SIEMENS

SIMOTION

SIMOTION SCOUT
SIMOTION ST Structured Text

Programmier- und Bedienhandbuch

01/2015

Rechtliche Hinweise

Warnhinweiskonzept

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

⚠ GEFAHR
bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ WARNUNG
bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ VORSICHT
bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

ACHTUNG
bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung **qualifiziertem Personal** gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch von Siemens-Produkten

Beachten Sie Folgendes:

⚠ WARNUNG
Siemens-Produkte dürfen nur für die im Katalog und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von Siemens empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

℗ 02/2015 Änderungen vorbehalten

Copyright © Siemens AG 2015.
Alle Rechte vorbehalten

Vorwort

Gültigkeitsbereich

Das vorliegende Dokument ist Bestandteil des Dokumentationspaketes **SIMOTION Programmieren**.

Das Dokument ist gültig für SIMOTION SCOUT, das Engineering System der Produktfamilie SIMOTION, in der Produktstufe V4.4 in Verbindung mit:

- Einem SIMOTION Gerät mit einem SIMOTION Kernel folgender Versionen:
  - V4.4
  - V4.3
  - V4.2
  - V4.1
  - V4.0
  - V3.2
- Folgenden SIMOTION Technologiepaketen in der zum jeweiligen Kernel passenden Version
  - Cam
  - Path (ab Kernel V4.1)
  - Cam_ext
  - TControl

Das Dokument beschreibt die Syntax der Programmiersprache SIMOTION ST Structured Text dieser Version des SIMOTION SCOUT sowie den Umgang mit dieser Programmiersprache. Außerdem sind Informationen zu folgenden Themenbereichen enthalten:

- ST-Editor und Compiler mit Beispielprogramm
- Datenhaltung und Datenverwaltung auf den SIMOTION Geräten
- Diagnosemöglichkeiten und Fehlerbehebung

Der Sprachumfang der Programmiersprache SIMOTION ST kann gegenüber früheren Versionen neue Syntaxelemente enthalten. Diese wurden nur mit der jeweils aktuellen Version des SIMOTION Kernels getestet und sind nur ab dieser Kernelversion freigegeben.

Konvertieren bestehender Projekte in die aktuelle Version des SIMOTION SCOUT

Ein Hochkonvertieren bestehender Projekte auf die aktuelle Version des SIMOTION SCOUT und der Programmiersprache SIMOTION ST ist möglich. Eine Neuübersetzung mit der aktuellen Version des Compilers ändert unter Umständen die Versionskennungen in den Datenbereichen der Programme und bewirkt damit ein Löschen und Initialisieren aller remanenten und nicht remanenten Daten auf dem SIMOTION Gerät. In Ausnahmefällen können auch geringfügige Anpassungen in den Programmquellen erforderlich werden.

Wenn neue Syntaxelemente der Programmiersprache SIMOTION ST auf einem SIMOTION Gerät mit einer älteren Version des SIMOTION Kernels verwendet werden, gibt der Compiler die Warnung 16700 aus. Wenn diese Syntaxelemente dennoch verwendet werden, kann das Projekt zwar im alten Projektformat gespeichert werden, aber nicht mehr mit dem Compiler einer alten Version des SIMOTION SCOUT übersetzt werden.

Informationsblöcke des Handbuchs

Nachstehende Informationsblöcke beschreiben Zweck und Nutzen des Handbuches.

- **Einleitung** (Kapitel 1)
- **Erste Schritte mit ST** (Kapitel 2)
  Voraussetzung zur Programmerstellung und Beispielprogramm
- **ST-Grundlagen** (Kapitel 3)
  Elemente der Programmiersprache ST, Variablen- und Datentypdeklarationen, Anweisungen
- **Funktionen, Funktionsbausteine, Programme** (Kapitel 4)
  Programmieren und Aufruf der Programmorganisationseinheiten (POE)
- **Einbindung von ST in SIMOTION SCOUT** (Kapitel 5)
  Verhalten der Variablen, Zugriff auf Ein- und Ausgänge, Bibliotheken, Präprozessor
- **Fehlerquellen und Programmtest** (Kapitel 6)
  Hinweise zu Fehlerquellen, effizientes Programmieren und Test der Programme
- **Anhänge**
  - **Formale Sprachbeschreibung** (Anhang A.1)
  - **Fehlermeldungen des Compilers und deren Behebung** (Anhang A.2)
  - **Template für Beispiel-Unit** (Anhang A.3)
- **Index**

Falls Sie sofort beginnen möchten, arbeiten Sie Kapitel 2 durch.

SIMOTION Dokumentation

Einen Überblick zur SIMOTION Dokumentation erhalten Sie im Dokument SIMOTION Dokumentations-Übersicht.

Diese Dokumentation ist als elektronische Dokumentation im Lieferumfang von SIMOTION SCOUT enthalten und besteht aus 10 Dokumentationspaketen.

Zur SIMOTION Produktstufe V4.4 stehen folgende Dokumentationspakete zur Verfügung:

- SIMOTION Engineering System Handhabung
- SIMOTION System- und Funktionsbeschreibungen
- SIMOTION Service und Diagnose
- SIMOTION IT
- SIMOTION Programmieren
- SIMOTION Programmieren - Referenzen

- SIMOTION C
- SIMOTION P
- SIMOTION D
- SIMOTION Ergänzende Dokumentation

Hotline und Internetadressen

Weiterführende Informationen

Unter folgendem Link finden Sie Informationen zu den Themen:

- Dokumentation bestellen / Druckschriftenübersicht
- Weiterführende Links für den Download von Dokumenten
- Dokumentation online nutzen (Handbücher/Informationen finden und durchsuchen)

http://www.siemens.com/motioncontrol/docu

My Documentation Manager

Unter folgendem Link finden Sie Informationen, wie Sie Dokumentation auf Basis der Siemens Inhalte individuell zusammenstellen und für die eigene Maschinendokumentation anpassen:

http://www.siemens.com/mdm

Training

Unter folgendem Link finden Sie Informationen zu SITRAIN - dem Training von Siemens für Produkte, Systeme und Lösungen der Automatisierungstechnik:

http://www.siemens.com/sitrain

FAQs

Frequently Asked Questions finden Sie in den SIMOTION Utilities & Applications, die im Lieferumfang von SIMOTION SCOUT enthalten sind, und in den Service&Support-Seiten unter **Produkt Support**:

http://support.automation.siemens.com

Technical Support

Landesspezifische Telefonnummern für technische Beratung finden Sie im Internet unter **Kontakt**:

http://www.siemens.com/automation/service&support

Inhaltsverzeichnis

Grundlegende Sicherheitshinweise 1

1.1 Allgemeine Sicherheitshinweise

⚠ WARNUNG

Lebensgefahr durch Nichtbeachtung von Sicherheitshinweisen und Restrisiken

Durch Nichtbeachtung der Sicherheitshinweise und Restrisiken in der zugehörigen Hardware-Dokumentation können Unfälle mit schweren Verletzungen oder Tod auftreten.

- Halten Sie die Sicherheitshinweise der Hardware-Dokumentation ein.
- Berücksichtigen Sie bei der Risikobeurteilung die Restrisiken.

⚠ WARNUNG

Lebensgefahr durch Fehlfunktionen der Maschine infolge fehlerhafter oder veränderter Parametrierung

Durch fehlerhafte oder veränderte Parametrierung können Fehlfunktionen an Maschinen auftreten, die zu Körperverletzungen oder Tod führen können.

- Schützen Sie die Parametrierungen vor unbefugtem Zugriff.
- Beherrschen Sie mögliche Fehlfunktionen durch geeignete Maßnahmen (z. B. NOT-HALT oder NOT-AUS).

1.2 Industrial Security

Hinweis

Industrial Security

Siemens bietet Produkte und Lösungen mit Industrial Security-Funktionen an, die den sicheren Betrieb von Anlagen, Lösungen, Maschinen, Geräten und/oder Netzwerken unterstützen. Sie sind wichtige Komponenten in einem ganzheitlichen Industrial Security-Konzept. Die Produkte und Lösungen von Siemens werden unter diesem Gesichtspunkt ständig weiterentwickelt. Siemens empfiehlt, sich unbedingt regelmäßig über Produkt-Updates zu informieren.

Für den sicheren Betrieb von Produkten und Lösungen von Siemens ist es erforderlich, geeignete Schutzmaßnahmen (z. B. Zellenschutzkonzept) zu ergreifen und jede Komponente in ein ganzheitliches Industrial Security-Konzept zu integrieren, das dem aktuellen Stand der Technik entspricht. Dabei sind auch eingesetzte Produkte von anderen Herstellern zu berücksichtigen. Weitergehende Informationen über Industrial Security finden Sie unter http://www.siemens.com/industrialsecurity.

Um stets über Produkt-Updates informiert zu sein, melden Sie sich für unseren produktspezifischen Newsletter an. Weitere Informationen hierzu finden Sie unter http://support.automation.siemens.com

⚠ WARNUNG

Gefahr durch unsichere Betriebszustände wegen Manipulation der Software

Manipulationen der Software (z. B. Viren, Trojaner, Malware, Würmer) können unsichere Betriebszustände in Ihrer Anlage verursachen, die zu Tod, schwerer Körperverletzung und zu Sachschäden führen können.

- Halten Sie die Software aktuell.
  Informationen und Newsletter hierzu finden Sie unter:
  http://support.automation.siemens.com
- Integrieren Sie die Automatisierungs- und Antriebskomponenten in ein ganzheitliches Industrial Security-Konzept der Anlage oder Maschine nach dem aktuellen Stand der Technik.
  Weitergehende Informationen finden Sie unter:
  http://www.siemens.com/industrialsecurity
- Berücksichtigen Sie bei Ihrem ganzheitlichen Industrial Security-Konzept alle eingesetzten Produkte.

Einleitung 2

Immer häufiger müssen Automatisierungssysteme neben den klassischen Steuerungs- und Regelungsaufgaben heute auch Datenverwaltungsaufgaben und komplexere mathematische Berechnungen übernehmen. Speziell für diese Aufgaben bieten wir Ihnen ST (Structured Text), die Programmiersprache, die das Programmieren leichter macht – genormt nach IEC 61131-3 (deutsch DIN EN-61131-3).

2.1 Höhere Programmiersprache

ST ist eine höhere Programmiersprache, die sich an PASCAL orientiert. Die Sprache basiert auf der Norm IEC 61131-3. Diese Norm standardisiert die Programmiersprachen für speicherprogrammierbare Steuerungen (SPS). Basis für ST ist der Teil *Strukturierter Text*.

Die Programmierung von Steuerungen mit einer Hochsprache wie ST eröffnet Ihnen vielfältige Möglichkeiten, beispielsweise:

- Datenverwaltung
- Prozessoptimierung
- mathematische/statistische Berechnungen

2.2 Programmiersprache mit Technologiebefehlen

Zusätzlich zur standardisierten Programmiersprache nach IEC 61131-3 verfügt SIMOTION ST über Befehle für die SIMOTION Geräte, Motion Control und Technologie.

