2: 'pip' für POINTER TO INTERFACE		pTelegramData : POINTER TO ARRAY[0..7] OF BYTE;	Option **Rekursive Präfixe für kombinierbare Datentypen** deaktivieren Siehe auch: Namenskonventionen (2)

Lokale und globale Konstanten	c		VAR CONSTANT
			cSyncID : UINT := 16#80;
			END_VAR
			VAR_GLOBAL CONSTANT
			cLowerThreshold : UDINT := 9;
			cOptionMove : DWORD := 16#04;
			END_VAR
				NC0062, NC0070

Globale Variablen		Für globale Variablen gelten die gleichen Namensregeln. (Hängen Sie vor den Variablennamen kein zusätzliches zweites Präfix.)	VAR_GLOBAL nProducedUnits : UINT; END_VAR	Option **Namensraumpräfix mit Datentyppräfix kombinieren** deaktivieren Siehe auch: Namenskonventionen (2)
HRESULT	hr		hrErrorCode : HRESULT;	NC0160
PVOID	p	pointer	pData : PVOID;	NC0160
GUID		Für eine GUID wird kein Präfix vergeben.		
AT%I*	optional vor dem Datentyppräfix: I	allokierter Eingang	fActPos AT%I*: LREAL; oder: IfActPos AT%I*: LREAL;	
AT%Q*	optional vor dem Datentyppräfix: O	allokierter Ausgang	fSetPos AT%Q*: LREAL; oder: OfSetPos AT%Q*: LREAL;	

Allokierte Ein- bzw. Ausgänge können optional mit einem 'I' bzw. 'O' als zusätzliches Präfix vor dem Datentyppräfix versehen werden. Für Instanzen stellt es das einzige Präfix dar, das großgeschrieben wird. Halten Sie sich konsequent an die Entscheidung für oder gegen die Nutzung dieses optionalen Präfixes, sodass die Bezeichnung von allokierten Variablen projektweit einheitlich ist.

18.2.3 Globale Variablenlisten und Parameterlisten

Themenpunkte:

1. Globale Variablenlisten [▶ 1138] [+]

2. Parameterlisten [▶ 1138] [+]
3. Attribut 'qualified_only' bei GVL verwenden [▶ 1138] [+]

Globale Variablenlisten

Eine Liste mit globalen Variablen oder Konstanten erhält den Namen „GVL“ oder das Präfix „GVL_“, woraufhin eine Beschreibung der GVL im Namen folgt.

Beispiele:

- GVL
- GVL_Axis
- GVL_Subsystems

Parameterlisten

Eine Liste mit globalen Parametern erhält den Namen „Param“ oder das Präfix „Param_“, woraufhin eine Beschreibung der Parameterliste im Namen folgt.

Beispiele:

- Param
- Param_Subsystems

Attribut 'qualified_only' bei GVL verwenden

Bei der Definition einer globalen Variablenliste [▶ 75] oder einer Parameterliste [▶ 75] das Attribut {attribute 'qualified_only'} [▶ 859] verwenden, womit bei Verwendung der Variablen die Nutzung des Namensraums der GVL erzwungen wird. Durch die Nutzung des Namensraums (Bsp.: `GVL_Ctrl.bPaintingActive`) wird der globale Scope der Variablen deutlich.

Positives Beispiel:

Globale Variablenliste "GVL_Ctrl":

```
{attribute 'qualified_only'}
VAR_GLOBAL
    bPaintingActive  : BOOL;
END_VAR
```

Sehen Sie dazu auch den Themenpunkt Attribut 'qualified_only' bei GVL verwenden [▶ 1138] im Abschnitt Programmierung [▶ 1139].

18.2.4 Beispiele

```
VAR_GLOBAL
    nSocketPort    : INT;
    sNetID         : T_AmsNetId;
END_VAR
```

```
VAR_GLOBAL CONSTANT
    cSocketAddr    : INT          := 1;
    cLocalNetID    : T_AmsNetId   := '';
END_VAR
```

```
FUNCTION F_CompareVelocity : UINT
```

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_StandardColor :
(
    Blue   := 0,
    Green  := 1
);
END_TYPE
```

```
FUNCTION_BLOCK FB_WirelessCom
VAR CONSTANT
    cFrequency     : REAL := 868.0;
END_VAR
VAR
```

```
    eColor          : E_StandardColor;
    aColors         : ARRAY[1..cMaxColors] OF E_StandardColor;
END_VAR
-------------------------------------------------------------------
IF (sNetId = cLocalNetID) THEN
    eColor     := E_StandardColor.Green;
    aColors[1] := eColor;
END_IF
```

```
FUNCTION_BLOCK FB_ModuleControl
VAR_IN_OUT
    aTerminalData  : ARRAY[1..cOversamples] OF INT;
END_VAR
VAR
    fbCalc         : FB_Calculate;
END_VAR
-------------------------------------------------------------------
fbCalc.Reset();
fbCalc( pDataIn     := ADR(aTerminalData),
        nDataInSize := SIZEOF(aTerminalData) );
```

18.3 Programmierung

In diesem Abschnitt der SPS-Programmierkonventionen werden die folgenden Themenpunkte behandelt.

Beachten Sie auch die Möglichkeit zur Überprüfung der Programmierkonventionen mit Hilfe von TE1200 PLC Static Analysis [▶ 1117].

Allgemeines

Schleifen und Bedingungen

1. CASE-Anweisung nur mit Enumerationsinstanz [▶ 1142] [++]
2. Grenzen einer Schleife als Konstante deklarieren [▶ 1142] [+]
3. In CASE-Anweisung alle Enumerationswerte der Enumerationsvariablen behandeln [▶ 1143] [+]
4. IF-ELSIF-Anweisungen mit ELSE-Zweig [▶ 1144] [+]
5. Reihenfolge der IF Bedingungen [▶ 1145] [+]

Fehlercodes

1. Datentyp für Fehlercodes [▶ 1145] [++]
2. Fehlerinformation an Funktionen und Methoden [▶ 1146] [+]
3. Fehlerinformation an Funktionsbausteinen [▶ 1146] [+]

Lesbarkeit, Wartbarkeit

1. Keine unbenutzten Deklarationen/Objekte oder unnützen Code [▶ 1147] [++]
2. Keine „magischen Werte/magic numbers“ [▶ 1148] [+]
3. Kein mehrfaches Nutzen gleicher Namen [▶ 1148] [+]
4. Gleiche Notation in Deklaration und Implementierung [▶ 1148] [+]
5. Leere Statements (;) vermeiden [▶ 1149] [+]
6. Jede Zuweisung in separater Codezeile [▶ 1149] [+]

Bibliotheken

Entwicklung von Bibliotheken

1. Bibliotheksnamen [▶ 1150] [++]
2. Versionierung [▶ 1150] [++]
3. Kapselung interner Typdefinitionen [▶ 1151] [+]
4. Bezeichner in Bibliotheken [▶ 1151] [+]

Verwendung von Bibliotheken

1. Bibliotheksverwendung [▶ 1151] [++]
2. Versionsüberprüfung [▶ 1151] [++]
3. Fixieren von Bibliotheksversionen [▶ 1151] [+]

DUTs

Implementierung von DUTs

1. Attribute 'qualified_only' und 'strict' bei Enumeration verwenden [▶ 1152] [++]
2. Bei Enumeration keinen, nur den ersten oder alle Werte initialisieren [▶ 1152] [+]

Verwendung von DUTs

1. Enumerationsvariablen nur zugehörige Enumeratoren zuweisen [▶ 1153] [++]

POUs

Implementierung von Funktionen, Methoden, Aktionen

1. Kein „Call by value“ von großen Parametern in Funktionen/Methoden [▶ 1155] [++]
2. Keine großen Variablen in Funktionen/Methoden deklarieren [▶ 1155] [++]
3. Keine Aktionen verwenden [▶ 1156] [+]
4. Alle Parameter einer Funktion/Methode intern verwenden [▶ 1156] [+]
5. Rückgabewert einer Funktion/Methode nur an einer Stelle zuweisen [▶ 1156] [+]
6. Zugriff auf Methoden so weit wie möglich einschränken [▶ 1158] [+]
7. Gruppierung von Parametern als Struktur [▶ 1158] [+]

Verwendung von Funktionen, Methoden

1. Zurückgelieferte Fehlerinformationen einer POU auswerten [▶ 1158] [++]
2. Rückgabewert einer Funktion/Methode verwenden [▶ 1159] [+]
3. Funktionen/Methoden nicht in sich selbst aufrufen [▶ 1160] [+]

Implementierung von Funktionsbausteinen

1. Online-Change-Fähigkeit sicherstellen [▶ 1161] [++]
2. Einheitliche Schnittstelle bei einmaliger asynchroner Abarbeitung [▶ 1161] [+]
3. Einheitliche Schnittstelle bei kontinuierlicher asynchroner Abarbeitung [▶ 1161] [+]
4. Alle Parameter eines Funktionsbausteins intern verwenden [▶ 1162] [+]
5. Gruppierung von Parametern als Struktur [▶ 1162] [+]

Verwendung von Funktionsbausteinen

1. Funktionsbaustein-Instanzen nicht als VAR PERSISTENT deklarieren [▶ 1163] [++]
2. Zurückgelieferte Fehlerinformationen einer POU auswerten [▶ 1158] [++]
3. Auf lokale Variablen einer POU nicht von außen zugreifen [▶ 1163] [++]
4. Keine direkte Zuweisung von Objekten [▶ 1164] [++]
5. Keine temporären Funktionsbaustein-Instanzen [▶ 1164] [+]

Variablen

Allgemein

1. Fließkommazahlen nicht auf Gleichheit bzw. Ungleichheit testen [▶ 1165] [++]
2. Unerwünschte Ergebnisse mit Hilfe von explizitem Casten vermeiden [▶ 1166] [+]

Variablenkapselung

1. Nicht veränderte Variablen als VAR CONSTANT deklarieren [▶ 1167] [+]
2. Globalere Bezeichner nicht verschatten [▶ 1167] [+]
3. ADS Zugriff einschränken [▶ 1168] [+]

Arrays

1. Array-Grenzen über Konstanten definieren [▶ 1168] [++]
2. Array-Untergrenze von 1 [▶ 1169] [+]

Pointer, Referenzen, Interfaces

1. Temporäre Existenz von Pointern/Referenzen/Interfaces auf temporär existierende Objekte [▶ 1169] [++]
2. Pointer/Referenzen jeden Zyklus neu setzen [▶ 1170] [++]
3. Pointer/Referenzen/Interfaces vor jeder Verwendung prüfen [▶ 1170] [++]
4. Referenzen gegenüber Pointern vorziehen [▶ 1171] [++]

Allokierte Variablen

1. Keine direkte Adressierung verwenden [▶ 1172] [++]
2. Mehrfache Schreibzugriffe auf Ausgänge vermeiden [▶ 1172] [++]
3. Überlappende Speicherbereiche von Adressvariablen vermeiden [▶ 1173] [+]

Globale Variablen

1. Attribut 'qualified_only' bei GVL verwenden [▶ 1174] [+]
2. Globale Variablen mit Bedacht verwenden [▶ 1174] [+]

Strings

1. Bezeichner für Größen- und Längenangaben [▶ 1175] [++]
2. Empfohlene Default-Größe [▶ 1175] [+]
3. Übergabe von großen Strings [▶ 1176] [+]
4. Verarbeitung von großen Strings [▶ 1176] [+]

Laufzeitverhalten

Allgemein

1. Division durch Null abfangen. [▶ 1176] [++]

Dynamischer Speicher

1. Dynamische Speicherallokationen mit Bedacht durchführen [▶ 1177] [++]

Multiple Tasks

1. Konkurrierende Zugriffe auf Speicherbereiche vermeiden [▶ 1178] [++]
2. Konkurrierende Zugriffe auf Funktionsbausteine vermeiden [▶ 1179] [++]
3. Globale Variablen mit Bedacht verwenden [▶ 1174] [+]

18.3.1 Allgemeines

18.3.1.1 Schleifen und Bedingungen

Themenpunkte:

1. CASE-Anweisung nur mit Enumerationsinstanz [▶ 1142] [++]
2. Grenzen einer Schleife als Konstante deklarieren [▶ 1142] [+]
3. In CASE-Anweisung alle Enumerationswerte der Enumerationsvariablen behandeln [▶ 1143] [+]
4. IF-ELSIF-Anweisungen mit ELSE-Zweig [▶ 1144] [+]
5. Reihenfolge der IF Bedingungen [▶ 1145] [+]

CASE-Anweisung nur mit Enumerationsinstanz

In einer CASE-Anweisung sollten Sie statt „magischer“ Zahlen eine Enumerationsinstanz samt den dazugehörigen Enumeratoren zur Definition der Zustandsmaschine verwenden. Dadurch sind die Werte der Zustandsmaschine selbstsprechend und es werden keine „Magic Numbers“ verwendet.

Beachten Sie auch die folgenden Themenpunkte der Programmierkonventionen:

- Attribute 'qualified_only' und 'strict' bei Enumeration verwenden [▶ 1152]
- Keine „magischen Werte/magic numbers“ [▶ 1148]

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    nState  : INT; // Used to operate the sample state machine
END_VAR
```

```
// NON COMPLIANT: Integer variable and "magic values" are used to define state machine
CASE nState OF
    0: F_DoSomethingUsefulHere();
    1: F_DoSomethingUsefulHere();
    2: F_DoSomethingUsefulHere();
ELSE
    F_DoSomethingUsefulHere();
END_CASE
```

Positives Beispiel:

```
// Enumeration for the positive sample
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SampleState :
(
    Entry,  // Entrance part of a state
    Work,   // Action part of a state
    Finish  // Exit part of a state
);
END_TYPE
```

```
PROGRAM Sample_pos
VAR
    eState  : E_SampleState;  // Used to operate the sample state machine
END_VAR
```

```
// COMPLIANT: Enum instance and enum values are used to define state machine
CASE eState OF
    E_SampleState.Entry:
        F_DoSomethingUsefulHere();
    E_SampleState.Work:
        F_DoSomethingUsefulHere();
    E_SampleState.Finish:
        F_DoSomethingUsefulHere();
END_CASE
```

Grenzen einer Schleife als Konstante deklarieren

Als Grenzen einer Schleife sollten konstante Variablen verwendet werden. Wenn innerhalb der Schleife auf ein Array zugegriffen wird, sollte das Array über die gleichen konstanten Variablen deklariert sein.