Technologieobjekte repräsentieren eine technologische Funktionalität, z. B. Achse positionieren, Nocken parametrieren. Technologiebefehle sind die Sprachbefehle, die von den Technologieobjekten zur Verfügung gestellt werden. Dies sind beispielsweise Befehle für das Aktivieren eines Kurvenscheibengleichlaufs oder zur Steuerung von Bewegungsabläufen, beispielsweise für das Positionieren einer Achse.

2.3 Ablaufebenen

Das SIMOTION Ablaufsystem stellt verschiedene Ablaufebenen (zyklisch, synchron, zeitgesteuert, alarmgesteuert, sequentiell) zur Verfügung, um die unterschiedlichen Aufgaben bei der Programmierung von Anwenderprogrammen bestmöglich zu unterstützen.

SIMOTION SCOUT ist das Engineering System der Produktfamilie SIMOTION. ST ist die Hochsprache zur Programmierung von Anwenderprogrammen, wobei man in ST Anwenderprogramme für die verschiedenen Ablaufebenen entwickeln kann.

Anwenderprogramme können zeitgesteuert ausgeführt werden, wenn sie synchron zum angegebenen Systemtakt oder in einem festen Zeitraster ablaufen sollen. Sie können interruptgesteuert ausgeführt werden, wenn sie beim Eintreten eines Ereignisses gestartet und einmalig ablaufen sollen. Sie können aber auch sequentiell oder zyklisch in der Round-Robin-Ablaufebene ausgeführt werden.

2.4 ST-Editor mit Werkzeugen für Erstellung und Test von Programmen

Zur Programmerstellung steht Ihnen ein komfortabler Texteditor zur Verfügung.

Der ST-Compiler übersetzt das editierte Programm in ausführbaren Code und zeigt jeden Syntaxfehler mit Angabe von Programmzeile und Fehlerursache an.

Zum Testen der ST-Programme stehen Ihnen Testfunktionen im SIMOTION SCOUT zur Verfügung. Sie können Ihre Programme online visualisieren und testen.

Erste Schritte mit ST

3

Dieses Kapitel beschreibt an einem einfachen Beispiel, wie Sie ein Programm erstellen, in lauffähigen Code übersetzen, es ausführen und testen.

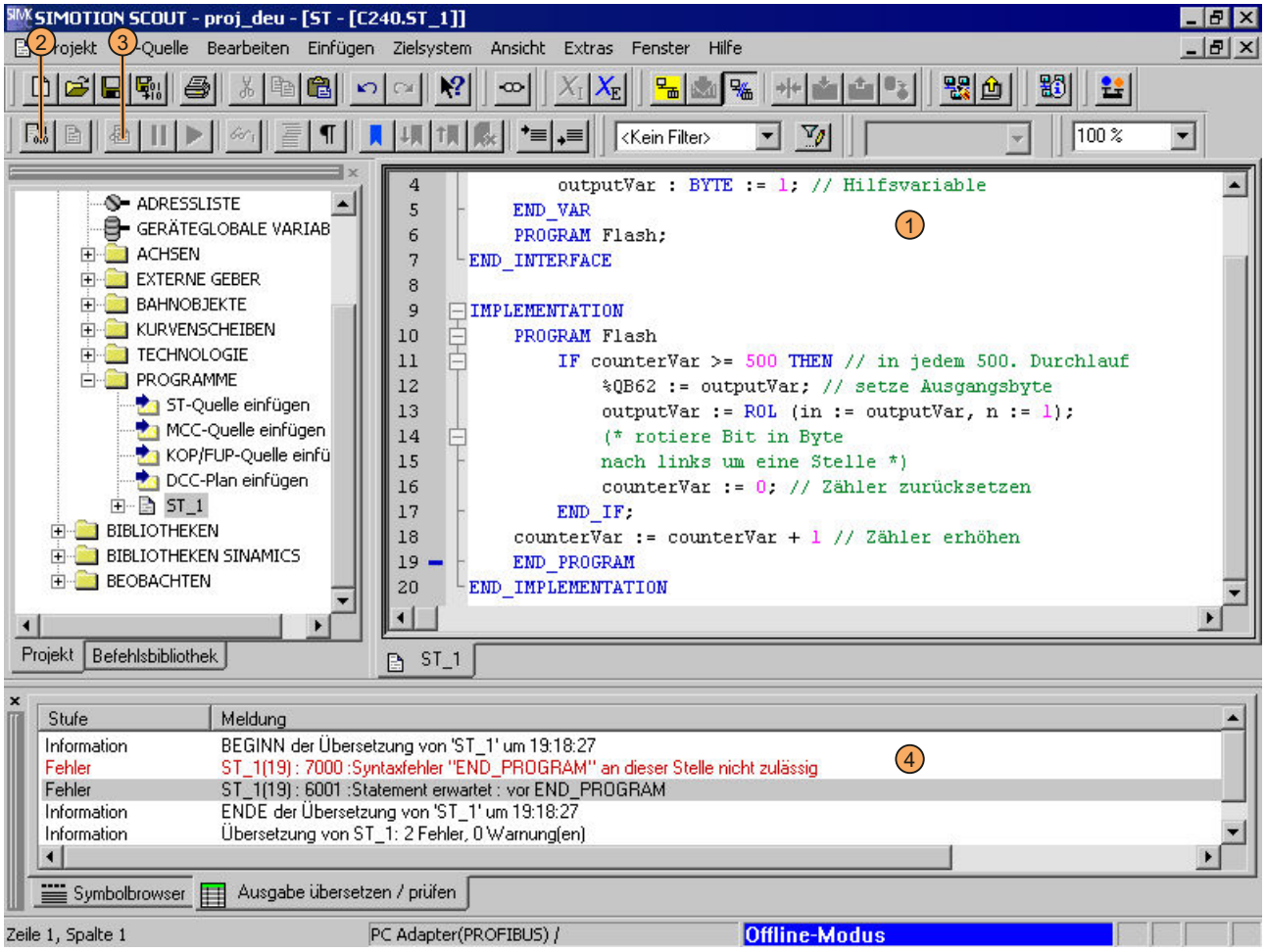
3.1 Einbindung von ST in SCOUT

Die Programmierumgebung zu ST besteht aus folgenden Komponenten:

- einem **Editor**, um Programme, bestehend aus Funktionen (FC), Funktionsbausteinen (FB), anwenderdefinierten Datentypen (UDT) usw., zu programmieren;
- einem **Compiler**, um das zuvor editierte ST-Programm in einen ausführbaren Maschinencode zu übersetzen;
- mehreren Diagnosefunktionen (z. B. **Status Programm**) zur Unterstützung bei der Suche nach logischen Programmfehlern im laufenden Programm;
- einer **Detailanzeige**, in der z. B. Fehlermeldungen des Compilers angezeigt werden. Ein wichtiges Register der Detailanzeige ist der **Symbol-Browser**; dort können Sie Variablen beobachten und ändern.

Die einzelnen Komponenten sind einfach und komfortabel zu handhaben. Sie sind direkt in die Workbench von SIMOTION SCOUT eingebunden.

Näheres zur Bedienung der Workbench und deren Werkzeuge siehe Projektierungshandbuch SIMOTION SCOUT.

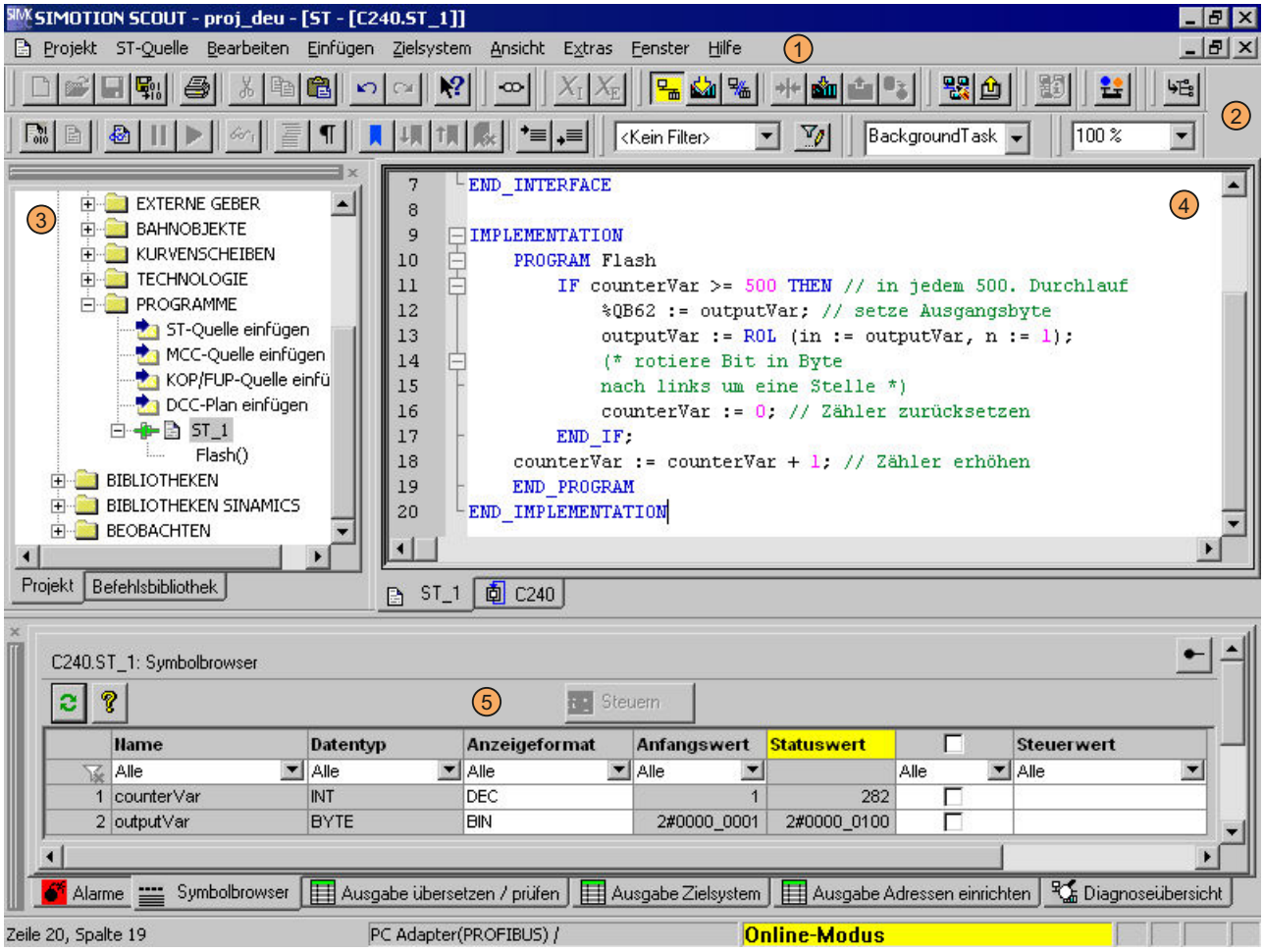


① Editor
② Compiler
③ Status Programm
④ Detailanzeige (Register Ausgabe übersetzen / prüfen)

Bild 3-1 Entwicklungsumgebung von ST

3.1.1 Elemente der Workbench kennenlernen

Die Workbench stellt den Rahmen für den SIMOTION SCOUT dar. Mit deren Werkzeugen können Sie alle nötigen Schritte durchführen, um eine Maschine so zu konfigurieren, optimieren und programmieren, damit sie die gewünschte Aufgabe ausführen kann.



1) Menüleiste
2) Funktionsleisten
3) Projektnavigator
4) Arbeitsbereich
5) Detailanzeige (Register Symbolbrowser)

Bild 3-2 Elemente der Workbench

Die Workbench enthält folgende Elemente:

- Menüs
  Menüs enthalten Menübefehle, mit denen Sie die Workbench steuern, Werkzeuge aufrufen können etc.
- Funktionsleisten
  Sie können viele der vorhandenen Menübefehle mit Klick auf das entsprechende Symbol in einer der Funktionsleisten ausführen.
- Projektnavigator
  Im Projektnavigator sehen Sie das gesamte Projekt und dessen Elemente (z. B. CPU, Achsen, Programme, Kurvenscheiben) in einer Baumstruktur.
- Arbeitsbereich
  In diesem Fenster führen Sie aufgabenspezifische Arbeiten durch, sei es selbständig (programmieren) oder mit Hilfe von Assistenten (konfigurieren).
- Detailanzeige
  Zu den im Projektnavigator ausgewählten Elementen sehen Sie nähere Informationen, z. B. alle globalen Variablen zu einem Programm oder das Fenster **Ausgabe übersetzen/ prüfen**.

3.2 Voraussetzungen für die Programmerstellung

Dieses Kapitel umreißt die allgemeinen Voraussetzungen, die für die Programmerstellung nötig sind. Ausführliche Informationen finden Sie im Projektierungshandbuch SIMOTION SCOUT und den Funktionsbeschreibungen SIMOTION Motion Control.

Projekt einfügen oder öffnen

Das Projekt ist die oberste Hierarchiestufe der Datenverwaltung. Alle Daten, die z. B. zu einer Produktionsmaschine gehören, legt SIMOTION SCOUT in dem zum Projekt gehörenden Verzeichnis ab.

Das Projekt bildet somit die Klammer für alle SIMOTION Geräte, Antriebe usw., die zu einer Maschine gehören.

Erst nach Anlegen eines Projekts können Sie:

- Hardware konfigurieren
- Technologieobjekte einfügen und konfigurieren

Hardware konfigurieren

Innerhalb des Projekts muss die verwendete Hardware dem System bekannt gemacht werden, u. a.:

- SIMOTION Gerät
- zentrale Peripherie (mit I/O-Adressen)
- dezentrale Peripherie (mit I/O-Adressen)

Erst nach Konfiguration eines SIMOTION Geräts können Sie ST-Quellen einfügen und bearbeiten.

Technologieobjekte einfügen und konfigurieren

Die Funktionalität von Achsen, Nocken usw. wird in SIMOTION durch Technologieobjekte (TO) repräsentiert.

Erst nach Einfügen und Konfigurieren der Technologieobjekte können Sie diese mittels Systemfunktionen programmieren und auf ihre Systemvariablen zugreifen.

3.3 Mit dem ST-Editor und dem Compiler umgehen

In diesem Kapitel erfahren Sie, wie Sie den ST-Editor und den Compiler benutzen.

3.3.1 ST-Quelle einfügen

ST-Quellen sind dem SIMOTION Gerät zugeordnet, auf dem die in der Quelle enthaltenen Programme später ablaufen sollen (z. B. SIMOTION C240).

Sie werden im Projektnavigator unter dem SIMOTION Gerät im Ordner PROGRAMME abgelegt.

Hinweis

Im Ordner **PROGRAMME** unter dem SIMOTION Gerät werden auch MCC-Quellen, KOP/FUP-Quellen und DCC-Pläne abgelegt.

Eine Beschreibung der Programmiersprache SIMOTION MCC (Motion Control Chart) finden Sie im Programmier- und Bedienhandbuch SIMOTION MCC.

Eine Beschreibung der Programmiersprachen SIMOTION KOP und SIMOTION FUP finden Sie im Programmier- und Bedienhandbuch SIMOTION KOP/FUP.

Vorgehensweise

1. Öffnen Sie im Projektnavigator das entsprechende SIMOTION Gerät.
2. Markieren Sie den Ordner **PROGRAMME**.
3. Wählen Sie das Menü **Einfügen > Programm ST-Quelle**.
4. Geben Sie den Namen der ST-Quelle ein.
   Namen für Programmquellen müssen den Regeln für Bezeichner (Seite 100) genügen: Sie sind aus Buchstaben (A … Z, a … z), Ziffern (0 … 9) oder einzelnen Unterstrichen (_) in beliebiger Reihenfolge zusammengesetzt, wobei das erste Zeichen ein Buchstabe oder ein Unterstrich sein muss. Zwischen Groß- und Kleinbuchstaben wird nicht unterschieden. Die zulässige Länge der Namen ist abhängig von der Version des SIMOTION Kernels:
   - Ab Version V4.1 des SIMOTION Kernels: maximal 128 Zeichen.
   - Bis Version V4.0 des SIMOTION Kernels: maximal 8 Zeichen.

   Namen müssen innerhalb des SIMOTION Geräts eindeutig sein.
   Geschützte oder reservierte Bezeichner (Seite 102) sind nicht erlaubt.
   Bisher vorhandene Programmquellen (z. B. ST-Quellen, MCC-Quellen) werden angezeigt.
5. Zusätzlich können Sie Autor, Version und einen Kommentar eingeben.
6. Markieren Sie die Checkbox **Editor automatisch öffnen**.

7. Wählen Sie gegebenenfalls weitere Register, um dort lokale Einstellungen (nur für diese ST-Quelle gültig) vorzunehmen:
   - Register **Compiler**: Lokale Einstellungen des Compilers (Seite 67) für die Codegenerierung und die Anzeige der Meldungen.
   - Register **Weitere Einstellungen**: Definitionen für Präprozessor (Seite 75)
8. Bestätigen Sie mit **OK**.

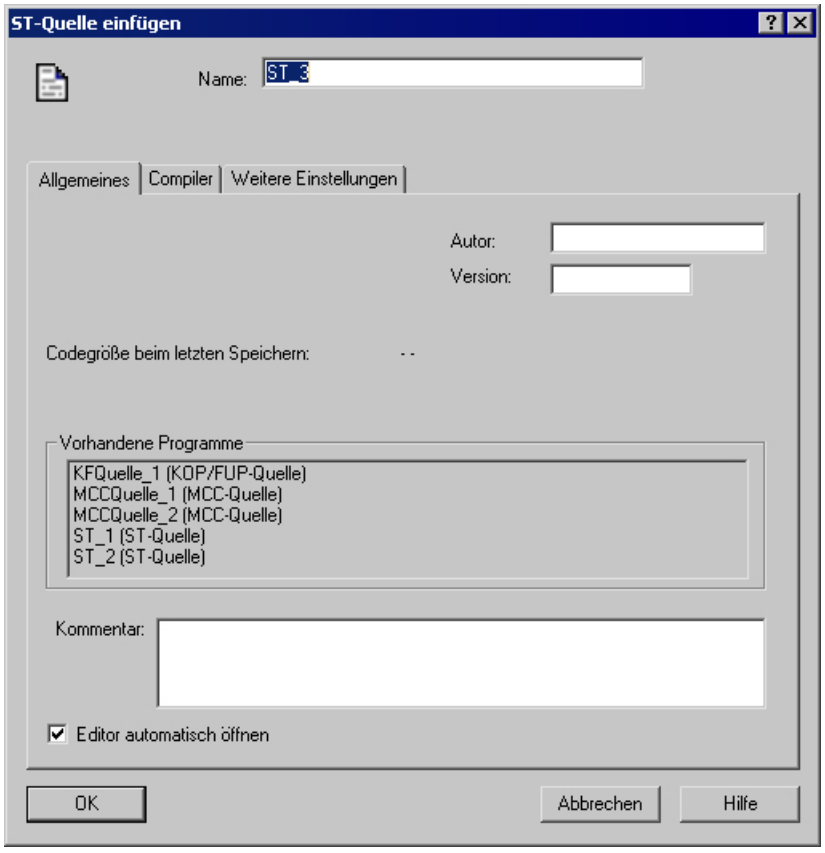


Bild 3-3 ST-Quelle einfügen

Hinweis

Mit **OK** wird die ST-Quelle nur in das Projekt übernommen. Auf dem Datenträger werden die Daten erst zusammen mit dem Projekt gespeichert, z. B. durch **Projekt > Speichern** oder **Projekt > Speichern und Änderungen übersetzen** oder **Projekt > Speichern und alles neu übersetzen**.

3.3.2 Vorhandene ST-Quelle öffnen

Vorgehensweise

So öffnen Sie eine ST-Quelle:

1. Öffnen Sie im Projektnavigator den Teilbaum des entsprechenden SIMOTION Geräts.
2. Öffnen Sie den Ordner **PROGRAMME**.
3. Markieren Sie die gewünschte ST-Quelle.
4. Wählen Sie Menü **Bearbeiten > Objekt öffnen**.
5. Nur bei ST-Quellen, die mit Know-how-Schutz versehen sind:
   Wenn die ST-Quelle nicht bereits geöffnet ist und das der ST-Quelle zugeordnete Login noch nicht angemeldet ist:
   - Geben Sie zum angezeigten Login das zugehörige Passwort ein.
     Der Know-how-Schutz wird für diese Quelle temporär (bis zu deren Schließen) aufgehoben.
   - Aktivieren Sie gegebenenfalls die Checkbox "Login als Standard-Login verwenden".
     Sie werden mit diesem Login angemeldet und können weitere Quellen, denen dasselbe Login zugeordnet ist, ohne erneute Eingabe des Passworts öffnen.

Dies ST-Quelle wird mit den gespeicherten Faltungsinformationen (Seite 40) und Lesezeichen (Seite 51) geöffnet. Es können mehrere Quellen geöffnet sein.

Hinweis

Sie können auch auf die gewünschte ST-Quelle doppelklicken, um sie zu öffnen.

3.3.3 Eigenschaften einer ST-Quelle ändern

Vorgehensweise

1. Öffnen Sie unter dem SIMOTION Gerät den Ordner **PROGRAMME**.
2. Markieren Sie die gewünschte ST-Quelle.