- Untergrenze des Arrays = Untergrenze der Schleife = konstante Variable 1
- Obergrenze des Arrays = Obergrenze der Schleife = konstante Variable 2

Beachten Sie auch die folgenden Empfehlungen:

- Array-Grenzen über Konstanten definieren [▶ 1168]
- Array-Untergrenze von 1 [▶ 1169]

Static Analysis:

Thematisch empfohlene Static Analysis Regeln:

- SA0072: Ungültige Verwendung einer Zählervariablen
- SA0080: Schleifenindexvariable für Arrayindex überschreitet Array-Bereich
- SA0081: Obergrenze ist kein konstanter Wert

Allgemeine Programmelemente für die folgenden Beispiele:

```
TYPE ST_Object :
STRUCT
    sName        : STRING;
    nID          : UINT;
END_STRUCT
END_TYPE
```

```
PROGRAM Sample
VAR CONSTANT
    cMin         : UINT := 1;
    cMax         : UINT := 10;
END_VAR
VAR
    aObjects     : ARRAY[cMin..cMax] OF ST_Object;
    bInitDone    : BOOL;
    nForIdx      : UINT;
END_VAR
```

Negatives Beispiel:

```
VAR
    nUpperBorder : UINT;
END_VAR
```

```
// Initialize objects
IF NOT bInitDone THEN
    nUpperBorder := cMin;

    FOR nForIdx := nUpperBorder TO 10 BY 1 DO
        aObjects[nForIdx].sName := CONCAT('Object_', UDINT_TO_STRING(nForIdx));
        aObjects[nForIdx].nID   := nForIdx;
    END_FOR

    bInitDone := TRUE;
END_IF
```

Positives Beispiel:

```
// Initialize objects
IF NOT bInitDone THEN
    FOR nForIdx := cMin TO cMax BY 1 DO
        aObjects[nForIdx].sName := CONCAT('Object_', UDINT_TO_STRING(nForIdx));
        aObjects[nForIdx].nID   := nForIdx;
    END_FOR

    bInitDone := TRUE;
END_IF
```

In CASE-Anweisung alle Enumerationswerte der Enumerationsvariablen behandeln

In einer CASE-Anweisung sollten Sie alle Enumerationswerte der Enumerationsvariablen behandeln. Falls dies bei einer großen Anzahl von gleichen oder ungenutzten Enumerationswerten nicht sinnvoll sein sollte, kann für diese Fälle ein ELSE-Zweig verwendet werden.

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0075: Fehlendes ELSE
- SA0076: Fehlende Aufzählungskonstante

Allgemeine Programmelemente für die folgenden Beispiele:

```
// Enumeration for all samples in this rule
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_ColorTrafficLight :
(
    Red      := 0,
    Yellow,
    Green
);
END_TYPE
```

```
PROGRAM Sample
VAR
    eColorTrafficLight : E_ColorTrafficLight; // Used to handle the state of the traffic light
END_VAR
```

Negatives Beispiel:

```
CASE eColorTrafficLight OF      // NON COMPLIANT: enum value Green is not handled
    E_ColorTrafficLight.Red:
        SetLightRed();
    E_ColorTrafficLight.Yellow:
        SetLightYellow();
END_CASE
```

Positives Beispiel:

```
CASE eColorTrafficLight OF      // COMPLIANT: all enum values are handled
    E_ColorTrafficLight.Red:
        SetLightRed();
    E_ColorTrafficLight.Yellow:
        SetLightYellow();
    E_ColorTrafficLight.Green:
        SetLightGreen();
END_CASE
```

IF-ELSIF-Anweisungen mit ELSE-Zweig

Aus Sicherheitsgründen wird empfohlen, einem IF-Konstrukt ein ELSE hinzuzufügen, falls es über ein ELSIF verfügen sollte. Ein ELSIF hat empfohlener Weise ein ELSE zur Folge, um zu verdeutlichen, dass alle möglichen Fälle bedacht worden sind. Falls für den ELSE-Zweig keine Anweisungen geplant sind, sollte nach dem ELSE ein Semikolon folgen. Per Kommentar können Sie für die Abnahme/Inbetriebnahme des Programms erläutern, dass der ELSE-Fall bedacht wurde und warum er über keine Anweisungen verfügt.

Allgemeine Programmelemente für die folgenden Beispiele:

```
PROGRAM Sample
VAR
    nTest   : INT:= 10;         // Test value for sample
END_VAR
```

Negatives Beispiel:

```
IF nTest < 10 THEN              // NON COMPLIANT: the ELSE should be used in any way
    nTest := 20;
ELSIF nTest > 20 THEN
    nTest := 10;
END_IF
```

Positives Beispiel 1:

```
IF nTest < 10 THEN              // COMPLIANT with any action in ELSE
    nTest := 20;
ELSIF nTest > 20 THEN
    nTest := 10;
ELSE
    F_DoSomethingUsefulHere(); // it's just a sample
END_IF
```

Positives Beispiel 2:

```
IF nTest < 10 THEN              // COMPLIANT with a comment in ELSE
    nTest := 20;
ELSIF nTest > 20 THEN
    nTest := 10;
ELSE
    ;(* No action needed for the case that nTest is >= 10 and <= 20 *)
END_IF
```

Reihenfolge der IF-Bedingungen

Die Reihenfolge der IF-/ELSIF-/ELSE-Bedingungen entsprechend der Eintrittswahrscheinlichkeit anordnen. So sollte der am wahrscheinlichsten eintretende Fall zuerst abgehandelt werden. Dies verbessert sowohl die Lesbarkeit als auch in einigen Fällen die Performanz, weil nichtzutreffende Abfragen seltener durchlaufen werden.

Positives Beispiel:

```
IF SUCCEEDED(hrErrorCode) THEN
    IF nReqIdx < cIdxMax THEN    // requested index in range
        ; // handle request
    ELSIF nReqIdx = cIdxMax THEN // requested index in range but at upper limit
        ; // handle request
    ELSE
        ; // add error output (error caused by invalid index)
    END_IF
ELSE
    ;     // add error handling (error in previous step)
END_IF
```

18.3.1.2 Fehlercodes

Themenpunkte:

1. Datentyp für Fehlercodes [▶ 1145] [++]
2. Fehlerinformation an Funktionen und Methoden [▶ 1146] [+]
3. Fehlerinformation an Funktionsbausteinen [▶ 1146] [+]

Datentyp für Fehlercodes

Fehlercodes sollten als `hrErrorCode` (Typ HRESULT) spezifiziert und weitergegeben werden. Der Typ HRESULT hat die Besonderheit, dass Fehler durch negative Werte repräsentiert werden. Warnungen oder Informationen können optional mittels positiver Werte ausgegeben werden.

Als Alternative können Sie `nErrorID` (Typ UDINT) spezifizieren.

Deklaration	Fehlerbereich	Kein Fehler	Meldung/Info	Prüffunktionen

hrErrorCode : HRESULT;	<0	>=0	>0	IF SUCCEEDED(hrErrorCode) THEN
				...
				END_IF
				IF FAILED(hrErrorCode) THEN
				...
				END_IF

nErrorID : UDINT;	>0	0		IF nErrorID = 0 THEN
				...
				END_IF
				IF nErrorID <> 0 THEN
				...
				END_IF

Fehlerinformation an Funktionen und Methoden

Funktionen und Methoden benötigen keine Fehlerinformation, sofern kein Fehler eintreten kann. Beispiel:
`c := AddTwoValues(a, b);`

Funktionen und Methoden können in Ausnahmefällen einen Fehlerfall allein mittels Boolean ausgeben, sofern es nur eine einzige Fehlerursache geben kann.

In den meisten Fällen geben Funktionen und Methoden einen Fehler mittels Fehlercode (HRESULT) am Rückgabewert an. Oft gibt ein Funktionsbaustein den Fehler der zuletzt aufgerufenen Methode aus. Siehe hierzu auch den Themenpunkt Fehlerinformation an Funktionsbausteinen [▶ 1146].

Fehlerinformation an Funktionsbausteinen

Funktionsbausteine benötigen keine Fehlerinformation, sofern kein Fehler eintreten kann.

Es wird die Bereitstellung von Fehlerinformationen an Funktionsbausteinen in folgender Weise empfohlen.

Bei kleinen Funktionsbausteinen, welche wenige Fehlerursachen haben, sind folgende Ausgänge ausreichend:

```
VAR_OUTPUT
    bError          : BOOL;         // TRUE if an error occurred.
    hrErrorCode     : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
END_VAR
```

Bei komplexeren Funktionsbausteinen, welche viele verschiedene Fehlerursachen mit ggf. verschiedenen Fehlerquellen haben, wird ein weiterer Ausgang empfohlen:

```
VAR_OUTPUT
    bError          : BOOL;         // TRUE if an error occurred.
    hrErrorCode     : HRESULT;      // '< 0' = error; '> 0' = info; '0' = no error/info
    ipErrorMessage  : I_TcMessage :=
fbErrorMessage; // shows detailed information about occurred errors, warnings and more
END_VAR
VAR
    {attribute 'conditionalshow'}
    fbErrorMessage  : FB_TcMessage;
END_VAR
```

`bError` ermöglicht eine möglichst einfache Abfrage, ob ein Fehlerfall vorliegt. Auch im Online View des SPS-Editors können Sie einen Fehler einfach sehen, sofern er über mehrere Taskzyklen ansteht.

`hrErrorCode` gibt den Fehlercode an und kann als solcher einfach weitergereicht werden. Im Online View des SPS-Editors wird der hexadezimale Wert angezeigt.

`ipErrorMessage` (alternativ `ipResultMessage`) zeigt detaillierte Informationen zum aufgetretenen Fehler. Im Online View des SPS-Editors wird unter anderem ein sprachabhängiger Klartext zur Fehlerbeschreibung angezeigt. Zudem kann diese Nachricht an den TwinCAT 3 Eventlogger übermittelt werden. Eigene Texte für eigene Fehlercodes können mittels eigener Eventklassen in Eventlisten definiert werden. Weitere Informationen finden sich in der Dokumentation der SPS Bibliothek Tc3_EventLogger und dem zugehörigen Beispiel. Am Ausgang des Funktionsbausteins sind die Informationen als Interface bereitgestellt, um den Zugriff auf relevante Inhalte zu begrenzen. Empfohlen wird allerdings, intern dafür zu sorgen, dass der Interfacepointer immer gültig ist.

Zu dem Thema Fehlercodes beachten Sie auch den folgenden Themenpunkt der Programmierkonventionen:

- Zurückgelieferte Fehlerinformationen einer POU auswerten [▶ 1158]

18.3.1.3 Lesbarkeit, Wartbarkeit

Themenpunkte:

1. Keine unbenutzten Deklarationen/Objekte oder unnützen Code [▶ 1147] [++]
2. Keine „magischen Werte/magic numbers“ [▶ 1148] [+]
3. Kein mehrfaches Nutzen gleicher Namen [▶ 1148] [+]
4. Gleiche Notation in Deklaration und Implementierung [▶ 1148] [+]

5. Leere Statements (;) vermeiden [▶ 1149] [+]
6. Jede Zuweisung in separater Codezeile [▶ 1149] [+]

Keine unbenutzten Deklarationen/Objekte oder unnützen Code

Ein Projekt sollte keine unausführbaren Pfade, keinen „Dead“ (unnötigen) Code und keine unbenutzten Typdeklarationen oder Variablen beinhalten.

Nicht verwendete Programmelemente führen in einem Projekt schnell zu unübersichtlichen Codestrukturen. Bei Wartungsarbeiten und Ergänzungen kann die Lesbarkeit des Codes stark erhöht werden, wenn das Projekt nur verwendete Programmelemente enthält.

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0001: Unerreichbarer Code
- SA0002: Leere Objekte
- SA0031: Nicht verwendete Signaturen
- SA0032: Nicht verwendete Aufzählungskonstante
- SA0033: Nicht verwendete Variablen
- SA0035: Nicht verwendete Eingabevariablen
- SA0036: Nicht verwendete Ausgabevariablen

Die Regel SA0033 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Beachten Sie auch die Möglichkeit, Static Analysis Regeln und Namenskonventionen für überprüfte Stellen, an denen eine bestimmte Regel/Namenskonvention nicht mehr gemeldet werden soll, lokal per Pragma oder Attribut zu deaktivieren (siehe auch: Kapitel Pragmas und Attribute in TE1200 PLC Static Analysis). Eine lokale Deaktivierung kommentieren Sie optimalerweise mit entsprechender Begründung.

Beispielsweise können Sie die Regel SA0002 für beabsichtigt leere Rümpfe von Funktionsbausteinen lokal für diesen FB-Rumpf per `{analysis -2}` deaktivieren.

Keine unausführbaren Pfade

Negatives Beispiel:

```
IF FALSE THEN                    // NON COMPLIANT
    F_DoSomethingUsefulHere();   // will never be executed
END_IF
```

Positives Beispiel:

```
IF bTest THEN                    // COMPLIANT
    F_DoSomethingUsefulHere();   // it's just a sample
END_IF
```

Keinen „Dead“ (unnötigen) Code

Negatives Beispiel:

```
FUNCTION F_Sample_neg : INT
VAR_INPUT
    nA       : INT;              // Variable a in term y = a*a + b
    nB       : INT;              // Variable b in term y = a*a + b
END_VAR
VAR
    nTemp    : INT;              // Used for temporary calculation of a*a
END_VAR
```

```
nTemp        := nA * nA;         // NON COMPLIANT: nTemp will not be used later
F_Sample_neg := nA * nA + nB;
```

Positives Beispiel:

```
FUNCTION F_Sample_pos : INT
VAR_INPUT
    nA       : INT;             // Variable a in term y = a*a + b
    nB       : INT;             // Variable b in term y = a*a + b
END_VAR
```

```
F_Sample_pos := nA * nA + nB;   // COMPLIANT: no dead code in this sample
```

Keine „magischen Werte/magic numbers“

Verwenden Sie keine „magischen Werte/magic numbers“.

Bei Vergleichen oder Zuweisungen können alternativ Konstanten oder anderweitig fest definierte Werte genutzt werden. Auch Texte, wie Ausgabe- oder Vergleichstexte, sollten z. B. in Konstanten gespeichert und nicht direkt im Quelltext definiert werden.

Beachten Sie auch den folgenden Themenpunkt der Programmierkonventionen:

- CASE-Anweisung nur mit Enumerationsinstanz [▶ 1142]

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    nValue   : INT;   // Sample INT-variable that is used for comparison
    bValueOk : BOOL;  // Indicates if the value of sample INT-variable is OK
END_VAR
```

```
// NON COMPLIANT: A "magic value" is used for comparison
bValueOk := (nValue = 125);
```

Positives Beispiel:

```
PROGRAM Sample_pos
VAR
    nValue    : INT;            // Sample INT-variable that is used for comparison
    bValueOk  : BOOL;           // Indicates if the value of sample INT-variable is OK
END_VAR
VAR CONSTANT
    cValueOk  : INT   := 125;   // Used to validate the sample INT-variable
END_VAR
```

```
// COMPLIANT: A constant variable is used for comparison
bValueOk := (nValue = cValueOk);
```

Kein mehrfaches Nutzen gleicher Namen

Vermeiden Sie das mehrfache Nutzen gleicher Namen. Dazu zählen folgende Fälle:

- Der Name einer Enumerationskonstanten verglichen mit dem Namen einer Variablen
- Die Namen von Objekten untereinander
- Der Name eines Objekts verglichen mit dem Namen einer Variablen

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0013: Deklarationen mit gleichem Variablennamen
- SA0027: Mehrfachverwendung des Namens

Die Regel SA0027 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Gleiche Notation in Deklaration und Implementierung

Aus Gründen der Einheitlichkeit sowie der Les- und Wartbarkeit sollten Sie in der Deklaration und in der Implementierung die gleiche Notation von Programmelementen und Variablen verwenden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0029: Notation in Implementierung und Deklaration unterschiedlich

Allgemeine Programmelemente für die folgenden Beispiele:

```
FUNCTION F_Sample
```

Negatives Beispiel:

```
PROGRAM Sample
VAR
    bTest  : BOOL;
END_VAR
-------------------------
IF btest THEN
    f_Sample();
END_IF
```

Positives Beispiel:

```
PROGRAM Sample
VAR
    bTest  : BOOL;
END_VAR
-------------------------
IF bTest THEN
    F_Sample();
END_IF
```

Leere Statements (;) vermeiden

Leere Statements (;)vermeiden, um den Code nicht unnötig auszudehnen. Falls ein leeres Statement zur Verdeutlichung eines bestimmten Falls unbedingt erforderlich sein sollte, sollte das leere Statement in einer eigenen Zeile stehen. Zusätzlich sollten Sie kommentieren, warum dieser Programmzweig leer ist.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0003: Leere Anweisungen

Allgemeine Programmelemente für die folgenden Beispiele:

```
PROGRAM Sample
VAR
    nTest : INT := 10;  // Test value used in sample
END_VAR
```

Negatives Beispiel:

```
IF nTest = 10 THEN
    ;                   // NON COMPLIANT without a useful comment
ELSE
    nTest := 10;
END_IF
```

Positives Beispiel:

```
IF nTest = 10 THEN
    ;                   // COMPLIANT with a comment: In case that nTest is equal to 10, no action is
 needed. This status is intended.
ELSE
    nTest := 10;
END_IF
```

Jede Zuweisung in separater Codezeile

Jede Zuweisung sollte sich in einer separaten Codezeile befinden. Somit sollten sich keine Zuweisungen in Bedingungen befinden und Zuweisungsoperatoren sollten keine Unterausdrücke beinhalten.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0095: Zuweisung in Bedingung

Allgemeine Programmelemente für die folgenden Beispiele:

```
PROGRAM Sample
VAR
    bVar1 : BOOL;
    bVar2 : BOOL;
    nA    : INT;
    nB    : INT;
    nC    : INT;
END_VAR
```

Negatives Beispiel:

```
// NON COMPLIANT: assignment in condition
IF bVar1 := bVar2 THEN
    DoSomething();
END_IF

// NON COMPLIANT: more than one assignment in a single line
nA := nC * (nB := nC + nC);
```

Positives Beispiel:

```
IF bVar1 = bVar2 THEN
    DoSomething();
END_IF

nB := nC + nC;
nA := nC * nB;
```

18.3.2 Bibliotheken

18.3.2.1 Entwicklung von Bibliotheken

Themenpunkte:

1. Bibliotheksnamen [▶ 1150] [++]
2. Versionierung [▶ 1150] [++]
3. Kapselung interner Typdefinitionen [▶ 1151] [+]
4. Bezeichner in Bibliotheken [▶ 1151] [+]

Bibliotheksnamen

Beckhoff SPS-Bibliotheken beginnen mit dem Präfix **Tc** (z. B.: **Tc3_EventLogger**), besitzen die Dateinamenerweiterung *.**compiled-library** und sind im Bibliotheks-Repository eingebunden.

Wählen Sie Bibliotheksnamen kurz, aber dennoch unmissverständlich.

Der Namespace einer Beckhoff SPS-Bibliothek ist identisch mit dem Bibliotheksnamen.

Der Standard-Platzhalter einer Beckhoff SPS-Bibliothek ist identisch mit dem Bibliotheksnamen.

Versionierung

Jede SPS-Bibliothek besitzt eine vierstellige Versionsnummer (Major.Minor.Build.Revision).

Major	Die erste Stelle wird inkrementiert, wenn die neue Version inkompatibel zur vorherigen Version ist. Dabei werden die Stellen für Minor und Build auf eins zurückgesetzt.
Minor	Die zweite Stelle wird inkrementiert, wenn die neue Version Funktionserweiterungen beinhaltet. Dabei wird die Stelle für Build auf eins zurückgesetzt.
Build (Patch)	Die dritte Stelle wird inkrementiert, wenn die neue Version nur Fehlerbehebungen beinhaltet.
Revision	Die vierte Stelle ist immer Null, sofern es sich um eine Release-Version handelt.

Ist eine SPS-Bibliothek noch im Beta-Status, so handelt es sich um die Major Version null bzw. um eine Versionsnummer 0.x.y.z. Eine Release-Version hat mindestens die Versionsnummer 1.1.1.0.

Sollten andere Programmteile eine dreistellige Versionierung (Major.Minor.Patch) vorsehen, so können die ersten drei Stellen entsprechend verwendet werden (Build = Patch).

Kapselung interner Typdefinitionen

Um die Verwendung interner Typdefinitionen von außerhalb der Bibliothek zu verhindern und um dem Anwender die für ihn relevanten POUs und DUTs zu verdeutlichen, wird in SPS-Bibliotheken Kapselung verwendet. Hierzu wird der Access-Specifier INTERNAL empfohlen.

Bei einzelnen Methoden von Funktionsbausteinen kann ebenso der Access-Specifier PRIVATE Anwendung finden. Um lokale Variablen von Funktionsbausteinen zu verstecken, kann das Attribut 'conditionalshow all locals' [▶ 834] verwendet werden.

Bezeichner in Bibliotheken

Um Compiler-Fehler zu doppelten Bezeichnern zu verhindern und eine Verwendung der Bibliothek ohne verpflichtenden Namespace zu ermöglichen, wird die Verwendung eines zusätzlichen Kürzels in den Bezeichnern empfohlen. Diese Abkürzung des Bibliotheksnamens wird zwischen das Präfix und den eigentlichen Namen des jeweiligen Typen gesetzt.

Ebenso sollten globale Variablenlisten einer Bibliothek mit dem Kürzel ergänzt werden, um mehrfache Existenz von Listen namens 'GVL' zu vermeiden.

Beispiele (aus der SPS-Bibliothek Tc3_Filter):

```
STRUCT ST_FTR_PT1
ENUM E_FTR_Type
FUNCTION_BLOCK FB_FTR_PT1
VAR_GLOBAL GVL_FTR
```

Siehe auch:

- Bibliotheken verwenden

18.3.2.2 Verwendung von Bibliotheken

Themenpunkte:

1. Bibliotheksverwendung [▶ 1151] [++]
2. Versionsüberprüfung [▶ 1151] [++]
3. Fixieren von Bibliotheksversionen [▶ 1151] [+]

Bibliotheksverwendung

SPS-Bibliotheken werden als Platzhalter referenziert.

Details finden Sie im Kapitel Bibliotheksplatzhalter [▶ 293] der TwinCAT-3-PLC-Dokumentation.

Versionsüberprüfung

SPS-Bibliotheken beinhalten eine globale Konstante vom Typ ST_LibVersion [▶ 826], in der Sie Auskünfte zur Bibliotheksversion finden können. Diese Konstante kann während der Laufzeit gelesen und mithilfe der Funktion F_CmpLibVersion (siehe Tc2_System-Bibliothek) mit einer benötigten Bibliotheksversion verglichen werden.

Fixieren von Bibliotheksversionen

Beim Anfügen einer Referenz als Bibliotheksplatzhalter wird die Bibliothek mit der Version ‚* / newest version' angefügt.

Sobald ein Archiv (zB. SPS Archiv oder TwinCAT Archiv) erstellt wird oder die TwinCAT-Konfiguration auf dem Zielsystem aktiviert wird, sollten die Bibliotheksversionen jedoch fixiert werden. Dies erreichen Sie am einfachsten mit dem Befehl 'Set to Effective Version' im Kontextmenü des Referenzknotens.

Beachten Sie auch die weiteren Informationen zum Thema Projektauslieferung [▶ 39].

Siehe auch:

- Bibliotheken verwenden

18.3.3 DUTs

18.3.3.1 Implementierung von DUTs

Themenpunkte:

1. Attribute 'qualified_only' und 'strict' bei Enumeration verwenden [▶ 1152] [++]
2. Bei Enumeration keinen, nur den ersten oder alle Werte initialisieren [▶ 1152] [+]

Attribute 'qualified_only' und 'strict' bei Enumeration verwenden

Bei der Definition einer Enumeration verwenden Sie die Attribute {attribute 'qualified_only'} [▶ 859] und {attribute 'strict'} [▶ 860].

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0025: Unqualifizierte Enumerationskonstanten
- SA0171: Enumerationen sollten das Attribut 'strict' haben

Negatives Beispiel:

```
TYPE E_SignalColor :
(
    Red,
    Yellow,
    Green
);
END_TYPE
```

Positives Beispiel:

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red,
    Yellow,
    Green
);
END_TYPE
```

Bei Enumeration keinen, nur den ersten oder alle Werte initialisieren

Bei einer Enumeration initialisieren Sie keinen der Werte, nur den ersten oder alle Werte mit dem Zuweisungsoperator ':='.

Negatives Beispiel:

```
// NON COMPLIANT: init none, the first or all enumerators
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red    := 0,
    Yellow := 1,
```

```
    Green
);
END_TYPE
```

Positives Beispiel 1:

```
// COMPLIANT: no enumerator is initialized
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red,
    Yellow,
    Green
);
END_TYPE
```

Positives Beispiel 2:

```
// COMPLIANT: first enumerator is initialized
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red  := 0,
    Yellow,
    Green
);
END_TYPE
```

Positives Beispiel 3:

```
// COMPLIANT: all enumerators are initialized
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SignalColor :
(
    Red    := 0,
    Yellow := 1,
    Green  := 2
);
END_TYPE
```

Siehe auch:

- Objekt DUT

18.3.3.2 Verwendung von DUTs

Themenpunkte:

1. Enumerationsvariablen nur zugehörige Enumeratoren zuweisen [▶ 1153] [++]

Enumerationsvariablen nur zugehörige Enumeratoren zuweisen

Der Instanz einer Enumeration sollten nur die Enumeratoren der dazugehörigen Enumeration zugewiesen werden. Um diese Regel zu erzwingen, wird empfohlen, bei Typdefinition der Enumeration das Attribut {attribute 'strict'} [▶ 860] zu setzen. Siehe auch: Attribute 'qualified_only' und 'strict' bei Enumeration verwenden [▶ 1152]

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0034: Enumerationsvariablen mit falscher Zuweisung
- SA0171: Enumerationen sollten das Attribut 'strict' haben

Negatives Beispiel:

```
{attribute 'qualified_only'}
TYPE E_ColorTrafficLight :
(
```

```
    Red    := 0,
    Yellow := 1,
    Green  := 2
);
END_TYPE
```

```
PROGRAM Sample_neg
VAR
    eColorTrafficLight : E_ColorTrafficLight;  // Used to handle the state of the traffic light
END_VAR
```

```
CASE eColorTrafficLight OF
    E_ColorTrafficLight.Green:
        eColorTrafficLight  := E_ColorTrafficLight.Yellow;
    E_ColorTrafficLight.Yellow:
        eColorTrafficLight  := E_ColorTrafficLight.Red;
    E_ColorTrafficLight.Red:
        eColorTrafficLight  := 2;              // NON COMPLIANT: Only values defines in E_StandardCo
lor should be assigned to enum instance eColorTrafficLight
ELSE
    eColorTrafficLight := E_ColorTrafficLight.Red;
END_CASE
```

Positives Beispiel:

```
{attribute 'strict'}
{attribute 'qualified_only'}
TYPE E_ColorTrafficLight :
(
    Red    := 0,
    Yellow := 1,
    Green  := 2
);
END_TYPE
```

```
PROGRAM Sample_pos
VAR
    eColorTrafficLight : E_ColorTrafficLight;             // Used to handle the state of the traffic
 light
END_VAR
```

```
CASE eColorTrafficLight OF
    E_ColorTrafficLight.Green:
        eColorTrafficLight  := E_ColorTrafficLight.Yellow;
    E_ColorTrafficLight.Yellow:
        eColorTrafficLight  := E_ColorTrafficLight.Red;
    E_ColorTrafficLight.Red:
        eColorTrafficLight  := E_ColorTrafficLight.Green; // COMPLIANT
ELSE
    eColorTrafficLight := E_ColorTrafficLight.Red;
END_CASE
```

Siehe auch:

- Objekt DUT [▶ 77]

18.3.4 POUs

18.3.4.1 Implementierung von Funktionen, Methoden und Aktionen

Themenpunkte:

1. Kein „Call by value“ von großen Parametern in Funktionen/Methoden [▶ 1155] [++]
2. Keine großen Variablen in Funktionen/Methoden deklarieren [▶ 1155] [++]
3. Keine Aktionen verwenden [▶ 1156] [+]
4. Alle Parameter einer Funktion/Methode intern verwenden [▶ 1156] [+]
5. Rückgabewert einer Funktion/Methode nur an einer Stelle zuweisen [▶ 1156] [+]
6. Zugriff auf Methoden so weit wie möglich einschränken [▶ 1158] [+]
7. Gruppierung von Parametern als Struktur [▶ 1158] [+]

Kein „Call by value“ von großen Parametern in Funktionen/Methoden

Eingangs- und Ausgangsparameter sowie der Rückgabewert einer Methode werden in den meisten Fällen beim Aufruf der Funktion/Methode kopiert, das ist der „Call by value“. Bei Parametern, die viel Speicher belegen, benötigt dieser Kopierschritt mehr Zeit. Um effizienten Programmcode zu schreiben, sollten große Parameter mittels „Call by reference“ übergeben werden. So wird nur ein Zeiger gesetzt. Hierzu gibt es einige verschiedene Möglichkeiten, welche im Beispiel aufgegriffen werden. Beachten Sie hierbei, dass manche Möglichkeiten auch schreibenden Zugriff auf die übergebenen Eingangsparameter erlauben bzw. ermöglichen.

Siehe auch „Übergabe von großen Strings [▶ 1176]“.