3. Wählen Sie Menü **Bearbeiten > Objekteigenschaften**.
4. Wählen Sie ggf. weitere Register, um dort lokale Einstellungen (nur für diese ST-Quelle gültig) vorzunehmen:
   - Register **Allgemeines**: Allgemeine Angaben zur ST-Quelle, z. B. Codegröße bei der letzten Übersetzung, Zeitstempel der letzten Änderung, Speicherort des Projekt (siehe Bild).
   - Register **Compiler**: Lokale Einstellungen des Compilers (Seite 67) für die Codegenerierung und die Anzeige der Meldungen.
   - Register **Weitere Einstellungen**: Definitionen für Präprozessor (Seite 75) sowie Anzeige der Compileroptionen (Seite 73) gemäß den aktuellen Einstellungen des Compilers.
   - Register **Übersetzung**: Anzeige der Compileroptionen (Seite 73) beim letzten Übersetzen der ST-Quelle.
   - Register **Objektadresse:** Einstellen der internen Objektadresse der ST-Quelle. Die Objektadressen anderer Programmquellen werden angezeigt.

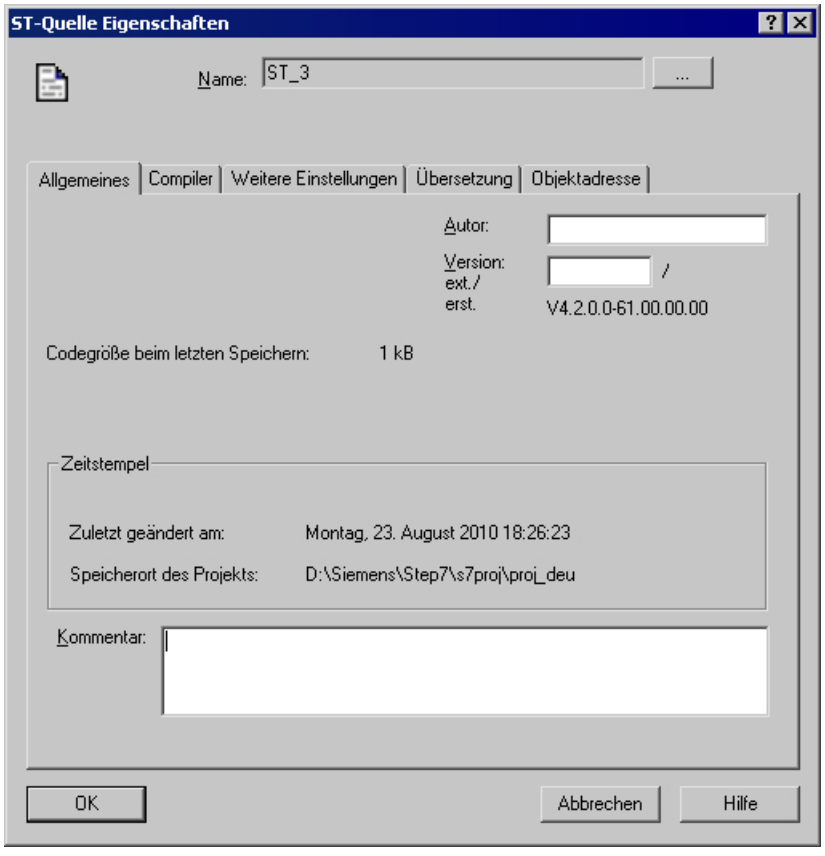


Bild 3-4 Eigenschaften einer ST-Quelle

Namen einer ST-Quelle ändern

Hier können Sie auch den Namen der ST-Quelle ändern. Klicken Sie hierzu auf den Button [...].

Namen für Programmquellen müssen den Regeln für Bezeichner genügen: Sie sind aus Buchstaben (A … Z, a … z), Ziffern (0 … 9) oder einzelnen Unterstrichen (_) in beliebiger Reihenfolge zusammengesetzt, wobei das erste Zeichen ein Buchstabe oder ein Unterstrich sein muss. Zwischen Groß- und Kleinbuchstaben wird nicht unterschieden.

Die zulässige Länge der Namen ist abhängig von der Version des SIMOTION Kernels:

- Ab Version V4.1 des SIMOTION Kernels: maximal 128 Zeichen.
- Bis Version V4.0 des SIMOTION Kernels: maximal 8 Zeichen.

Namen müssen innerhalb des SIMOTION Geräts eindeutig sein.

Geschützte oder reservierte Bezeichner (Seite 102) sind nicht erlaubt.

Bisher vorhandene Programmquellen (z. B. ST-Quellen, MCC-Quellen) werden angezeigt.

Hinweis

Bei Versionen des SIMOTION Kernels bis V4.0 wird eine Überschreitung der zulässigen Länge des Namens einer Programmquelle unter Umständen erst bei einer Konsistenzprüfung oder beim Download der Programmquelle erkannt!

3.3.4 Mit dem ST-Editor umgehen

Der ST-Editor erleichtert Ihnen den Umgang mit der ST-Quelle, Variablen und Technologieobjekten durch folgende Bedienelemente:

- Syntaxcoloring
- Drag&Drop
- Menübefehle und Shortcuts

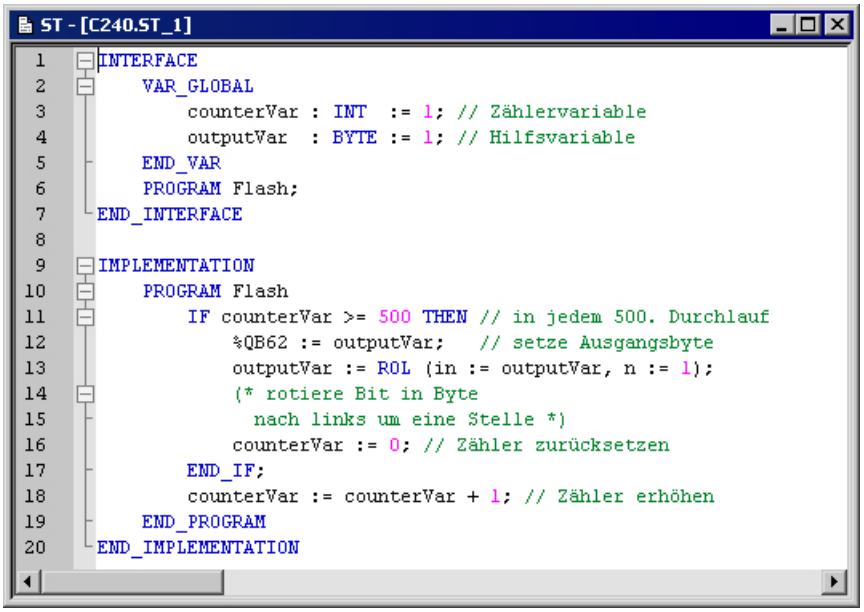


Bild 3-5 Geöffnete ST-Quelle im ST-Editor

3.3.4.1 Syntaxcoloring

Der ST-Editor stellt Sprachelemente mit verschiedenen Farben dar:

- Blau: Schlüsselwörter und Compiler-Buildin-Funktionen
- Magenta: Zahlen, Werte
- Grün: Kommentare
- Schwarz: Technologieobjekte, Anwendercode, Variablen

3.3.4.2 Drag&Drop

Drag&Drop

Mit Drag&Drop (Ziehen mit gedrückter linker Maustaste) können Sie:

- Markierte Textbereiche innerhalb der ST-Quelle oder in eine andere geöffnete ST-Quelle verschieben.
- Variablennamen aus dem Symbol-Browser in die ST-Quelle kopieren.
- Namen (z. B. von Technologieobjekten, Funktionen, Funktionsbausteinen) aus dem Projektnavigator in die ST-Quelle kopieren.
- Systemfunktionen aus der Befehlsbibliothek in die ST-Quelle kopieren

So kopieren Sie Variablennamen aus dem Symbol-Browser in die ST-Quelle:

1. Markieren Sie im Symbol-Browser die ganze Zeile der gewünschten Variablen. Klicken Sie dazu auf die Zeilennummer am Beginn der Zeile.
2. Ziehen Sie mit gedrückter linker Maustaste die Zeilennummer an die gewünschte Stelle innerhalb der ST-Quelle.
   Der Name der ausgewählten Variablen wird in die ST-Quelle eingefügt.

So kopieren Sie den Namen eines Elements (z. B. eines Technologieobjekts, einer Funktion oder eines Funktionsbausteins) aus dem Projektnavigator in die ST-Quelle:

1. Wählen Sie im Projektnavigator das Register **Projekt**.
2. Markieren Sie das Element im Projektnavigator.
3. Ziehen Sie mit gedrückter linker Maustaste das Element an die gewünschte Stelle innerhalb der ST-Quelle.
   Der Name des ausgewählten Elements wird in die ST-Quelle eingefügt.

So kopieren Sie eine Systemfunktion aus der Befehlsbibliothek in die ST-Quelle:

1. Wählen Sie im Projektnavigator das Register **Befehlsbibliothek**.
2. Markieren Sie die Systemfunktion in der Befehlsbibliothek.
3. Ziehen Sie mit gedrückter linker Maustaste die Systemfunktion an die gewünschte Stelle innerhalb der ST-Quelle.
   Die Systemfunktion wird mit ihren Parametern in die ST-Quelle eingefügt.

3.3.4.3 Einstellungen des ST-Editors

Vorgehensweise:

1. Wählen Sie das Menü **Extras > Einstellungen**.
2. Wählen Sie das Register **ST-Editor / Skripting**.
3. Nehmen Sie die Einstellungen vor.
4. Bestätigen Sie mit **OK** oder **Übernehmen**.

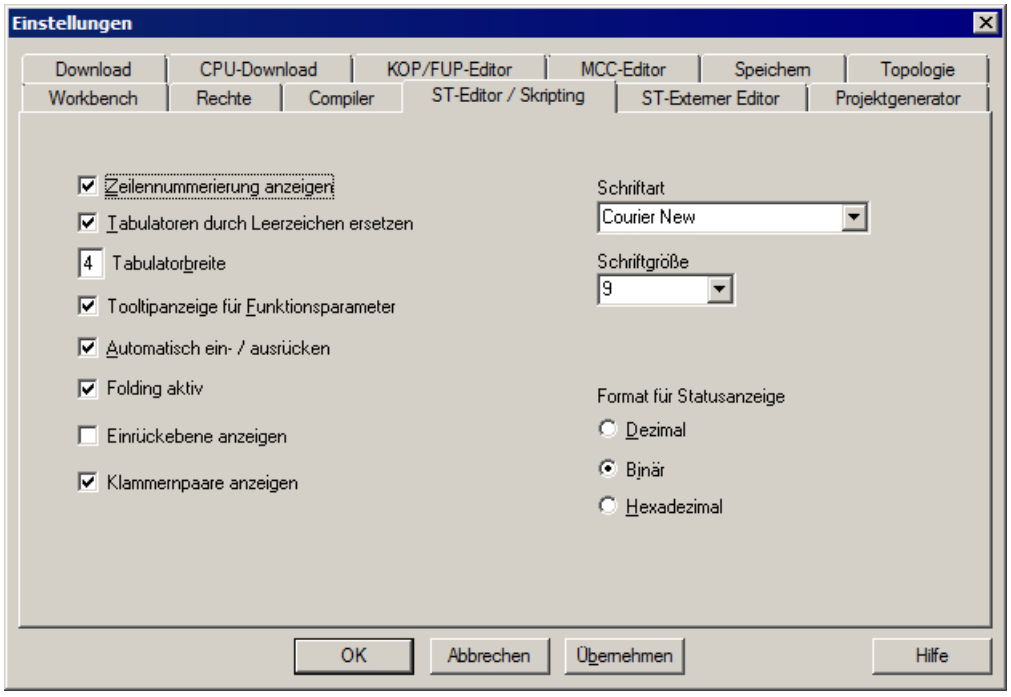


Bild 3-6 Einstellungen ST-Editor / Skripting

Die Einstellungen sind auch gültig für den Skript-Editor.