Negatives Beispiel:

```
METHOD SampleMethod_neg : ST_DataCollection    // return value with call by value
VAR_INPUT
    aSensor1Data     : ARRAY[1..100] OF LREAL; // input with call by value
    aSensor2Data     : ARRAY[1..200] OF LREAL; // input with call by value
    aSensor3Data     : ARRAY[1..300] OF LREAL; // input with call by value
    aSensor4Data     : ARRAY[1..400] OF LREAL; // input with call by value
END_VAR
VAR_OUTPUT
    aOutputData      : ARRAY[1..500] OF LREAL; // output with call by value
END_VAR
```

Positives Beispiel:

```
METHOD SampleMethod_pos : HRESULT
VAR_INPUT
    aSensor1Data     : REFERENCE TO ARRAY[1..100] OF LREAL; // input with call by reference (write
access possible)
    pSensor2Data     : POINTER TO ARRAY[1..200] OF LREAL;   // input buffer, similar to call by
reference (write access possible)
    nSensor2DataSize : UDINT;                               // size in bytes of buffer to ensure the
correct buffer size
    aOutputData      : REFERENCE TO ARRAY[1..500] OF LREAL; // output with call by reference
    stDataCollection : REFERENCE TO ST_DataCollection;      // output with call by reference
END_VAR
VAR_IN_OUT CONSTANT
    aSensor3Data     : ARRAY[1..300] OF LREAL;              // input with call by reference
END_VAR
VAR_IN_OUT
    aSensor4Data     : ARRAY[1..400] OF LREAL;              // input with call by reference (write
access possible)
END_VAR
```

Keine großen Variablen in Funktionen/Methoden deklarieren

Bei Variablendeklarationen innerhalb einer Funktion/Methode sollten Sie eine große Anzahl sowie eine große Datengröße vermeiden. Solche Variablendeklarationen werden aus dem Stack-Speicher genommen. Werden Funktionen/Methoden verschachtelt aufgerufen, so addiert sich die benötigte Speichermenge auf dem Stack. Es ist nicht beliebig viel Stack-Speicher verfügbar und „Stack Overflow“-Situationen sollten Sie im Vorhinein vermeiden.

Die maximale Größe des Stack-Speichers wird im TwinCAT-Projekt unterhalb vom Knoten-System im Knoten Real-Time angegeben. Oft sind dies 64 KB. Deshalb empfiehlt sich, die gesamte deklarierte Datenmenge in einer Funktion/Methode möglichst klein, beispielsweise unter 1 KB, zu halten. Achten Sie daher besonders bei STRING-Typen und Arrays beliebiger Typen auf möglichst geringe Längen.

Alternativ können Variablen innerhalb der Funktionsbausteininstanz deklariert werden. Beachten Sie hierbei, dass diese Variablen dauerhaft zur Verfügung stehen und nicht bei jedem Aufruf der Methode initialisiert werden.

Für Parameter einer Funktion/Methode gilt das gleiche. Siehe hierzu „Kein Call-By-Value von großen Parametern in Funktionen/Methoden [▶ 1155]“.

Negatives Beispiel:

```
FUNCTION_BLOCK FB_Sample_neg
VAR
END_VAR
```

```
METHOD SampleMethod_neg : LREAL
VAR
    fSum     : LREAL;
    nCnt     : UINT;
    aLogList : ARRAY[1..50] OF STRING(1023); // locally declared in method leads to stack allocation
END_VAR
```

Positives Beispiel:

```
FUNCTION_BLOCK FB_Sample_pos
VAR
    aLogList : ARRAY[1..50] OF STRING(1023); // declared as member of the FB instance
END_VAR
```

```
METHOD SampleMethod_pos : LREAL
VAR
    fSum     : LREAL;
    nCnt     : UINT;
END_VAR
```

Positives Beispiel (Alternative):

```
FUNCTION_BLOCK FB_Sample2_pos
VAR
END_VAR
```

```
METHOD SampleMethod2_pos : LREAL
VAR
    fSum     : LREAL;
    nCnt     : UINT;
END_VAR
VAR_INST
    aLogList : ARRAY[1..50] OF STRING(1023); // declared as member of the FB instance
END_VAR
```

Keine Aktionen verwenden

Ein Programm oder ein Funktionsbaustein sollte keine Aktionen beinhalten. Implementieren Sie stattdessen Methoden. Diese können je nach Verwendung als PUBLIC, PRIVATE oder PROTECTED definiert werden.

Alle Parameter einer Funktion/Methode intern verwenden

Alle Parameter einer Funktion oder einer Methode sollten intern auch verwendet werden.

Negatives Beispiel:

```
FUNCTION F_Sample_neg : INT
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
    nB       : INT;          // NON COMPLIANT: nB will not be used
END_VAR
```

```
F_Sample_neg := nA * nA;
```

Positives Beispiel:

```
FUNCTION F_Sample_pos : INT  // COMPLIANT: no unused parameter
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
END_VAR
```

```
F_Sample_pos := nA * nA;
```

Rückgabewert einer Funktion/Methode nur an einer Stelle zuweisen

Den Rückgabewert einer Funktion/Methode sollten Sie nur an einer Stelle zuweisen. Eine Ausnahme dieser Regel gilt, wenn die Zuweisungen in verschiedenen Zweigen einer IF- bzw. CASE-Anweisung durchgeführt werden.

Beispiel 1:

Negatives Beispiel:

```
FUNCTION F_Sample_neg_1 : LREAL
VAR
    fOnePlusHalfEpsilon  : LREAL;          // Auxiliary variable to calculate and validate the machi
ne epsilon
END_VAR
```

```
// NON COMPLIANT: F_Sample_neg_1 is assigned or used more than once
F_Sample_neg_1            := 1.0;

REPEAT
    F_Sample_neg_1       := 0.5 * F_Sample_neg_1;        // NON COMPLIANT: see above
    fOnePlusHalfEpsilon  := 1.0 + 0.5 * F_Sample_neg_1;  // NON COMPLIANT: see above
UNTIL fOnePlusHalfEpsilon <= 1.0
END_REPEAT;
```

Positives Beispiel:

```
FUNCTION F_Sample_pos_1 : LREAL
VAR
    fOnePlusHalfEpsilon  : LREAL;          // Auxiliary variable to calculate and validate the machi
ne epsilon
    fResult              : LREAL := 1.0;   // Temporary variable to
END_VAR
```

```
REPEAT
    fResult              := 0.5 * fResult;
    fOnePlusHalfEpsilon  := 1.0 + 0.5 * fResult;
UNTIL fOnePlusHalfEpsilon <= 1.0
END_REPEAT;

F_Sample_pos_1 := fResult;                 // COMPLIANT: F_Sample_pos_1 is only assigned here
```

Beispiel 2:

Allgemeine Programmelemente für dieses Beispiel:

```
// Enumeration for sample 2
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE E_SampleStatus :
(
    Ok,                         // Used when the status is OK
    Warning,                    // Used when a warning occurs
    Error                       // Used when an error occurs
);
END_TYPE
```

```
// GVL_Sample - global variable list for sample 2
VAR_GLOBAL CONSTANT
    cLimitMin    : INT := 10;  // Minimal limit to validate a sample value
    cLimitMax    : INT := 20;  // Maximal limit to validate a sample value
    cTolerance   : INT := 2;   // Upper and lower tolerance to validate a sample value
END_VAR
```

```
// Returns a status variable (ok, warning, error) for a measure value, using a constant range of val
ues defined on a GVL
FUNCTION F_Sample : E_SampleStatus
VAR_INPUT
    nValue       : INT;        // Sample value that is validated in respect of a range of values
END_VAR
```

Negatives Beispiel:

```
// NON COMPLIANT: F_Sample is assigned within different IF-
statements. Multiple write access on return value cannot be excluded.
IF (nValue >= cLimitMin) AND (nValue <= cLimitMax) THEN
    F_Sample := E_SampleStatus.Ok;
END_IF

IF (nValue >= (cLimitMin - cTolerance))
AND (nValue <= (cLimitMax + cTolerance)) THEN
    F_Sample := E_SampleStatus.Warning;
END_IF

IF (nValue < (cLimitMin - cTolerance))
OR (nValue > (cLimitMax + cTolerance))
    F_Sample := E_SampleStatus.Error;
END_IF
```

Positives Beispiel:

```
// COMPLIANT: F_Sample is only assigned within one IF-
statement. No multiple write access on return value occurs.
IF (nValue >= cLimitMin) AND (nValue <= cLimitMax) THEN
    F_Sample := E_SampleStatus.Ok;
ELSIF (nValue >= (cLimitMin - cTolerance))
  AND (nValue <= (cLimitMax + cTolerance)) THEN
    F_Sample := E_SampleStatus.Warning;
```

```
ELSE
    F_Sample := E_SampleStatus.Error;
END_IF
```

Zugriff auf Methoden so weit wie möglich einschränken

Der Zugriff auf Methoden sollte mit Hilfe der Zugriffsmodifizierer PRIVATE oder PROTECTED so weit wie möglich eingeschränkt werden. Methoden, die nur für den internen Aufruf vorgesehen sind, können somit nicht von außen aufgerufen werden. Dies trägt dazu bei, dass ein Funktionsbaustein eher auf die Weise verwendet wird, wie es seinem vorgesehenen Verwendungszweck entspricht. Die Kapselung von Methoden führt daher zu sichererem Code. Zudem sind nachträgliche Änderungen an internen Methoden einfacher möglich, wenn diese, wie vorgesehen, vom Anwender nicht von außen aufgerufen wurden.

Gruppierung von Parametern als Struktur

Müssen Sie viele Parameter beim Aufruf angeben, so bietet es sich an, diese in einer oder mehreren Strukturen zu gruppieren.

Dies hat den Vorteil, dass innerhalb der Strukturdefinition Default-Werte hinterlegt sind. Mehrere Konstanten des Strukturdatentyps können wiederum verschiedene Default-Werte für unterschiedliche Anwendungsfälle bereitstellen.

Der Aufruf kann effizient umgesetzt werden, indem die Struktur mittels „Call by reference“ übergeben wird. So werden viele einzelne „Call by value“-Übergaben vermieden. Sehen Sie dazu auch den Themenpunkt Kein „Call by value“ von großen Parametern in Funktionen/Methoden [▶ 1155].

Siehe auch:

- Objekt Methode [▶ 93]
- Objekt Funktion [▶ 83]

18.3.4.2 Verwendung von Funktionen und Methoden

Themenpunkte:

1. Zurückgelieferte Fehlerinformationen einer POU auswerten [▶ 1158] [++]
2. Rückgabewert einer Funktion/Methode verwenden [▶ 1159] [+]
3. Funktionen/Methoden nicht in sich selbst aufrufen [▶ 1160] [+]

Zurückgelieferte Fehlerinformationen einer POU auswerten

Wenn eine Funktion, eine Methode oder ein Funktionsbaustein Fehlerinformationen liefert, werten Sie diese immer aus.

Static Analysis:

Thematisch empfohlene Static Analysis Regeln:

- SA0009: Nicht verwendete Rückgabewerte
- SA0169: Ignorierte Ausgänge

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    fbFileOpen    : FB_FileOpen;      // FileOpen-FB for logger
    bFileOpenExec : BOOL;             // Execute FileOpen-FB
END_VAR
```

```
// NON COMPLIANT: error information of fbFileOpen will not be used
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
```

```
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S);
```

Positives Beispiel:

```
PROGRAM Sample_pos
VAR
    fbFileOpen        : FB_FileOpen;  // FileOpen-FB for logger
    bFileOpenExec     : BOOL;         // Execute FileOpen-FB
    bFileOpenError    : BOOL;         // Error flag of FileOpen-FB
    nFileOpenErrorID  : UDINT;        // Error code of FileOpen-FB
END_VAR
```

```
// COMPLIANT: error information will be handled
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S,
    bError    => bFileOpenError,
    nErrId    => nFileOpenErrorID);

IF bFileOpenError THEN
    F_DoSomethingUsefulHere();        // Handle error here
END_IF
```

Rückgabewert einer Funktion/Methode verwenden

Der Rückgabewert einer Funktion/Methode sollte an der Aufrufstelle der Funktion/Methode verwendet, d.h. abgefragt und ausgewertet, werden. Dies ist insbesondere sinnvoll, wenn auf diese Weise ein Fehlerwert zurückgegeben wird. Allerdings sind auch Ausnahmen möglich, bei denen der Rückgabewert nicht bei jedem Aufruf der Funktion/Methode verwendet werden muss.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0009: Nicht verwendete Rückgabewerte

Allgemeine Programmelemente für diese Regel:

```
(* Function for all samples in this rule: Adds a message to logger system.
   Returns TRUE if successful, FALSE on Error. *)
FUNCTION F_AddLogMessage : BOOL
VAR_INPUT
    sMessage        : WSTRING;        // Message to be logged
    dtTimestamp     : DATE_AND_TIME;  // Timestamp of the message
END_VAR
```

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    dtNow           : DATE_AND_TIME;  // Actual system time
END_VAR
```

```
// NON COMPLIANT: return value of function will not be used
F_AddLogMessage(sMessage := "Test Message", dtTimestamp := dtNow);
```

Positives Beispiel:

```
PROGRAM Sample_pos
VAR
    dtNow           : DATE_AND_TIME;  // Actual system time
    bSendMessageOk  : BOOL;           // Used to check if sending of message was successful
END_VAR
```

```
// COMPLIANT: return value of function will be used
bSendMessageOk := F_AddLogMessage(sMessage := "Test Message", dtTimestamp := dtNow);

IF NOT bSendMessageOk THEN
    F_DoSomethingUsefulHere();        // Handle error here
END_IF
```

Funktionen/Methoden nicht in sich selbst aufrufen

Funktionen/Methoden sollten sich nicht direkt oder indirekt selbst aufrufen, um Rekursionen zu vermeiden. Auch in anderen Programmiersprachen als der IEC61131 sollten Sie Rekursionen nur mit Bedacht einsetzen.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0160: Rekursive Aufrufe

Negatives Beispiel:

```
FUNCTION F_Sample_neg : DWORD
VAR_INPUT
    nFac : DWORD;     // The faculty of this value will be calculated
END_VAR
-------------------------------------------------------------------
IF nFac = 0 THEN
    F_Sample_neg := 1;
ELSE
    // NON COMPLIANT: implicit recursion. F_Sample_neg_IndirectFac calls F_Sample_neg
    F_Sample_neg := nFac * F_Sample_neg_IndirectFac(nFac := (nFac - 1));
END_IF
-------------------------------------------------------------------
-------------------------------------------------------------------
FUNCTION F_Sample_neg_IndirectFac : DWORD
VAR_INPUT
    nFac : DWORD;     // The faculty of this value will be calculated
END_VAR
-------------------------------------------------------------------
F_Sample_neg_IndirectFac := F_Sample_neg(nFac := nFac);
```

Positives Beispiel:

```
FUNCTION F_Sample_pos : DWORD
VAR_INPUT
    nFac : DWORD;     // The faculty of this value will be calculated
END_VAR
VAR
    nTemp   : DWORD;  // Temporary variable used to calculate faculty
    nCount  : DWORD;  // Counter variable used to calculate faculty
END_VAR
-------------------------------------------------------------------
nTemp := 1;

IF nFac > 0 THEN
    FOR nCount := 1 TO nFac DO
        nTemp := nTemp * nCount;
    END_FOR
END_IF

F_Sample_pos := nTemp;
```

Siehe auch:

- Objekt Methode [▶ 93]
- Objekt Funktion [▶ 83]

18.3.4.3 Implementierung von Funktionsbausteinen

Themenpunkte:

1. Online-Change-Fähigkeit sicherstellen [▶ 1161] [++]
2. Einheitliche Schnittstelle bei einmaliger asynchroner Abarbeitung [▶ 1161] [+]
3. Einheitliche Schnittstelle bei kontinuierlicher asynchroner Abarbeitung [▶ 1161] [+]
4. Alle Parameter eines Funktionsbausteins intern verwenden [▶ 1162] [+]
5. Gruppierung von Parametern als Struktur [▶ 1162] [+]

Online-Change-Fähigkeit sicherstellen

Die Möglichkeit zum Online-Change ist ein wichtiges grundlegendes Feature von SPS-Anwendungen. Ein SPS-Programm kann mittels Online-Change auf verschiedene Arten verändert werden, ohne im zyklischen Ablauf gestoppt zu werden. Häufig wird Programmcode im Implementierungsteil verändert, wozu auch im Deklarationsteil einzelne Variablen ergänzt werden.

Implementieren Sie Funktionsbausteine so, dass Instanzen in ihrer Funktionsfähigkeit nicht beeinträchtigt werden, sollten diese in Folge eines Online-Change im Speicher verschoben werden.

Details finden Sie im Kapitel Ausführen eines Online-Change [▶ 263] in der TwinCAT-3-PLC-Dokumentation

Einheitliche Schnittstelle bei einmaliger asynchroner Abarbeitung

Zusätzlich zu den Namenskonventionen für Variablen [▶ 1134] und der Reihenfolge der Textblöcke bei Variablendeklarationen [▶ 1125] erleichtert eine einheitliche Schnittstelle die Verwendung von Funktionsbausteinen.

Bei Funktionsbausteinen, deren Rumpfaufruf eine asynchrone und einmalige ADS-Abarbeitung übernimmt, sollten Sie folgende Schnittstelle verwenden. Sollte es sich nicht um eine ADS-Kommunikation handeln, wird der Parameter `sNetId` weggelassen.

Die Abarbeitung startet bei steigender Flanke des Eingangs `bExecute`. Dies ist nur möglich, wenn der Funktionsbaustein nicht `bBusy` ist. Eingangsparameter sollen vom Anwender während einer laufenden Abarbeitung nicht verändert werden. Dies stellt auch sicher, dass die Ergebnisse der Abarbeitung (Ausgänge) zu den angegebenen Parametern (Eingänge) passen.

Ist die Abarbeitung beendet, wird `bBusy := FALSE` gesetzt. Spätestens jetzt müssen Sie alle Ausgangsparameter setzen. Der Anwender hat so die Möglichkeit, auf `bBusy = FALSE` zu reagieren und daraufhin die Ausgänge auszuwerten.

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    bExecute      : BOOL;
    ...           : ...;                          // optional inputs
    ...           : ...;                          // optional inputs
    tTimeout      : TIME := DEFAULT_ADS_TIMEOUT; // ADS communication timeout
    sNetId        : T_AmsNetId := '';             // keep empty '' for the local device
END_VAR
VAR_OUTPUT
    bBusy         : BOOL;
    bError        : BOOL;
    hrErrorCode   : HRESULT;
    ...           : ...;                          // optional outputs
    ...           : ...;                          // optional outputs
END_VAR
```

Einheitliche Schnittstelle bei kontinuierlicher asynchroner Abarbeitung

Zusätzlich zu den Namenskonventionen für Variablen [▶ 1134] und der Reihenfolge der Textblöcke bei Variablendeklarationen [▶ 1125] erleichtert eine einheitliche Schnittstelle die Verwendung von Funktionsbausteinen.

Bei Funktionsbausteinen, deren Rumpfaufruf eine asynchrone und kontinuierliche ADS-Abarbeitung übernimmt, verwenden Sie die folgende Schnittstelle. Sollte es sich nicht um eine ADS-Kommunikation handeln, wird der Parameter `sNetId` weggelassen.

Die Abarbeitung startet bei steigender Flanke von `bEnable`. Der Funktionsbaustein arbeitet so lange wie `bEnable` gesetzt ist. Ein Zurücksetzen von `bEnable` hat aufgrund der asynchronen Abarbeitung nicht zwangsweise die sofortige Beendigung jeglicher Abarbeitung zur Folge. Diese wird mittels `bBusy = FALSE` angezeigt. Während `bEnable` gesetzt ist, können geänderte Eingangsparameter verarbeitet werden.

Der Ausgang `bValid` zeigt die erfolgreiche Abarbeitung und die Gültigkeit optionaler Ausgangsparameter an.

Der Ausgang `bError` zeigt einen eingetretenen Fehler an. Nähere Information liefert der Fehlercode am Ausgang `hrErrorCode`. Weil `bError` bei kontinuierlicher Abarbeitung jederzeit gesetzt werden kann und oft nur einen Zyklus lang ansteht, soll der Anwender diesen Ausgang dauerhaft abfragen.

```
FUNCTION_BLOCK FB_Sample
VAR_INPUT
    bEnable       : BOOL;
    ...           : ...;                          // optional inputs
    ...           : ...;                          // optional inputs
    tTimeout      : TIME := DEFAULT_ADS_TIMEOUT; // ADS communication timeout
    sNetId        : T_AmsNetId := '';             // keep empty '' for the local device
END_VAR
VAR_OUTPUT
    bValid        : BOOL;
    bBusy         : BOOL;
    bError        : BOOL;
    hrErrorCode   : HRESULT;
    ...           : ...;                          // optional outputs
    ...           : ...;                          // optional outputs
END_VAR
```

Alle Parameter eines Funktionsbausteins intern verwenden

Alle Parameter eines Funktionsbausteins sollten auch intern verwendet werden.

Negatives Beispiel:

```
FUNCTION FB_Sample_neg
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
    nB       : INT;          // NON COMPLIANT: nB will not be used
END_VAR
VAR_OUTPUT
    nY       : INT;
END_VAR
```

```
nY := nA * nA;
```

Positives Beispiel:

```
FUNCTION FB_Sample_pos       // COMPLIANT: no unused parameter
VAR_INPUT
    nA       : INT;          // Variable a in term y = a*a
END_VAR
VAR_OUTPUT
    nY       : INT;
END_VAR
```

```
nY := nA * nA;
```

Gruppierung von Parametern als Struktur

Müssen Sie viele Parameter beim Funktionsbaustein angeben, so bietet es sich an, diese in einer oder mehreren Strukturen zu gruppieren. So können beispielsweise Konfigurationsparameter optisch gut von Kontrollparametern separiert werden.

Siehe auch:

- Objekt Funktionsbaustein [▶ 86]

18.3.4.4 Verwendung von Funktionsbausteinen

Themenpunkte:

1. Funktionsbaustein-Instanzen nicht als VAR PERSISTENT deklarieren [▶ 1163] [++]
2. Zurückgelieferte Fehlerinformationen einer POU auswerten [▶ 1163] [++]
3. Auf lokale Variablen einer POU nicht von außen zugreifen [▶ 1163] [++]
4. Keine direkte Zuweisung von Objekten [▶ 1164] [++]
5. Keine temporären Funktionsbaustein-Instanzen [▶ 1164] [+]

Funktionsbaustein-Instanzen nicht als VAR PERSISTENT deklarieren

Funktionsbaustein-Instanzen nicht als VAR PERSISTENT [▶ 726] deklarieren, da hierdurch die gesamte FB-Instanz in der Datei der persistenten Variablen gespeichert werden würde. Dies könnte z. B. bei Funktionsblöcken mit intern verwendeten ADS-Bausteinen oder Zeigervariablen zu Problemen führen.

Zurückgelieferte Fehlerinformationen einer POU auswerten

Wenn eine Funktion, eine Methode oder ein Funktionsbaustein Fehlerinformationen liefert, werten Sie diese immer aus.

Static Analysis:

Thematisch empfohlene Static Analysis Regeln:

- SA0009: Nicht verwendete Rückgabewerte
- SA0169: Ignorierte Ausgänge

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    fbFileOpen    : FB_FileOpen;      // FileOpen-FB for logger
    bFileOpenExec : BOOL;             // Execute FileOpen-FB
END_VAR
```

```
// NON COMPLIANT: error information of fbFileOpen will not be used
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S);
```

Positives Beispiel:

```
PROGRAM Sample_pos
VAR
    fbFileOpen        : FB_FileOpen;  // FileOpen-FB for logger
    bFileOpenExec     : BOOL;         // Execute FileOpen-FB
    bFileOpenError    : BOOL;         // Error flag of FileOpen-FB
    nFileOpenErrorID  : UDINT;        // Error code of FileOpen-FB
END_VAR
```

```
// COMPLIANT: error information will be handled
fbFileOpen(
    sPathName := 'C:\TestFile.txt',
    nMode     := FOPEN_MODEWRITE OR FOPEN_MODETEXT,
    ePath     := PATH_GENERIC,
    bExecute  := bFileOpenExec,
    tTimeout  := T#3S,
    bError    => bFileOpenError,
    nErrId    => nFileOpenErrorID);

IF bFileOpenError THEN
    F_DoSomethingUsefulHere();        // Handle error here
END_IF
```

Auf lokale Variablen einer POU nicht von außen zugreifen

Auf lokale Variablen einer POU sollte nicht von außerhalb der POU zugegriffen werden.

In TwinCAT 3 sind Schreibzugriffe auf lokale Variablen von außerhalb des Programmierobjekts nicht möglich, da diese vom Compiler als Fehler ausgegeben werden. Lesezugriffe werden vom Compiler hingegen nicht abgefangen.

Um die vorgesehene Datenkapselung zu wahren, wird dringend empfohlen, auf lokale Variablen einer POU nicht von außerhalb der POU weder lesend noch schreibend zuzugreifen. Bei Bibliotheksbausteinen ist zudem nicht sichergestellt, dass die lokalen Variablen eines Funktionsbausteins bei späteren Updates der SPS-Bibliothek unverändert bleiben. Dadurch könnte das Applikationsprojekt nach dem Bibliotheksupdate nicht mehr fehlerfrei übersetzt werden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0102: Zugriff von außen auf lokale Variable

Negatives Beispiel:

```
FUNCTION_BLOCK FB_Sample
VAR
    nLocal          : INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbSample        : FB_Sample;
    nToBeProcessed  : INT;
END_VAR
----------------------------------------------------------------------
nToBeProcessed := fbSample.nLocal;  // NON COMPLIANT: Do not access local variables from external co
ntext.
```

Positives Beispiel:

```
FUNCTION_BLOCK FB_Sample
VAR_OUTPUT
    nOutput         : INT;
END_VAR
VAR
    nLocal          : INT;
END_VAR
```

```
PROGRAM MAIN
VAR
    fbSample        : FB_Sample;
    nToBeProcessed  : INT;
END_VAR
----------------------------------------------------------------------
nToBeProcessed := fbSample.nOutput;  // COMPLIANT: Output variable is accessed from external context
.
```

Keine direkte Zuweisung von Objekten

Einfache Datentypen (INT, BYTE, STRING, …) oder auch Strukturen werden oft direkt zugewiesen.

Instanzen von Funktionsbausteinen sollten Sie nicht mittels Zuweisungsoperator ‚:=' zuweisen. Bei einer solchen Zuweisung erfolgt eine Kopie des Objektes. Funktionsbausteine enthalten jedoch intern häufig Adressen, welche bei einer Zuweisung korrumpiert würden.

Um eine solche Zuweisung grundsätzlich zu verbieten, können Sie in der Definition eines Funktionsbausteins das Attribut 'no_assign' [▶ 848] setzen.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0014: Zuweisungen auf Instanzen

Keine temporären Funktionsbaustein-Instanzen

Funktionsbausteininstanzen sollten Sie nicht auf dem Stack, d.h. nicht als temporäre Variable, deklarieren. Temporäre Instanzen sind solche, die in einer Methode oder einer Funktion oder als VAR_TEMP deklariert sind, und die deshalb in jedem Abarbeitungszyklus bzw. bei jedem Bausteinaufruf neu initialisiert werden.

Funktionsbausteine haben einen Zustand, der meist über mehrere SPS-Zyklen hinweg erhalten bleibt. Eine Instanz auf dem Stack existiert nur für die Dauer des Funktionsaufrufs. Es ist daher nur selten sinnvoll, eine Instanz als temporäre Variable anzulegen. Zweitens sind Funktionsbausteininstanzen häufig groß und verbrauchen sehr viel Platz auf dem Stack, der auf Steuerungen meist begrenzt ist. Drittens kann die Initialisierung und häufig auch die Terminierung eines Funktionsbausteins ziemlich viel Zeit in Anspruch nehmen.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0167: Temporäre Funktionsbaustein-Instanzen

Die Regel SA0167 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Negatives Beispiel:

```
FUNCTION F_Sample
VAR
    fbCtrl : FB_Control;
END_VAR
```

Positives Beispiel:

```
FUNCTION F_Sample
VAR_INPUT
    fbCtrl : REFERENCE TO FB_Control;
END_VAR
```

Siehe auch:

- Objekt Funktionsbaustein [▶ 86]

18.3.5 Variablen

18.3.5.1 Allgemein

Themenpunkte:

1. Fließkommazahlen nicht auf Gleichheit bzw. Ungleichheit testen [▶ 1165] [++]
2. Unerwünschte Ergebnisse mit Hilfe von explizitem Casten vermeiden [▶ 1166] [+]

Fließkommazahlen nicht auf Gleichheit bzw. Ungleichheit testen

Fließkommazahlen sollten Sie nicht direkt oder indirekt auf Gleichheit bzw. Ungleichheit testen. Verwenden Sie stattdessen die Operatoren <, >, <=, >=.

Fließkommazahlen können nicht immer ohne Rundungsfehler in einem Computer dargestellt werden. Gerade bei Berechnungen mit mehreren Fließkommazahlen können schnell kleine Rundungsfehler auftreten. Daher sollte eine Fließkommazahl niemals direkt mit dem „=“-Operator verglichen werden. Ein Maß für die kleinste darstellbare Zahl ist das sogenannte Maschinen-Epsilon (ε). Im Anschluss an die folgenden Beispiele befindet sich eine Funktion, mit der die Genauigkeit einer Maschine numerisch ermittelt werden kann.