In der folgenden Tabelle finden Sie eine Beschreibung der einzelnen Parameter:

Tabelle 3-1 Parameter Einstellungen ST-Editor / Skripting

Parameter	Beschreibung
Zeilennummerierung anzeigen	Wenn aktiv, werden Zeilennummern angezeigt. Siehe: Weitere Hilfsmittel des ST-Editors (Seite 54).
Tabulatoren durch Leerzeichen ersetzen	Hier wählen Sie, wie das Einrücken von Text erfolgt (beim automatischen Einrücken bzw. durch Drücken der Tabulatortaste): • Wenn aktiv: Durch Einfügen der entsprechenden Anzahl von Leerzeichen ($20). • Wenn inaktiv: Durch Einfügen des Tabulatorzeichen ($09). Siehe: Einzüge und Tabulatoren (Seite 37).
Tabulatorbreite	Anzahl der durch einen Tabulator übersprungenen Zeichen. Siehe: Einzüge und Tabulatoren (Seite 37).
Tooltippanzeige für Funktionsparameter	Wenn aktiv, werden bei Funktionen die Parameter als Tooltipp angezeigt.
Automatisch ein- / ausrücken	Wenn aktiv, werden bei der Texteingabe Quelldateiabschnitte und Blöcke automatisch um die eingestellte Tabulatorbreite eingerückt. Siehe: Einzüge und Tabulatoren (Seite 37).

Parameter	Beschreibung
Folding aktiv	Wenn aktiv, wird links neben dem Editierbereich die Spalte mit den Faltungsinformationen eingeblendet. Sie können dann Blöcke in einer ST-Quelle ausblenden, so dass nur die erste Zeile des Blocks sichtbar bleibt. Siehe: Falten (Blöcke ein- und ausblenden) (Seite 40).
Einrückebene anzeigen	Wenn aktiv, werden die Ein- und Ausrückungen bei Blöcken durch senkrechte Hilfslinien (entsprechend der eingestellten Tabulatorbreite) optisch hervorgehoben. Siehe: Einzüge und Tabulatoren (Seite 37).
Klammernpaare anzeigen	Wenn aktiv, wird zu einer Klammer, an der der Cursor steht, die zugehörige Klammer des Paars gesucht und optisch hervorgehoben. Siehe: Weitere Hilfsmittel des ST-Editors (Seite 54).
Schriftart	Schriftart für die Anzeige des Textes im Editor. Zur Auswahl stehen alle auf dem PC installierten Nichtproportionalschriften.
Schriftgröße	Schriftgröße (in pt) für die Anzeige des Textes im Editor. Siehe: Schriftgröße im ST-Editor ändern (Seite 45).
Format für Statusanzeige [1]	Format, in dem die Werte von Variablen mit Bitdatentyp bei Status Programm oder Status Variable angezeigt werden (nur für ST-Editor). Siehe: Eigenschaften von Status Programm (Seite 340) , Status Variable (Seite 337)

1 Nur beim ST-Editor

3.3.4.4 Einzüge und Tabulatoren

Tabulatorbreite festlegen

Die Standard-Tabulatorbreite für alle ST-Quellen legen Sie in den Einstellungen des ST-Editors (Seite 35) fest.

Diese Einstellung gilt für alle ST-Quellen, die anschließend geöffnet werden.

Einrücken durch Tabulatoren bzw. Leerzeichen

Sie können in den Einstellungen des ST-Editors (Seite 35) wählen, wie das Einrücken von Text erfolgt (z. B. beim automatischen Ein- und Ausrücken bzw. durch Drücken der Tabulatortaste):

- Durch Einfügen der entsprechenden Anzahl von Leerzeichen ($20).
- Durch Einfügen des Tabulatorzeichens ($09).

Diese Einstellung gilt für alle ST-Quellen, die anschließend geöffnet werden.

Blöcke automatisch ein- und ausrücken

Der ST-Editor erkennt Blöcke, die durch ein Schlüsselwort eingeleitet und durch ein weiteres Schlüsselwort beendet werden, z. B.:

- INTERFACE / END_INTERFACE
- IMPLEMENTATION / END_IMPLEMENTATION
- Deklarationsblöcke (z. B. TYPE / END_TYPE, VAR / END_VAR)
- Programmorganisationseinheiten (z. B. PROGRAM / END_PROGRAM)
- Kontrollanweisungen (z. B. IF / END_IF, FOR / END_FOR)

Während der Texteingabe kann der ST-Editor Texte innerhalb von Blöcken automatisch um eine Tabulatorweite einrücken. Die Endzeile des Blocks wird automatisch ausgerückt.

Diese Funktion aktivieren Sie in den Einstellungen des ST-Editors (Seite 35).

Hinweis

Diese Einstellung beeinflusst nur das Verhalten bei der Texteingabe. Sie hat keine Auswirkung auf bestehende Texte in den ST-Quellen.

Aktuelle Auswahl formatieren

Mit dieser Funktion können Sie erzwingen, dass in einem bestehenden Text die Blöcke (siehe oben) gemäß ihrer Hierarchie um jeweils eine Tabulatorweite eingerückt werden. Die Anzahl der führenden Leerzeichen bzw. Tabulatoren wird angepasst:

- Gemäß der aktuellen Tabulatorweite der ST-Quelle.
- Gemäß der aktuellen Einstellung für die Art des Einzugs (mit Tabulatoren oder Leerzeichen).

So gehen Sie vor:

1. Markieren Sie den Textbereich im ST-Editor, den Sie formatieren wollen (siehe Text markieren (Seite 46)).
2. Wählen Sie das Kontextmenü **Ansicht > Aktuelle Auswahl formatieren**.

Hinweis

In einer Zeile werden führende Tabulatoren oder Leerzeichen nur dann ersetzt, wenn sich beim Formatieren ihre Anzahl ändert.

Markierten Bereich einrücken bzw. ausrücken

Mit dieser Funktion können Sie markierte Textabschnitte um jeweils eine Tabulatorweite einrücken bzw. ausrücken.

So gehen Sie vor:

1. Markieren Sie den Textbereich im ST-Editor, den Sie einrücken bzw. ausrücken wollen (siehe Text markieren (Seite 46)).
   Der markierte Textbereich muss mehrere Zeilen umfassen.
2. Wählen Sie das Kontextmenü **Markierten Bereich einrücken** bzw. **Markierten Bereich rückgängig**.

Einrückhilfe (Einrückebene anzeigen)

Die Ein- und Ausrückungen bei Blöcken können Sie durch senkrechte Hilfslinien (entsprechend der eingestellten Tabulatorweite) optisch hervorheben.

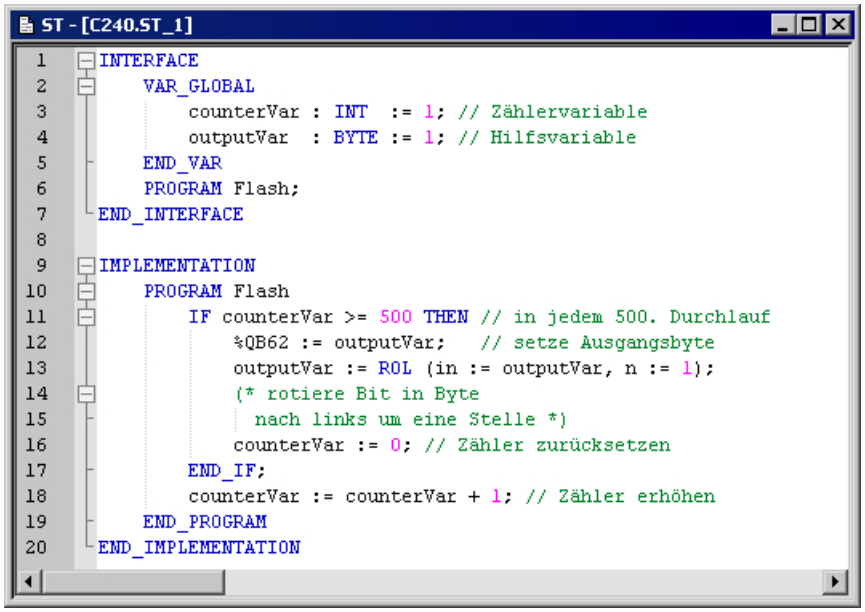


Bild 3-7 ST-Quelle mit sichtbarer Einzugshilfe

Diese Funktion können Sie aktivieren oder deaktivieren:

- Für die aktive ST-Quelle
  - Wählen Sie das Kontextmenü **Ansicht > Einrückhilfe**.

  Diese Einstellung wird beim Schließen der ST-Quelle nicht gespeichert.
- Für alle geöffneten ST-Quellen:
  - Aktivieren oder deaktivieren Sie die Checkbox **Einrückebene anzeigen** in den Einstellungen des ST-Editors (Seite 35).

  Diese Einstellung gilt auch für alle ST-Quellen, die anschließend geöffnet werden.

3.3.4.5 Falten (Blöcke ein- und ausblenden)

Sie können Blöcke in einer ST-Quelle ausblenden, so dass nur die erste Zeile des Blocks sichtbar bleibt. Dies erhöht die Übersichtlichkeit beim Bearbeiten oder Lesen einer längeren ST-Quelle.

Ein Block wird durch ein Schlüsselwort eingeleitet und durch ein weiteres Schlüsselwort beendet, z. B.:

- INTERFACE / END_INTERFACE
- IMPLEMENTATION / END_IMPLEMENTATION
- Deklarationsblöcke (z. B. TYPE / END_TYPE, VAR / END_VAR)
- Programmorganisationseinheiten (z. B. PROGRAM / END_PROGRAM)
- Kontrollanweisungen (z. B. IF / END_IF, FOR / END_FOR)
- Blockkommentar (* / *)

Zum Ein- und Ausblenden von Blöcken muss das Falten aktiviert sein, siehe unten. Die Spalte mit den Faltungsinformationen (links neben dem Editierbereich) ist dann eingeblendet.

So erkennen Sie, dass bei aktiviertem Falten ein Block eingeblendet ist:

- Sie sehen in Höhe der ersten Zeile des Blocks das Zeichen ⊟ (Minus).

So erkennen Sie, dass ein Block ausgeblendet ist:

- Sie sehen in der Höhe der sichtbaren ersten Zeile des Blocks das Zeichen ⊞ (Plus).
- Unter dieser Zeile wird ein Querstrich angezeigt. Dieser Querstrich ist auch dann sichtbar, wenn die Spalte mit den Faltungsinformationen ausgeblendet ist.