Einzige Ausnahme stellt die Prüfung auf gleich (‚=‘) oder ungleich (‚<>‘) Null dar.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0054: Vergleich von REAL/LREAL auf Gleichheit/Ungleichheit

Negatives Beispiel 1:

```
PROGRAM Sample_neg_1
VAR
    fTest          : REAL   := 1E-20;  // Test number
END_VAR
```

```
// The IF-condition is NON COMPLIANT: The comparison is not reliable due rounding errors
IF fTest = 0 THEN
    F_DoSomethingUsefulHere();         // it's just a sample
END_IF
```

Positives Beispiel 1:

```
PROGRAM Sample_pos_1
VAR
    fTest          : REAL   := 1E-20;  // Test number
END_VAR
```

```
VAR CONSTANT
    cFloatEpsilon  : REAL   := 1E-12;  // The deviation epsilon
END_VAR
```

```
// The IF-condition is COMPLIANT: Test with the deviation (± epsilon) around 0
IF (fTest > (0 - cFloatEpsilon)) AND (fTest < (0 + cFloatEpsilon)) THEN
    F_DoSomethingUsefulHere();         // it's just a sample
END_IF
```

Negatives Beispiel 2:

```
PROGRAM Sample_neg_2
VAR
    fCounter       : REAL;             // Test counter
END_VAR
```

```
// The WHILE-
condition is NON COMPLIANT: The comparison is not reliable due rounding errors. Attention: This loop
 will be repeated endlessly!
WHILE fCounter <> 1 DO
    fCounter := fCounter + 0.001;
END_WHILE
```

Positives Beispiel 2:

```
PROGRAM Sample_pos_2
VAR
    fCounter       : REAL;             // Test counter
END_VAR
```

```
// The WHILE-
condition is COMPLIANT because of comparison operator '<'. The loop ends when fCounter is not smalle
r than 1 anymore.
WHILE fCounter < 1 DO
    fCounter := fCounter + 0.001;
END_WHILE
```

Die folgende Funktion kann verwendet werden, um die Genauigkeit einer Maschine numerisch zu ermitteln.

```
FUNCTION F_CalculateMachineEpsilon : LREAL
VAR
    fOnePlusHalfEpsilon  : LREAL;              // Auxiliary variable to calculate and validate the
machine epsilon
    fResult              : LREAL := 1.0;       // Temporary variable to calculate the result
END_VAR
```

```
REPEAT
    fResult   := 0.5 * fResult;                // Devide fResult by 2.
    fOnePlusHalfEpsilon := 1.0 + 0.5 * fResult; // If new result devided by 2 plus 1.0 is smaller th
an or equal to 1.0, the Epsilon is calculated.
UNTIL fOnePlusHalfEpsilon <= 1.0
END_REPEAT;

F_CalculateMachineEpsilon := fResult;          // Set return value to fResult
```

Unerwünschte Ergebnisse mit Hilfe von explizitem Casten vermeiden

Aufgrund von implizitem oder fehlendem Casten können unerwünschte Ergebnisse entstehen. Zum einen kann dies durch zu späte, implizite Konvertierung des Compilers verursacht werden (siehe Beispiel und SA0130). Zum anderen können unerwünschte Ergebnisse durch zu spätes Casten von Variablen auftreten, deren Operation dann in der plattformabhängigen Registerbreite des Prozessors und nicht in der Größe des Variablentyps ausgeführt wird (siehe SA0066). Daher sollte bei der Programmierung genau auf die (Größe der) Variablentypen geachtet und ggf. eine Konvertierung mit Hilfe von explizitem Casten durchgeführt werden.

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0020: Möglicherweise Zuweisung eines abgeschnittenen Werts an REAL-Variable
- SA0066: Verwendung von Zwischenergebnissen
- SA0130: Implizite erweiternde Konvertierungen

Allgemeine Programmelemente für die folgenden Beispiele:

```
PROGRAM Sample
VAR
    nLINT  : LINT;
    nDINT  : DINT;
END_VAR
```

Negatives Beispiel:

```
// Possibly wrong result due to variable overflow caused by late converting of compiler
nLINT := nDINT * nDINT;
```

Positives Beispiel:

```
// Correct result due to explicit variable cast
nLINT := TO_LINT(nDINT) * TO_LINT(nDINT);
```

18.3.5.2 Variablenkapselung

Themenpunkte:

1. Nicht veränderte Variablen als VAR CONSTANT deklarieren [▶ 1167] [+]
2. Globalere Bezeichner nicht verschatten [▶ 1167] [+]
3. ADS Zugriff einschränken [▶ 1168] [+]

Nicht veränderte Variablen als VAR CONSTANT deklarieren

Variablen, die nicht verändert werden, sollten als VAR CONSTANT deklariert werden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0012: Variablen, die als Konstanten deklariert werden könnten

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    fTest         : REAL := 1E-20;   // Test value
    cFloatEpsilon : REAL := 1E-12;   // NON COMPLIANT: cFloatEpsilon will not be changed, but is not
 declared as VAR CONSTANT
END_VAR
-------------------------------------------------------------------
IF (fTest > (0 - cFloatEpsilon)) AND (fTest < (0 + cFloatEpsilon)) THEN
    F_DoSomethingUsefulHere();       // it's just a sample
END_IF
```

Positives Beispiel:

```
PROGRAM Sample_pos
VAR
    fTest         : REAL := 1E-20;   // Test value
END_VAR
VAR CONSTANT
    cFloatEpsilon : REAL := 1E-12;   // COMPLIANT: cFloatEpsilon will not be changed and is declared
 as VAR CONSTANT
END_VAR
-------------------------------------------------------------------
IF (fTest > (0 - cFloatEpsilon)) AND (fTest < (0 + cFloatEpsilon)) THEN
    F_DoSomethingUsefulHere();       // it's just a sample
END_IF
```

Globalere Bezeichner nicht verschatten

Bezeichner in einem inneren Zusammenhang sollten keine globaleren Bezeichner verschatten. Bezeichner in einem inneren Zusammenhang sind z. B. Variablen, die in einer Methode instanziiert werden. Die globaleren Bezeichner sind in diesem Fall beispielsweise Variablen des dazugehörigen Funktionsblocks.

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0013: Deklarationen mit gleichem Variablennamen

- SA0027: Mehrfachverwendung des Namens

Die Regel SA0027 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Allgemeine Programmelemente für die folgenden Beispiele:

```
FUNCTION_BLOCK FB_Sample
VAR
    sDescription : STRING;  // STRING for POU description
END_VAR
-------------------------------------------------------------------
sDescription := 'This is a function block';
```

Negatives Beispiel:

```
METHOD PUBLIC nccl_Test : STRING
VAR
    sDescription : STRING;  // NON COMPLIANT: sDescription is already defined in method's outer scop
e FB_Sample
END_VAR
-------------------------------------------------------------------
sDescription  := 'This is a method';
nccl_Test     := sDescription;
```

Positives Beispiel:

```
METHOD PUBLIC nccl_Test : STRING
VAR
    sMethodDescription  : STRING;  // COMPLIANT
END_VAR
-------------------------------------------------------------------
sMethodDescription  := 'This is a method';
nccl_Test           := sMethodDescription;
```

ADS Zugriff einschränken

Bei Bedarf kann der Zugriff auf Variablen so eingeschränkt werden, dass keine Sichtbarkeit über ADS gegeben ist. Solche Variablen sind dann nicht für HMIs nutzbar. Hierzu kann das Attribut ‚TcNoSymbol' verwendet werden.

18.3.5.3 Arrays

Themenpunkte:

1. Array-Grenzen über Konstanten definieren [▶ 1168] [++]
2. Array-Untergrenze von 1 [▶ 1169] [+]

Array-Grenzen über Konstanten definieren

Die Grenzen eines Arrays sollten Sie mit Hilfe von konstanten Variablen definieren. Wenn innerhalb einer Schleife auf das Array zugegriffen wird, sollten die gleichen konstanten Variablen zur Definition der Schleifengrenzen verwendet werden.

- Untergrenze des Arrays = Untergrenze der Schleife = konstante Variable 1
- Obergrenze des Arrays = Obergrenze der Schleife = konstante Variable 2

Beachten Sie auch den folgenden Themenpunkt der Programmierkonventionen:

- Grenzen einer Schleife als Konstante deklarieren [▶ 1142].

Allgemeine Programmelemente für die folgenden Beispiele:

```
TYPE ST_Object :
STRUCT
    sName     : STRING;
    nID       : UINT;
END_STRUCT
END_TYPE
```

Negatives Beispiel:

```
VAR
    aObjects  : ARRAY[1..10] OF ST_Object;
END_VAR
```

Positives Beispiel:

```
VAR CONSTANT
    cMin      : UINT := 1;
    cMax      : UINT := 10;
END_VAR
VAR
    aObjects  : ARRAY[cMin..cMax] OF ST_Object;
END_VAR
```

Array-Untergrenze von 1

Bei Arrays in den Sprachen der IEC 61131 können sowohl die Ober- als auch die Untergrenze explizit definiert werden. Als Untergrenze sollte eine konstante Variable mit dem Wert 1 verwendet werden, sodass sich als Obergrenze die Anzahl an Elementen im Array ergibt. Die Obergrenze sollte ebenfalls über eine konstante Variable definiert werden.

Dies steht der in Hochsprachen üblicherweise implizit vorgegebenen Untergrenze von 0 entgegen. Es wird dennoch empfohlen, eine Untergrenze von 1 zu verwenden, um einen einheitlichen und einfachen Umgang mit Arrays zu ermöglichen.

Abhängig vom konkreten Anwendungsfall können auch andere Grenzwerte sinnvoll sein. Innerhalb eines Bausteins sollte die Verwendungsweise jedoch unbedingt einheitlich sein.

Positives Beispiel:

```
VAR CONSTANT
    cMin          : UDINT := 1;
    cMax          : UDINT := 10;
    cMaxObjects   : UDINT := 10;
END_VAR
VAR
    nIndex        : UDINT;
    aSample       : ARRAY[cMin..cMax] OF REAL;
    aObjects      : ARRAY[cMin..cMaxObjects] OF ST_Object;
END_VAR
```

```
FOR nIndex := cMin TO cMax DO
    aSample[nIndex] := 123.456;
END_FOR
```

18.3.5.4 Pointer, Referenzen, Interfaces

Themenpunkte:

1. Temporäre Existenz von Pointern/Referenzen/Interfaces auf temporär existierende Objekte [▶ 1169] [++]
2. Pointer/Referenzen jeden Zyklus neu setzen [▶ 1170] [++]
3. Pointer/Referenzen/Interfaces vor jeder Verwendung prüfen [▶ 1170] [++]
4. Referenzen gegenüber Pointern vorziehen [▶ 1171] [++]

Temporäre Existenz von Pointern/Referenzen/Interfaces auf temporär existierende Objekte

Pointer, Referenzen oder Interfaces, die auf temporär existierende Objekte verweisen, sollten nur so lange wie die Objekte selbst existieren.
Objekte existieren beispielsweise temporär, wenn sie in Methoden [▶ 93] oder Funktionen [▶ 83] oder als VAR_TEMP [▶ 723] instanziiert werden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0021: Weitergabe der Adresse einer temporären Variablen

Negatives Beispiel:

```
FUNCTION_BLOCK FB_Sample_neg
VAR
    pPointerToSample : POINTER TO ST_Sample;  // Sample pointer to a structure
END_VAR
```

```
METHOD PUBLIC nccl_TestMethod
VAR
    stSample         : ST_Sample;
END_VAR
----------------------------------------------------------------------
pPointerToSample := ADR(stSample);
// NON COMPLIANT: stSample is declared in an inner scope. So pPointerToSample points to an invalid o
bject outside of the method.
```

Positives Beispiel:

```
FUNCTION_BLOCK FB_Sample_pos
VAR
END_VAR
```

```
METHOD PUBLIC nccl_TestMethod
VAR
    stSample         : ST_Sample;
    pPointerToSample : POINTER TO ST_Sample;  // COMPLIANT
END_VAR
----------------------------------------------------------------------
pPointerToSample := ADR(stSample);
```

Pointer/Referenzen jeden Zyklus neu setzen

Pointer und Referenzen sollten Sie jeden Zyklus neu setzen.

Durch einen Online-Change können sich die Adressen von statisch deklarierten Variablen und Objekten verändern. Daher sollten Pointer und Referenzen, die auf solche Variablen zeigen, jeden Zyklus neu gesetzt werden. Hierzu kann die Adresse der Variablen mittels ADR()-Operator [▶ 736] neu abgefragt werden.

Pointer auf dynamisch erzeugte Objekte (__NEW [▶ 768]) dürfen wiederum nicht jeden Zyklus neu gesetzt werden, sondern müssen so lange bestehen bleiben, bis das Objekt explizit freigegeben wird (__DELETE [▶ 771]).

Werden Pointer/Referenzen am Eingang einer Methode verlangt, so bewirkt die zwingende Zuweisung aller Methoden-Eingangsparameter, dass die Pointer/Referenzen vom Aufrufer neu gesetzt werden.

Negatives Beispiel:

```
FUNCTION_BLOCK FB_Sample_neg
VAR
    aBuffer : ARRAY[cMin..cMax] OF BYTE;        // Sample buffer
    pBuffer : POINTER TO BYTE := ADR(aBuffer);  // Sample pointer to buffer
// NON COMPLIANT: the memory address of aBuffer could change during an online change. So pBuffer cou
ld become invalid.
END_VAR
```

Positives Beispiel:

```
FUNCTION_BLOCK FB_Sample_pos
VAR
    aBuffer : ARRAY[cMin..cMax] OF BYTE;        // Sample buffer
    pBuffer : POINTER TO BYTE;                  // Sample pointer to buffer
END_VAR
-------------------------------------------------
pBuffer := ADR(aBuffer);    // COMPLIANT: pointer is updated before usage in implementation
```

Pointer/Referenzen/Interfaces vor jeder Verwendung prüfen

Pointer, Referenzen und Interfaces sollten vor jeder Verwendung auf ihre Gültigkeit überprüft werden. Bei Pointern und Interfaces findet diese Überprüfung mit Hilfe der Abfrage auf „ungleich 0“ (<> 0) statt, bei Referenzen wird der Operator __ISVALIDREF [▶ 772]() verwendet.

Für spezielle Überprüfungen eines Pointers steht für Ausnahmesituationen die Funktion F_CheckMemoryArea der Tc2_System-Bibliothek zur Verfügung, mit deren Hilfe der Speicherbereich abgefragt werden kann, auf den ein Pointer verweist.

Innerhalb der Bedingung, welche die Gültigkeit des Pointers prüft, sollte dieser nicht zugleich verwendet werden. Möchten Sie dies umsetzen, verwenden Sie zwingend den Operator AND_THEN. Siehe Beispiel 2.

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0039: Mögliche Null-Pointer-Dereferenzierung
- SA0046: Mögliche Verwendung nicht initialisierter Schnittstellen
- SA0145: Mögliche Verwendung nicht initialisierter Referenzen

Thematisch empfohlene Static Analysis Regeln:

- SA0124: Dereferenzierungszugriff in Initialisierungen
- SA0125: Referenzen in Initialisierungen

Allgemeine Programmelemente für die folgenden Beispiele:

Funktionsbaustein FB_Sample implementiert das Interface I_Sample:

```
FUNCTION_BLOCK FB_Sample IMPLEMENTS I_Sample
```

Programm Sample:

```
PROGRAM Sample
VAR
    pSample   : POINTER TO FB_Sample;
    refSample : REFERENCE TO FB_Sample;
    ipSample   : I_Sample;
END_VAR
```

Negatives Beispiel:

```
pSample^.DoSomething();
refSample.DoSomething();
ipSample.DoSomething();
```

Positives Beispiel 1:

```
IF pSample <> 0 THEN
    pSample^.DoSomething();
END_IF

IF __ISVALIDREF(refSample) THEN
    refSample.DoSomething();
END_IF

IF ipSample <> 0 THEN
    ipSample.DoSomething();
END_IF
```

Positives Beispiel 2:

```
IF ipBuffer <> 0 THEN
    IF ipBuffer.bAvailable THEN
        ipBuffer.Clear();
    END_IF
END_IF

IF ipBuffer <> 0 AND_THEN ipBuffer.bAvailable THEN
    ipBuffer.Clear();
END_IF
```

Referenzen gegenüber Pointern vorziehen

Ziehen Sie Referenzen Pointern nach Möglichkeit vor, da Referenzen im Vergleich zu Pointern Typ-sicherer sind.

In speziellen Fällen ist jedoch die Verwendung von Pointern erforderlich, beispielsweise bei der Notwendigkeit von Pointer-Arithmetik. Ebenso werden gerne Puffer beliebiger Größe an eine Funktion übergeben. Hierzu werden ebenfalls Pointer verwendet.

Static Analysis:

Hilfreiche Regeln für eine möglicherweise erforderliche Pointer-Verwendung:

- SA0017: Nicht-reguläre Zuweisungen
- SA0019: Implizite Pointer-Konvertierungen
- SA0061: Unübliche Operation auf Pointer
- SA0064: Addition eines Pointers
- SA0065: Pointer-Addition passt nicht zur Basisgröße

Siehe auch:

- Zeiger / POINTER [▶ 804]
- REFERENCE [▶ 807]
- Objekt Schnittstelle [▶ 107]

18.3.5.5 Allokierte Variablen

Themenpunkte:

1. Keine direkte Adressierung verwenden [▶ 1172] [++]
2. Mehrfache Schreibzugriffe auf Ausgänge vermeiden [▶ 1172] [++]
3. Überlappende Speicherbereiche von Adressvariablen vermeiden [▶ 1173] [+]

Keine direkte Adressierung verwenden

Für allokierte Variablen sollte keine direkte Adressierung verwendet werden. Statt einer direkten Adressierung wird die Verwendung der automatischen Adressierung mit Hilfe des Platzhalters * empfohlen. Mit dem Platzhalter * (%I*, %Q* bzw. %M*) wird eine flexible und optimierte Adressierung von TwinCAT automatisch durchgeführt.

Im Implementierungsteil sollten ebenfalls keine direkten Adresszugriffe stattfinden.

Static Analysis:

Überprüfen mit Hilfe von folgenden Static Analysis Regeln:

- SA0047: Zugriff auf direkte Adressen
- SA0048: AT-Deklarationen auf direkte Adressen

Thematisch empfohlene Static Analysis Regel:

- SA0005: Ungültige Adressen und Datentypen

Negatives Beispiel:

```
VAR
    bInputSignal  AT%IX4.0 : BOOL;
    bVar                   : BOOL;
END_VAR
```

```
bVar := %IX0.0;
```

Positives Beispiel:

```
VAR
    bInputSignal  AT%I*    : BOOL;
END_VAR
```

Mehrfache Schreibzugriffe auf Ausgänge vermeiden

Mehrfache Schreibzugriffe auf Ausgänge sollten vermieden werden. Ein mehrfacher Schreibzugriff auf Ausgänge tritt auf, wenn Ausgänge an mehr als einer Stelle im Programm geschrieben werden.

Ausnahme dieser Regel ist, wenn die Zuweisungen in verschiedenen Zweigen einer IF- bzw. CASE-Anweisung durchgeführt werden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0004: Mehrfacher Schreibzugriff auf Ausgang

Die Regel SA0004 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    bErrorLed      AT%Q*  : BOOL;             // Machine's error LED
    bErrorDrill           : BOOL;             // Drill error status
    bErrorTransport       : BOOL;             // Transport error status
    nTestCounter          : WORD;             // Simple counter
END_VAR
-------------------------------------------------------------------
bErrorLed := bErrorDrill OR bErrorTransport;  // NON COMPLIANT: bErrorLed will be written more than
once: See action CounterInc
CounterInc();
-------------------------------------------------------------------
-------------------------------------------------------------------
(* --- Method: CounterInc ---
   This method increments the counter and checks if nCounterValue is equal to 0 *)
bErrorLed   := (nTestCounter = 0);            // NON COMPLIANT: bErrorLed will be written more than
once: See program MAIN
nTestCounter := nTestCounter + 1;
```

Positives Beispiel:

```
PROGRAM Sample_pos
VAR
    bErrorLed      AT%Q* : BOOL;     // Machine's error LED
    bErrorDrill          : BOOL;     // Drill error status
    bErrorTransport      : BOOL;     // Transport error status
    bErrorCounter        : BOOL;     // Test counter error status
    nTestCounter         : WORD;     // Simple counter
END_VAR
-------------------------------------------------------------------
CounterInc();
    bErrorLed := bErrorDrill         // COMPLIANT
    OR bErrorTransport
    OR bErrorCounter;
-------------------------------------------------------------------
-------------------------------------------------------------------
(* --- Method: CounterInc ---
This method increments the counter and checks if nCounterValue is equal to 0 *)
bErrorCounter  := (nTestCounter = 0); // COMPLIANT
nTestCounter   := nTestCounter + 1;
```

Überlappende Speicherbereiche von Adressvariablen vermeiden

Überlappende Speicherbereiche von Adressvariablen sollten vermieden werden. Überlappende Speicherbereiche treten auf, wenn derselbe Speicherplatz von mehreren Variablen belegt wird.

Wenn ein überlappender Speicherbereich programmtechnisch benötigt wird, können hierfür UNION [▶ 820]s als gezieltes Stilmittel genutzt werden. Überlappende Speicherbereiche von anderen Variablen, z. B. von Adressvariablen, sollten hingegen vermieden werden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0028: Überlappende Speicherbereiche

Die Regel SA0028 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Negatives Beispiel:

```
PROGRAM Sample_neg
VAR
    nTestWord       AT%MW2   : WORD;  // Test WORD
```

```
    nTestLowByte    AT%MB4  : BYTE;  // NON COMPLIANT: overlaps with nTestWord
    nTestHighByte   AT%MB5  : BYTE;  // NON COMPLIANT: overlaps with nTestWord
END_VAR
-----------------------------------------------------------------------
nTestLowByte   := 0;                // NON COMPLIANT: writes nTestWord = 16#xx00
nTestHighByte  := 0;                // NON COMPLIANT: writes nTestWord = 16#00xx
nTestWord      := 16#C0FF;          // Writes nTestLowByte and nTestHighByte too
```

Positives Beispiel:

```
// Structure for the positive sample
TYPE ST_2_BYTES :
STRUCT
    nLow        : BYTE;        // Low byte
    nHigh       : BYTE;        // High byte
END_STRUCT
END_TYPE
```

```
// Union for the positive sample
TYPE U_BYTE_WORD :
UNION
    stLowHigh   : ST_2_BYTES;  // Struct with two bytes
    nValue      : WORD;        // Test WORD to be united with the two-byte-struct
END_UNION
END_TYPE
```

```
PROGRAM Sample_pos
VAR
    uTestWordToBytes   AT%MW2  : U_BYTE_WORD;               // Test union
    nTestLowByte               : BYTE;                      // COMPLIANT: not addressed
    nTestHighByte              : BYTE;                      // COMPLIANT: not addressed
END_VAR
-----------------------------------------------------------------------
uTestWordToBytes.nValue := 16#C0FF;                         // A test value
nTestLowByte            := uTestWordToBytes.stLowHigh.nLow;  // Will be 16#FF
nTestHighByte           := uTestWordToBytes.stLowHigh.nHigh; // Will be 16#C0
```

Siehe auch:

- AT-Deklaration [▶ 71]
- Adressen [▶ 790]

18.3.5.6 Globale Variablen

Themenpunkte:

1. Attribut 'qualified_only' bei GVL verwenden [▶ 1174] [+]
2. Globale Variablen mit Bedacht verwenden [▶ 1174] [+]

Attribut 'qualified_only' bei GVL verwenden

Bei der Definition einer globalen Variablenliste [▶ 75] oder einer Parameterliste [▶ 75] das Attribut {attribute 'qualified_only'} [▶ 859] verwenden, womit bei Verwendung der Variablen die Nutzung des Namensraums der GVL erzwungen wird. Durch die Nutzung des Namensraums (Bsp.: `GVL_Ctrl.bPaintingActive`) wird der globale Scope der Variablen deutlich.

Positives Beispiel:

Globale Variablenliste "GVL_Ctrl":

```
{attribute 'qualified_only'}
VAR_GLOBAL
    bPaintingActive  : BOOL;
END_VAR
```

Globale Variablen mit Bedacht verwenden

Um konkurrierenden Zugriffen vorzubeugen und die Datenkapselung zu unterstützen, möglichst auf globale Instanziierungen verzichten.

Ebenso sollten Sie innerhalb von Funktionsbausteinen möglichst auf die Verwendung von vorhandenen globalen Variablen verzichten. Notwendige Daten weisen Sie über Eingangsparameter zu.

Siehe auch:

- Multiple Tasks [▶ 1178]

18.3.5.7 Strings

Themenpunkte:

1. Bezeichner für Größen- und Längenangaben [▶ 1175] [++]
2. Empfohlene Default-Größe [▶ 1175] [+]
3. Übergabe von großen Strings [▶ 1176] [+]
4. Verarbeitung von großen Strings [▶ 1176] [+]

Bezeichner für Größen- und Längenangaben

Die Größe einer String-Variablen in Bytes als „Size“ bezeichnen.

Die aktuelle Länge eines Strings in Zeichen als „Len“ bezeichnen.

Positives Beispiel:

```
VAR
    sText     : T_MaxString;
    nTextSize : UDINT;
    nTextLen  : UDINT;
END_VAR
-----------------------------------------
nTextSize := SIZEOF(sText);
nTextLen  := LEN(sText);
```

Empfohlene Default-Größe

Die empfohlene Default-Größe einer String-Variablen ist T_MaxString (Alias von STRING(255)). Dies gilt sowohl für die lokale Deklaration des Strings als auch für die Deklaration des Parameters in einer Funktion/ Methode.

Falls eine andere Größe für den Anwendungsfall fest vorgegeben ist, sollte diese als Größenbeschränkung verwendet werden (Beispiel: T_AmsNetId).

Die Größe von T_MaxString ist als Maximum für Funktionen wie LEN() und CONCAT() definiert und führt zu akzeptabler Performanz von regulären Parameterzuweisungen als „Call by value“.

Static Analysis:

Thematisch empfohlene Static Analysis Regel:

- SA0026: Möglicherweise Abschneiden von Strings

Positives Beispiel:

```
FUNCTION F_Sample : BOOL
VAR_INPUT
    sParam1   : T_MaxString;
    sParam2   : T_AmsNetId;
END_VAR
```

```
PROGRAM MAIN
VAR
    sLocal1   : T_MaxString;
    sLocal2   : T_AmsNetId;
    bReturn   : BOOL;
END_VAR
```

```
bReturn := F_Sample(sParam1 := sLocal1,
                    sParam2 := sLocal2);
```

Übergabe von großen Strings

Falls ein größerer String als T_MaxString erforderlich ist, sollte die Übergabe als „Call by reference“ verwendet werden. Dies kann entweder über VAR_IN_OUT CONSTANT [▶ 719] bei ausschließlich lesendem Zugriff oder POINTER TO STRING erreicht werden.

Ein Eingangsparameter als REFERENCE TO STRING(1023) bietet sich nur bedingt an, weil dadurch immer eine fixe Größe der zugewiesenen Variablen verlangt würde. Ist diese Einschränkung hinnehmbar oder gewünscht, bevorzugen Sie diese Variante.

Positives Beispiel:

```
FUNCTION F_Sample : BOOL
VAR_IN_OUT CONSTANT
    sLonger1     : STRING;
END_VAR
VAR_INPUT
    pLonger2     : POINTER TO STRING;
    nLonger2Size : UDINT;
END_VAR
```

```
PROGRAM MAIN
VAR
    sLocal       : STRING(1023);
    bReturn      : BOOL;
END_VAR
```

```
bReturn := F_Sample(sLonger1     := sLocal,
                    pLonger2     := ADR(sLocal),
                    nLonger2Size := SIZEOF(sLocal));
```

Verarbeitung von großen Strings

Bei Verwendung von T_MaxString oder kleineren String-Variablen können Sie String-Funktionen wie CONCAT() oder LEN() verwenden (Tc2_Standard).

Bei größeren String-Variablen müssen Sie Funktionen wie CONCAT2() oder LEN2() nutzen (Tc2_Utilities).

18.3.6 Laufzeitverhalten

18.3.6.1 Allgemein

Themenpunkte:

1. Division durch Null abfangen. [▶ 1176] [++]

Division durch Null abfangen

Um Laufzeitfehlern vorzubeugen, sollte vor einer Division immer eine Abfrage auf ungleich Null stattfinden.

Static Analysis:

Überprüfen mit Hilfe von Static Analysis Regel:

- SA0040: Mögliche Division durch Null

Allgemeine Programmelemente für die folgenden Beispiele:

```
FUNCTION F_Sample
VAR_INPUT
    fDividend : LREAL;
    fDivisor  : LREAL;
END_VAR
VAR
    fResult   : LREAL;
END_VAR
```

Negatives Beispiel:

```
fResult := fDividend / fDivisor;
```

Positives Beispiel:

```
IF fDivisor <> 0 THEN
    fResult := fDividend / fDivisor;
END_IF
```

18.3.6.2 Dynamischer Speicher

Themenpunkte:

1. Dynamische Speicherallokationen mit Bedacht durchführen [▶ 1177] [++]

> **SPS_Bibliothek Tc3_DynamicMemory**
>
> Die Verwendung von dynamischem Speicher kann mit Hilfe der SPS-Bibliothek Tc3_DynamicMemory vereinfacht werden.