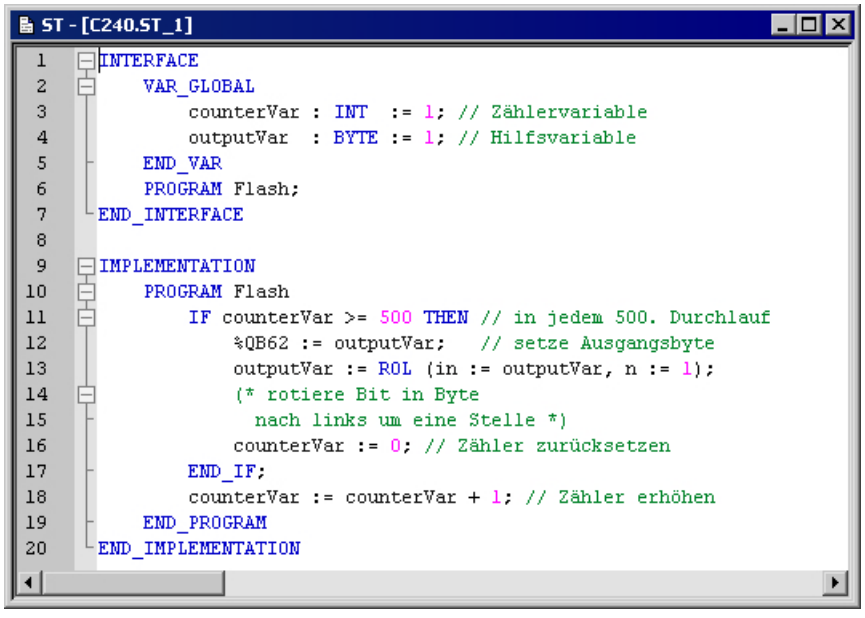


Bild 3-8 ST-Quelle, bei der alle Blöcke eingeblendet sind

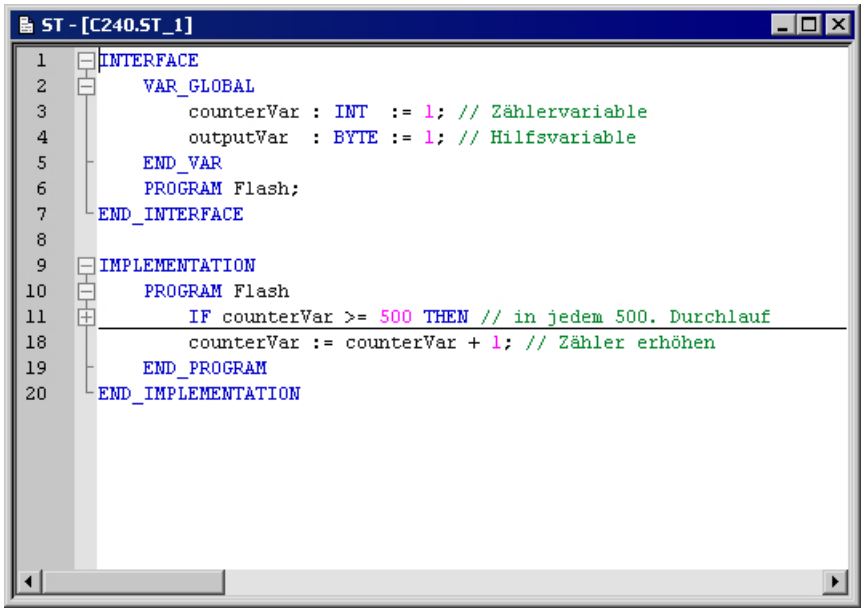


Bild 3-9 ST-Quelle mit ausgeblendetem IF-Block (einschließlich Blockkommentar)

Falten aktivieren oder deaktivieren

Um das Ein- und Ausblenden von Blöcken nutzen zu können, muss das Falten aktiviert sein. Die Spalte mit den Faltungsinformationen (links vom Editierbereich) ist dann eingeblendet.

So aktivieren oder deaktivieren Sie das Falten:

- Für die aktive ST-Quelle:
  - Wählen Sie das Kontextmenü **Ansicht > Folding**.

  Diese Einstellung wird beim Schließen der ST-Quelle nicht gespeichert.
- Für alle geöffneten ST-Quellen:
  - Aktivieren oder deaktivieren Sie die Checkbox **Folding aktiv** in den Einstellungen des ST-Editors (Seite 35).

  Diese Einstellung gilt auch für alle ST-Quellen, die anschließend geöffnet werden.

Einblenden und Ausblenden von Blöcken

Sie können Blöcke nur dann ein- oder ausblenden, wenn für die Quelle das Falten aktiviert ist (siehe oben). Die Spalte mit den Faltungsinformationen ist dann eingeblendet.

So blenden Sie Blöcke ein oder aus:

- Einzelnen Block ausblenden (Alternative):
  - Klicken Sie mit der Maus auf das Zeichen ⊟ (Minus) in der Spalte mit den Faltungsinformationen.
  - Setzen Sie den Cursor in die entsprechende Zeile des Blocks und drücken Sie die Tastenkombination **STRG+ALT+T**.

  Nur die erste Zeile des Blocks bleibt sichtbar. Alle nachfolgenden Zeilen des Blocks (einschließlich der Zeilen untergeordneter Blöcke) werden ausgeblendet.
  Der Ein- und Ausblendstatus untergeordneter Blöcke wird gespeichert. Beim Einblenden einzelner Blöcke wird dieser wieder hergestellt.
- Einzelnen Block einblenden (Alternative):
  - Klicken Sie mit der Maus auf das Zeichen ⊞ (Plus) in der Spalte mit den Faltungsinformationen.
  - Setzen Sie den Cursor in die sichtbare Zeile des Blocks und drücken Sie die Tastenkombination **STRG+ALT+T**.

  Alle nachfolgenden Zeilen des Blocks werden eingeblendet. Untergeordnete Blöcke werden wie folgt angezeigt: Wenn der Ein- und Ausblendstatus gespeichert wurde (z. B. beim Ausblenden einzelner Blöcke) wird dieser wieder hergestellt.
- Alle Blöcke ausblenden:
  - Drücken Sie die Tastenkombination **STRG+ALT+C**.

  Alle Blöcke der ST-Quelle (einschließlich aller untergeordneten Blöcke) werden ausgeblendet. Nur die jeweils erste Zeile der Blöcke 1. Ordnung bleibt sichtbar (in der Regel INTERFACE und IMPLEMENTATION).
- Alle Blöcke einblenden:
  - Drücken Sie die Tastenkombination **STRG+ALT+D**.

  Alle Blöcke der ST-Quelle (einschließlich aller untergeordneten Blöcke) werden eingeblendet. Alle Zeilen der ST-Quelle sind sichtbar.
- Untergeordnete Blöcke ausblenden:
  - Setzen Sie den Cursor in die entsprechende Zeile des Blocks und drücken Sie die Tastenkombination **STRG+ALT+V**.

  Alle untergeordneten Blöcke zum aktuellen Block werden ausgeblendet, der aktuelle Block wird eingeblendet. Nur die Zeilen des aktuellen Block sowie die ersten Zeilen der Blöcke nächster Ordnung sind sichtbar.
- Untergeordnete Blöcke einblenden:
  - Setzen Sie den Cursor in die entsprechende Zeile des Blocks und drücken Sie die Tastenkombination **STRG+ALT+R**.

  Der aktuelle Block und alle untergeordneten Blöcke werden eingeblendet. Alle Zeilen dieser Blöcke sind sichtbar.

Hinweis

Die Informationen über ein- und ausgeblendete Blöcke werden beim Schließen der ST-Quelle im Projekt gespeichert.

Beim Export der ST-Quelle (Seite 77) werden diese Informationen nicht übernommen.

3.3.4.6 Editorfenster teilen

Sie können das Fenster des ST-Editor in zwei Segmente teilen. Dies ermöglicht zwei Sichten auf dieselbe ST-Quelle.

Fenster teilen bzw. Teilung aufheben

So gehen Sie vor, um das aktuelle Editorfenster in zwei Segmente zu teilen oder die Teilung aufzuheben.

- Wählen Sie Menü **ST-Quelle > Fenster teilen**.

Der Cursor befindet sich nach der Teilung im oberen Segment des Fensters.

Die Einstellungen zur Teilung des Editorfensters werden beim Schließen der ST-Quelle nicht gespeichert. ST-Quellen werden immer in einem ungeteilten Fenster geöffnet.

Hinweis

Bei Verwendung der Testfunktion Status Programm (Seite 342) kann das Editorfenster nicht geteilt werden.

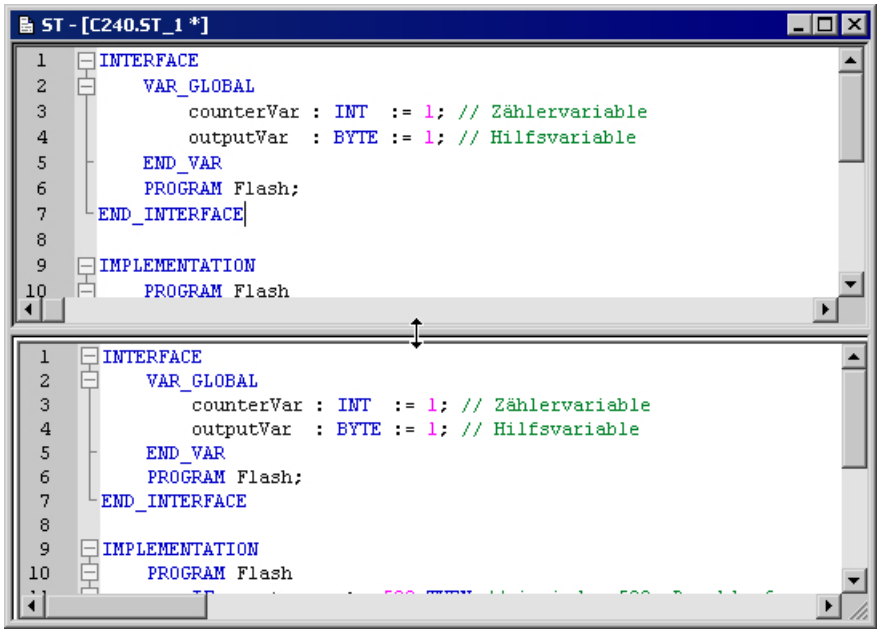


Bild 3-10 ST-Editor mit Fensterteilung

Segmenthöhe anpassen

Sie können die Höhe der beiden Segmente Ihren individuellen Erfordernissen anpassen:

1. Bewegen Sie mit der Maus den Cursor zur Trennlinie zwischen den beiden Segmenten, bis er die Form eines Doppelpfeils annimmt (siehe obiges Bild).
2. Ziehen Sie mit gedrückter linker Maustaste die Trennlinie an die gewünschte Stelle.

3.3.4.7 Leerzeichen und Tabulatoren anzeigen

Sie können Leerzeichen und Tabulatoren in den ST-Quellen anzeigen.