Dynamische Speicherallokationen mit Bedacht durchführen

Dynamische Speicherallokationen mittels __NEW [▶ 768]() sollten Sie mit Bedacht durchführen. Grund hierfür ist, dass diese den Speicherbedarf und beispielsweise Array-Grenzen zur Laufzeit verändern. Diese Änderungen sollten Sie stets berücksichtigen, um unerlaubte Speicherzugriffe und damit einen Programmabsturz zu verhindern.

Achten Sie bei dynamischen Speicherallokationen insbesondere darauf, welcher Speicher allokiert wird und wie oft dies geschieht. Da dynamische Speicherallokationen die Laufzeit und ggf. den Speicher beeinträchtigen können, sollten sie nur einmalig zu bestimmten Zeitpunkten durchgeführt werden, wie z. B. beim Hochfahren oder nach dem Umrüsten einer Anlage.

Beachten Sie außerdem, dass dynamisch allokierter Speicher mittels __DELETE [▶ 771]() wieder freigegeben werden muss.

Negatives Beispiel:

```
FUNCTION_BLOCK FB_Sample_neg
VAR
    pDynamicLRealArray : POINTER TO LREAL;  // Pointer to dynamic array
    nArrayCounter      : INT;               // Counter variable
    bInit              : BOOL;              // Initialize array only once
END_VAR
```

```
(* NON COMPLIANT:
1: If the amount of new LREALs is constant (in this case 10 elements), use a static ARRAY [0..10] OF
 LREAL.
2: If the amount of new LREALs is dynamic, do not use magic numbers. If the amount of elements (10)
is changed here, there would be no way to ensure that this number is also changed in any other code
parts.
3: If an array is created dynamically and assigned to a pointer, do not use this original pointer to
 work on the array. The start address of the array would be overwritten. *)
IF NOT bInit THEN
    pDynamicLRealArray := __NEW(LREAL, 10);                  // NON COMPLIANT: see above (1, 2)
    FillDynamicArray();
    bInit              := TRUE;
END_IF
```

Methode FillDynamicArray:

```
FOR nArrayCounter := 1 TO 10 DO                               // NON COMPLIANT: see above (2)
    pDynamicLRealArray^ := 1.25;
    pDynamicLRealArray  := pDynamicLRealArray + SIZEOF(LREAL); // NON COMPLIANT: see above (3)
END_FOR
```

Positives Beispiel:

```
FUNCTION_BLOCK FB_Sample_pos
VAR
    pDynamicLRealArray  : POINTER TO LREAL; // Pointer to dynamic array
    nDynamicArrayLen    : INT;              // Dynamic array length
    nArrayCounter       : INT;              // Counter variable
    pDynamicLRealTemp   : POINTER TO LREAL; // Temp. pointer to dynamic array
    bInit               : BOOL;             // Initialize array only once
END_VAR
```

```
IF NOT bInit THEN
    nDynamicArrayLen    := F_CalculateBufferLen();
    pDynamicLRealArray  := __NEW(LREAL, nDynamicArrayLen);     // COMPLIANT
```

```
    FillDynamicArray();
    bInit               := TRUE;
END_IF
```

Methode FillDynamicArray:

```
// Do not work on the array with the pointer that points to the start address of the dynamic array.
Use a temporary pointer for working on the array to keep the start address of the array.
pDynamicLRealTemp := pDynamicLRealArray;

// The following FOR-statement is COMPLIANT if nDynamicArrayLen is the length of array in any case.
FOR nArrayCounter := 1 TO nDynamicArrayLen DO
    pDynamicLRealTemp^ := 1.25;
    pDynamicLRealTemp  := pDynamicLRealTemp + SIZEOF(LREAL);   // COMPLIANT: Temporary pointer used
to work on the array
END_FOR
```

Alternative Methode FillDynamicArray:

```
// The following FOR-statement is COMPLIANT if nDynamicArrayLen is the length of array in any case.
FOR nArrayCounter := 1 TO nDynamicArrayLen DO
    pDynamicLRealArray[nArrayCounter-1] := 1.25;               // COMPLIANT: Pointer is not changed
END_FOR
```

18.3.6.3 Multiple Tasks

Themenpunkte:

1. Konkurrierende Zugriffe auf Speicherbereiche vermeiden [▶ 1178] [++]
2. Konkurrierende Zugriffe auf Funktionsbausteine vermeiden [▶ 1179] [++]
3. Globale Variablen mit Bedacht verwenden [▶ 1180] [+]

Details zur Absicherung von konkurrierendem Zugriff sind im Kapitel Multitask-Datenzugriffs-Synchronisation in der SPS [▶ 382] beschrieben.

Konkurrierende Zugriffe auf Speicherbereiche vermeiden

Konkurrierende Zugriffe auf Speicherbereiche sollten Sie vermeiden. Ein konkurrierender Zugriff auf Speicherbereiche tritt beispielsweise auf, wenn Variablen von mehr als einem Task gelesen und/oder geschrieben werden, wobei mindestens ein schreibender Zugriff vorliegt.

⚠ WARNUNG

Inkonsistente Variablenwerte

Wird aus mehreren Taskkontexten beispielsweise auf eine globale Variable zugegriffen und erfolgt mindestens ein Zugriff auch schreibend, kann der Variablenwert mit großer Wahrscheinlichkeit inkonsistent werden. Ein inkonsistenter Variablenwert hat meistens schwerwiegende Folgen auf den Programmablauf und kann auch einen Stopp der SPS bewirken.

Static Analysis:

Überprüfen mit Hilfe der folgenden Static Analysis Regeln:

- SA0006: Schreibzugriff aus mehreren Tasks
- SA0103: Gleichzeitiger Zugriff auf nicht-atomare Daten

Die Regel SA0006 ist auch in der lizenzfreien Variante Static Analysis Light [▶ 158] enthalten.

Negatives Beispiel:

GVL_Sample:

```
VAR_GLOBAL
    nTestOutput    AT%Q*  : WORD;  // Test output to be allocated
    nGlobalCounter        : UDINT;
    nLastErrorID          : UDINT;
END_VAR
```

Programm Sample_neg_task_M, ausgeführt in Task M:

```
PROGRAM Sample_neg_task_M
VAR
    nLocalTaskM  : WORD;
    fbLocalTaskM : FB_Test;
END_VAR
```

```
GVL_Sample.nTestOutput     := nLocalTaskM;                    // NON COMPLIANT: Task N reads variable
 being written by Task M whereas the accesses are not synchronized.
GVL_Sample.nGlobalCounter  := GVL_Sample.nGlobalCounter + 1; // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nGlobalCounter
GVL_Sample.nLastErrorID    := fbLocalTaskM.nErrorID;          // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nLastErrorID
```

Programm Sample_neg_task_N, ausgeführt in Task N:

```
PROGRAM Sample_neg_task_N
VAR
    nLocalTaskN  : DWORD;
    fbLocalTaskN : FB_Test;
END_VAR
```

```
nLocalTaskN                := GVL_Sample.nTestOutput;         // NON COMPLIANT: Task N reads variable
 being written by Task M whereas the accesses are not synchronized.
GVL_Sample.nGlobalCounter  := 0;                              // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nGlobalCounter
GVL_Sample.nLastErrorID    := fbLocalTaskN.nErrorID;          // NON COMPLIANT: Task M & Task N write
 to GVL_Sample.nLastErrorID
```

Positives Beispiel:

GVL_Sample:

```
VAR_GLOBAL
    nTestOutput    AT%Q* : WORD;  // Test output to be allocated
    nGlobalCounter       : UDINT;
    nLastErrorID_TaskM   : UDINT; // only used in Task M
    nLastErrorID_TaskN   : UDINT; // only used in Task N
END_VAR
```

Programm Sample_pos_task_M, ausgeführt in Task M:

```
PROGRAM Sample_pos_task_M
VAR
    nLocalTaskM  : WORD;
    fbLocalTaskM : FB_Test;
END_VAR
```

```
GVL_Sample.nLastErrorID_TaskM  := fbLocalTaskN.nErrorID; // writes data that is declared for task M
and only used in task M
IF (* special condition to synchronize access - see InfoSys chapter 'multitask data access' *) THEN
    GVL_Sample.nTestOutput     := nLocalTaskM;
    GVL_Sample.nGlobalCounter  := GVL_Sample.nGlobalCounter + 1;
END_IF
```

Programm Sample_pos_task_N, ausgeführt in Task N:

```
PROGRAM Sample_pos_task_N
VAR
    nLocalTaskN  : DWORD;
    fbLocalTaskN : FB_Test;
END_VAR
```

```
GVL_Sample.nLastErrorID_TaskN  := fbLocalTaskN.nErrorID; // writes data that is declared for task N
and only used in task N
IF (* special condition to synchronize access - see InfoSys chapter 'multitask data access' *) THEN
    nLocalTaskN                := GVL_Sample.nTestOutput;
    GVL_Sample.nGlobalCounter  := 0;
END_IF
```

Konkurrierende Zugriffe auf Funktionsbausteine vermeiden

Konkurrierende Zugriffe auf Funktionsbausteine sollten Sie vermeiden.

Ein Zugriff aus mehreren Taskkontexten auf einen Funktionsbaustein ist nicht erlaubt. Das betrifft sowohl den Aufruf des Rumpfes als auch von Methoden oder Properties. Der interne Ablauf kann mit großer Wahrscheinlichkeit zu inkonsistenten Variablenwerten führen. Die Auswirkungen sind wiederum schwerwiegende Folgen auf den Programmablauf oder ein sofortiger Stopp der SPS.

In wenigen Ausnahmen erlaubt die Definition eines Funktionsbausteines einen Zugriff aus mehreren Taskkontexten. Sollte ein Funktionsbaustein für diesen Anwendungsfall konzipiert und entwickelt sein, so ist dies explizit dokumentiert. In allen anderen Fällen ist eine globale Instanziierung und Verwendung aus mehreren Taskkontexten nicht erlaubt.

Globale Variablen mit Bedacht verwenden

Um konkurrierenden Zugriffen vorzubeugen und die Datenkapselung zu unterstützen, möglichst auf globale Instanziierungen verzichten.

Ebenso sollten Sie innerhalb von Funktionsbausteinen möglichst auf die Verwendung von vorhandenen globalen Variablen verzichten. Notwendige Daten weisen Sie über Eingangsparameter zu.

19 Beispiele

Im folgenden Kapitel finden Sie einige Beispiele, die bei der Arbeit mit der TwinCAT 3 PLC hilfreich sein können.

19.1 Basisbeispiele

19.1.1 Erste Schritte – Zustandsmaschine, Timer, Trigger

Beschreibung	Dieses SPS-Beispiel zeigt die grundlegende Syntax sowie einige grundlegende Anweisungen und Funktionalitäten. Das Projekt enthält eine Enumerations-basierte Zustandsmaschine, bei der die Zustandsweiterschaltung über die folgenden Mechanismen gesteuert wird: • Timer zur Zeitüberwachung (TON) • Trigger zur Erkennung einer steigenden Flanke (R_TRIG)
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644037771.zip
Weiterführende Informationen	In der Dokumentation PLC: Ihr erstes TwinCAT-3-SPS-Projekt In der Dokumentation PLC: SPS-Projekt programmieren

19.1.2 Erste Schritte – Basis-SPS-Elemente

Beschreibung	Dieses SPS-Beispiel zeigt die Verwendung einiger grundlegender SPS-Elemente, die zur Strukturierung eines SPS-Projekts verwendet werden können. Die Elemente, die in diesem Projekt verwendet werden, sind: • Programm [▶ 88] • Funktionsbaustein (FB) [▶ 86] • Funktion [▶ 83] • Methode [▶ 93] • Struktur [▶ 814] • Enumeration [▶ 816] • Globale Variablenliste (GVL) [▶ 75]
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643973259.zip
Weiterführende Informationen	In der Dokumentation PLC: SPS-Projekt programmieren

19.1.3 STRING-Funktionen

Beschreibung	Dieses SPS-Beispiel enthält eine Sammlung von Beispielen zu den grundlegenden STRING-Funktionen. Die gezeigten Funktionen sind: • CONCAT • DELETE • FIND • INSERT • LEFT • LEN • MID • REPLACE • RIGHT
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644039435.zip
Weiterführende Informationen	In der Dokumentation Tc2_Standard Bibliothek: STRING-Funktionen

19.1.4 OOP Basis-Sample

Beschreibung	Dieses SPS-Beispiel zeigt einige Basisfunktionalitäten der objektorientierten Programmierung (OOP). Die enthaltenen Elemente/Funktionalitäten sind: • Funktionsbausteine (FB) • Methoden • Properties • FB-Vererbung/-Erweiterung • Interface (ITF) -Implementierung und -Verwendung
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644034443.zip
Weiterführende Informationen	In der Dokumentation PLC: Objektorientiert programmieren

19.2 Erweiterte Beispiele

19.2.1 OOP Extended-Sample

Beschreibung	Dieses SPS-Beispiel enthält ein objektorientiertes Programm zur Steuerung einer Sortieranlage. Die Anwendung kann über eine integrierte Visualisierung gesteuert werden.
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644036107.zip
Weiterführende Informationen	In der Dokumentation PLC: Objektorientiertes Programm zur Steuerung einer Sortieranlage

19.2.2 Byte-Alignment

Beschreibung	Dieses SPS-Beispiel enthält einige Strukturen mit implizitem oder explizitem Byte-Alignment. Abhängig von dem verwendeten Byte-Alignment und der Reihenfolge der Variablen innerhalb der Struktur werden innerhalb oder am Ende der Struktur Füllbytes eingefügt.
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643974923.zip
Weiterführende Informationen	In der Dokumentation PLC: Alignment

19.2.3 Multitask-Datenzugriffs-Synchronisation

Beschreibung	Es gibt verschiedene Möglichkeiten, wie Multitask-Datenzugriffe in der SPS synchronisiert werden können. Die folgenden SPS-Beispiele zeigen die folgenden Optionen auf: • Mutex-Verfahren (TestAndSet, FB_IecCriticalSection) zum Absichern von kritischen Bereichen • Austausch von Daten über synchronisierte Puffer • Austausch von Daten über das SPS-Prozessabbild
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643978251.zip https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644032779.zip https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643976587.zip https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7643979915.zip
Weiterführende Informationen	In der Dokumentation PLC: Multitask-Datenzugriffs-Synchronisation in der SPS

19.2.4 Bibliotheksdokumentation reStructuredText

Beschreibung	Für eine attraktivere Darstellung der Dokumentation von Bibliotheksobjekten im Bibliotheksverwalter kann das Dokumentationsformat reStructuredText verwendet werden. Dieses SPS-Beispiel zeigt die Syntax verschiedener Struktur- und Layout-Elemente der reStructuredText-Dokumentationsmöglichkeiten.
Beispielprojekt	https://infosys.beckhoff.com/content/1031/tc3_plc_intro/Resources/7644044171.zip
Weiterführende Informationen	In der Dokumentation PLC: Erweitert – reStructuredText

Mehr Informationen:
www.beckhoff.com/te1000

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