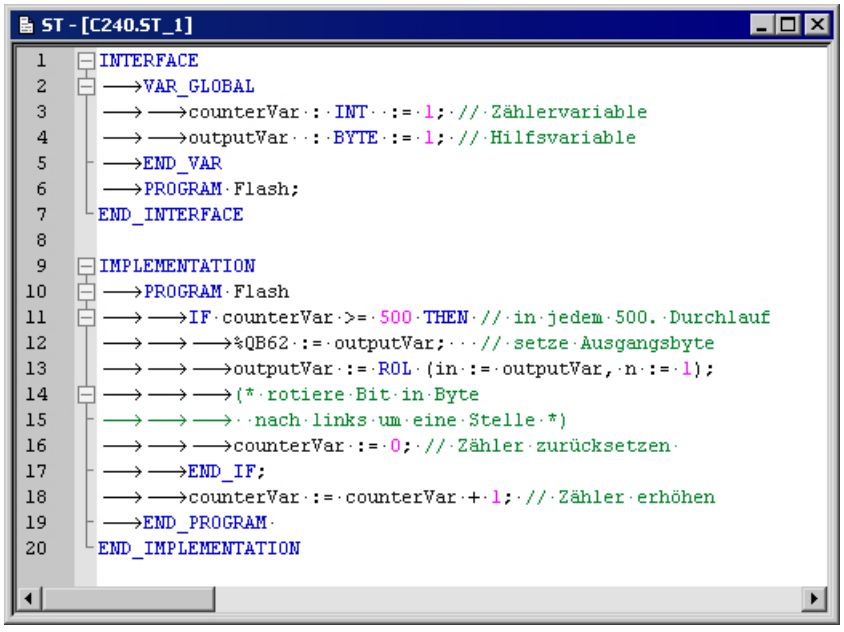


Bild 3-11 ST-Quelle mit sichtbaren Leerzeichen und Tabulatoren

Vorgehensweise

So schalten Sie um, ob Leerzeichen und Tabulatoren in der aktiven ST-Quelle angezeigt werden:

1. Setzen Sie den Cursor in die geöffnete ST-Quelle.
2. Wählen Sie das Kontextmenü **Ansicht > Formatierungssymbole**.

Diese Einstellung wird beim Schließen der ST-Quelle nicht gespeichert.

3.3.4.8 Schriftgröße im ST-Editor ändern

Sie können die Schriftgröße der ST-Quelle im Editor verändern. Dabei werden auch die Schriftgröße der Zeilennummern und die Größe anderer Anzeigeelemente (z. B. Faltmarkierungen, Lesezeichen) angepasst.

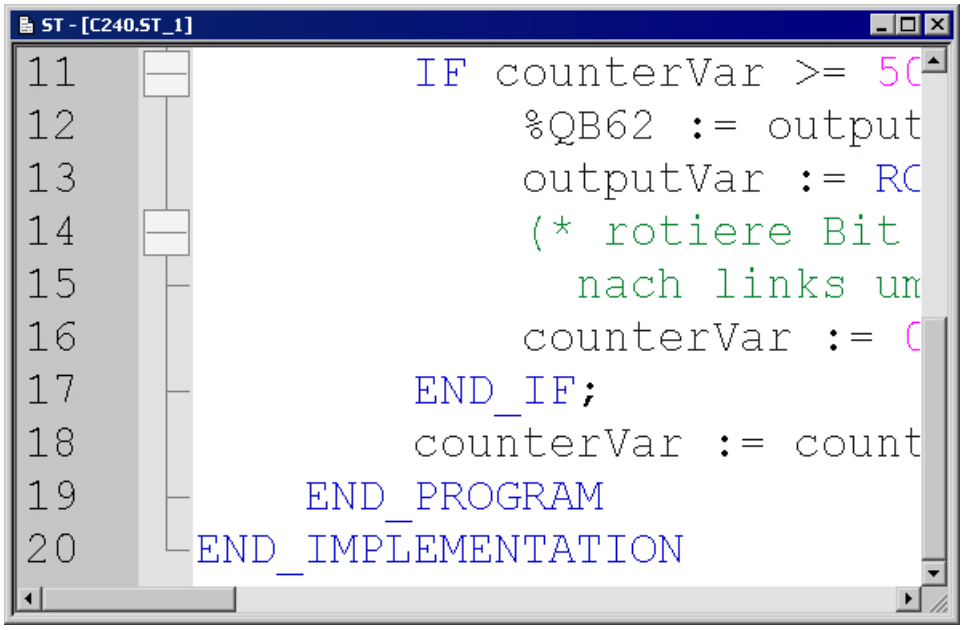


Bild 3-12 Vergrößerte Anzeige der ST-Quelle

Vorgehensweise

Sie können die Schriftgröße verändern:

- für die aktuelle ST-Quelle
- für künftig zu öffnende ST-Quellen

So verändern Sie die Schriftgröße für die aktuelle ST-Quelle (Alternative):

- Drücken Sie die Taste **STRG** und bewegen Sie gleichzeitig das Mausrad.
- Drücken Sie gleichzeitig die Taste **STRG** und eine der folgenden Tasten auf dem Ziffernblock:
  - **ADD** (**+**) zum Vergrößern,
  - **MINUS** (**-**) zum Verkleinern,
  - **DIV** für 100 %.

So verändern Sie die Schriftgröße für künftig zu öffnende ST-Quellen

1. Öffnen Sie die Einstellungen für den ST-Editor (siehe Einstellungen des ST-Editors (Seite 35)).
2. Geben Sie die gewünschte Schriftgröße ein.

Die Einstellung wird wirksam bei allen ST-Quellen, die anschließend geöffnet werden. Sie wirkt sich nicht auf die aktuell geöffneten ST-Quellen aus.

3.3.4.9 Text markieren

Text zeilenweise markieren

So gehen Sie vor, um Text zeilenweise zu markieren:

- Mit der Maus:
  - Überstreichen Sie mit gedrückter linker Maustaste den zu markierenden Text.

  oder

- Mit der Tastatur oder der Maus:
  - Setzen Sie den Cursor mit den Pfeiltasten der Tastatur oder mit der Maus an den Beginn des zu markierenden Textes.
  - Drücken Sie die Taste **SHIFT** und setzen Sie gleichzeitig den Cursor an das Ende des zu markierenden Textes. Siehe hierzu auch die Tastenkürzel bzw. Shortcuts (Seite 60).

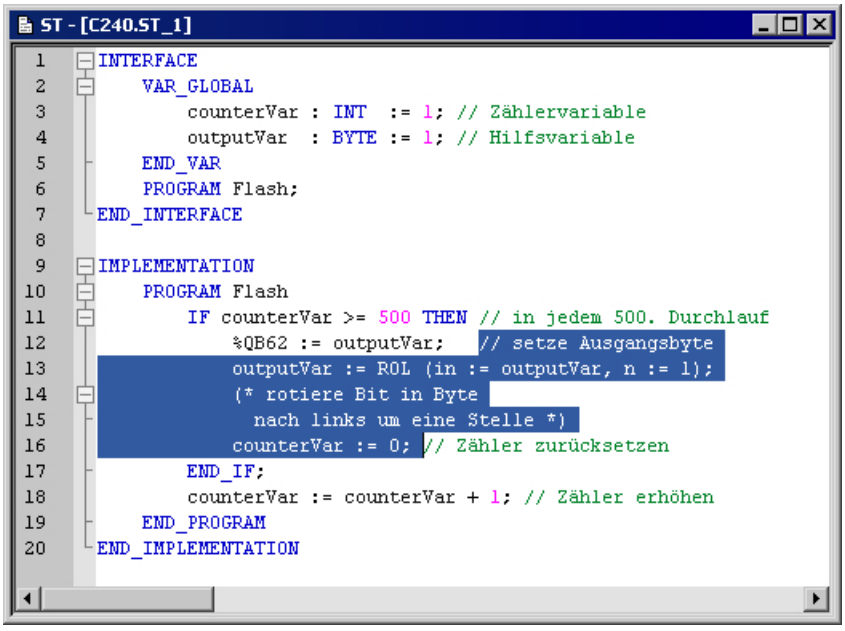


Bild 3-13 ST-Quelle mit zeilenweise markiertem Text

Text spaltenweise markieren

So gehen Sie vor, um Text spaltenweise zu markieren:

- Mit der Maus:
  - Drücken Sie die Taste **ALT** und überstreichen Sie gleichzeitig mit gedrückter linker Maustaste den zu markierenden Text.

  oder

- Mit der Tastatur oder Maus:
  - Setzen Sie den Cursor mit den Pfeiltasten der Tastatur oder mit der Maus an den Beginn des zu markierenden Textes.
  - Drücken Sie die Tastenkombination **ALT+SHIFT** und setzen gleichzeitig Sie den Cursor an das Ende des zu markierenden Textes. Siehe hierzu auch die Tastenkürzel bzw. Shortcuts (Seite 60).

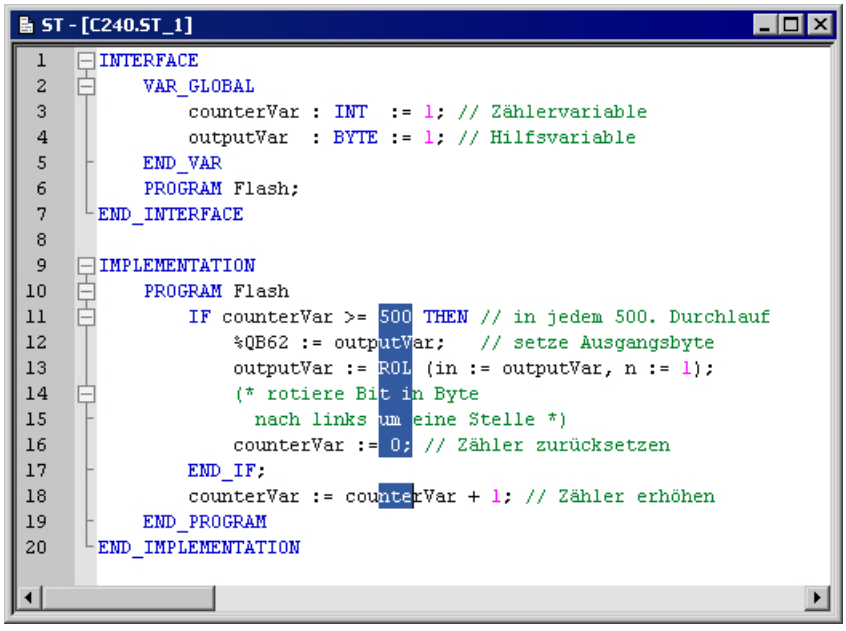


Bild 3-14 ST-Quelle mit spaltenweise markiertem Text

Einzelne Zeile markieren

So gehen Sie vor, um eine einzelne Zeile zu markieren:

- Klicken Sie mit der linken Maustaste rechts neben die Zeilennummer der entsprechenden Zeile.

Gesamten Text markieren

So gehen Sie vor, um den gesamten Text zu markieren (Alternativen):

- Drücken Sie die Taste **STRG** und klicken gleichzeitig mit der linken Maustaste in die Spalte mit den Zeilennummern.
- Drücken Sie die Tastenkombination **Strg+A**.

3.3.4.10 Einfache Zahlenfolge erzeugen (Sequenz erzeugen)

Sie können im ST-Editor einfache ganzzahlige Zahlenfolgen (Sequenzen) erzeugen, die einem bestimmten Muster folgen. Dies kann z. B. bei der Initialisierung eines Feldes nützlich sein.

Vorgehensweise

1. Markieren Sie im ST-Editor alle Zahlen, die Sie durch eine Zahlenfolge ersetzen wollen (z. B. spaltenweise), siehe Text markieren (Seite 46).
2. Drücken Sie die Tastenkombination **STRG+SHIFT+F7**.

Der ST-Editor interpretiert alle markierten Zahlen als mögliche Folgenglieder und versucht, aus den Anfangsgliedern das Muster der Folge zu erkennen und die weiteren Folgenglieder zu berechnen (siehe unten). Die Anfangsglieder der Folge müssen hierbei ganzzahlige Werte darstellen. Nach der Berechnung ersetzt der ST-Editor alle markierten Zahlen, die auf die Anfangsglieder folgen, durch die berechneten Werte.

Bild 3-15 Beispiel für das Erzeugen von Zahlenfolgen im ST-Editor

Berechnung der Folgenglieder

Alle im ST-Editor markierten Zahlen werden als mögliche Folgenglieder interpretiert. Die Anfangsglieder, aus denen die Folge berechnet wird, müssen hierbei ganzzahlige Werte darstellen.

Bei der Ermittlung der Zahlenfolge wird angenommen, dass sich ein Folgenglied a_{n+1} aus dem jeweils vorhergehenden Folgenglied a_n mit einer linearen Funktion mit den ganzzahligen Koeffizienten j und k bestimmen lässt. Es gilt somit folgende Formel:

$$a_{n+1} = f(a_n) = j \cdot a_n + k$$

Die Koeffizienten j und k werden aus den Anfangsgliedern der Folge berechnet:

1. Zuerst wird versucht, die ganzzahligen Koeffizienten j und k aus den 3 Anfangsgliedern der Zahlenfolge a_1, a_2 und a_3 zu bestimmen ($j \geq 0$).
2. Ist dies nicht möglich, werden nur die ersten beiden Folgenglieder a_1 und a_2 zur Auswertung herangezogen:
   - Der Koeffizient j wird = 1 gesetzt.
   - Der Koeffizient k wird aus der resultierenden Formel $a_2 = a_1 + k$ bestimmt.

Mit diesen Werten wird die Zahlenfolge berechnet. Alle markierten Zahlen werden durch die berechneten Folgenglieder ersetzt; die verwendeten Anfangsglieder der Folge bleiben unverändert.

Die nachfolgende Tabelle zeigt einige Beispiele für Zahlenfolgen, die auf diese Weise berechnet werden.

Tabelle 3-2 Beispiele für lineare Zahlenfolgen mit vorgegebenen Anfangsgliedern

Anfangsglieder der Folgen			Berechnetes Ergebnis			Resultierende Zahlenfolge
a_1	a_2	a_3	j	k	Formel	
0	1	2	1	1	$a_{n+1} = a_n + 1$	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 …
1	2	3	1	1	$a_{n+1} = a_n + 1$	1, 2, 3, 4, 5, 6, 7, 8, 9, 10 …
0	2	4	1	2	$a_{n+1} = a_n + 2$	0, 2, 4, 6, 8, 10, 12, 14, 16, 18 …
1	2	4	2	0	$a_{n+1} = 2a_n$	1, 2, 4, 8, 16, 32, 64, 128, 256, 512 …
2	3	5	2	-1	$a_{n+1} = 2a_n - 1$	2, 3, 5, 9, 17, 33, 65, 129, 257, 513 …
2	1	1	0	1	$a_{n+1} = 1$	2, 1, 1, 1, 1, 1, 1, 1, 1, 1 …
3	2	1	1	-1	$a_{n+1} = a_n - 1$	3, 2, 1, 0, -1, -2, -3, -4, -5, -6 …
0	1	0 [1]	1	1	$a_{n+1} = a_n + 1$	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 …
0	0	2 [1]	1	0	$a_{n+1} = a_n$	0, 0, 0, 0, 0, 0, 0, 0, 0, 0 …

[1] Nur die Anfangsglieder a_1 und a_2 der Folge werden berücksichtigt, a_3 bleibt unberücksichtigt und wird berechnet.

Hinweis

Die zur Berechnung verwendeten Anfangsglieder der Folge müssen ganzzahlige Werte darstellen. Sie können jedoch auch in der Gleitpunktdarstellung angegeben werden.

Bei der Folgenberechnung zu berücksichtigende Zahlen müssen vollständig markiert werden.

Nur Zahlen und ausgewählte Sonderzeichen (z. B. öffnende und schließende Klammern, Kommata) dürfen sich im markierten Bereich befinden.

Beim ersten unzulässigen Zeichen im markierten Bereich wird die Berechnung der Folge abgebrochen.

Siehe Beispiele im nachfolgenden Bild.

Bild 3-16 Beispiele, wie falsche Markierungen zu unerwünschten Ergebnissen beim Erzeugen einer Zahlenfolge im ST-Editor führen

3.3.4.11 Lesezeichen verwenden

Sie können im ST-Editor Lesezeichen setzen. Damit können Sie innerhalb der ST-Quelle gezielt zu ausgewählten Zeilen springen.

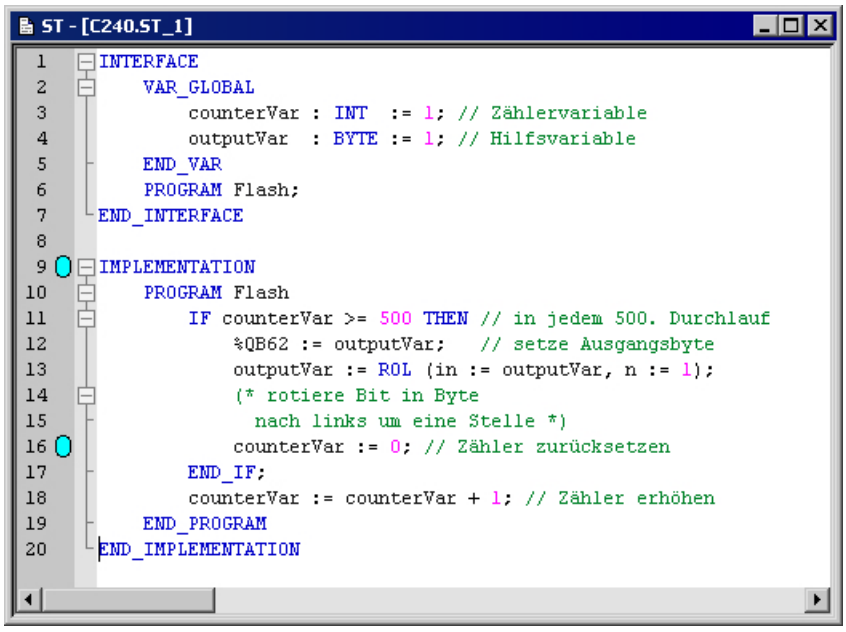


Bild 3-17 ST-Quelle mit Lesezeichen

Lesezeichen setzen und löschen

So gehen Sie vor, um an einer Zeile der aktiven ST-Quelle ein Lesezeichen zu setzen bzw. ein vorhandenes Lesezeichen zu löschen:

- Mit Tastatur und Maus:
  - Drücken Sie die Taste **STRG**.
  - Klicken Sie gleichzeitig mit der linken Maustaste rechts neben die Zeilennummer der entsprechenden Zeile.
- Mit der Maus:
  - Setzen Sie den Cursor in die entsprechende Zeile der ST-Quelle.
  - Wählen Sie das Kontextmenü **Lesezeichen > Lesezeichen einfügen/entfernen**.

Hinweis

Lesezeichen werden beim Schließen der ST-Quelle gespeichert.

Zu Lesezeichen springen

So gehen Sie vor, um zum nächsten Lesezeichen innerhalb der ST-Quelle zu springen:

- Wählen Sie das Kontextmenü **Lesezeichen > Nächstes Lesezeichen**.

So gehen Sie vor, um zum vorherigen Lesezeichen innerhalb der ST-Quelle zu springen:

- Wählen Sie das Kontextmenü **Lesezeichen > Vorheriges Lesezeichen**.

Alle Lesezeichen löschen

So gehen Sie vor, um alle Lesezeichen in einer ST-Quelle zu löschen:

- Wählen Sie das Kontextmenü **Lesezeichen > Alle Lesezeichen entfernen**.

3.3.4.12 Automatisches Vervollständigen

Sie können im ST-Editor Bezeichner automatisch vervollständigen lassen. Es wird eine Auswahlliste mit Bezeichnern eingeblendet, die mit den bisher eingegebenen Zeichen beginnen.

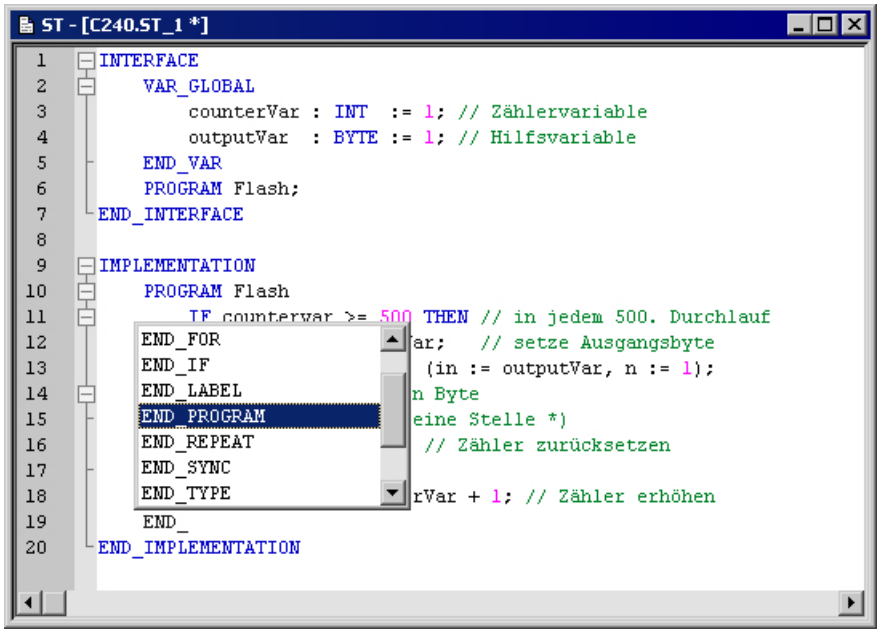


Bild 3-18 ST-Editor Automatisches Vervollständigen eines Bezeichners (z. B. END_)

Vorgehensweise

So vervollständigen Sie automatisch einen Bezeichner:

1. Schreiben Sie die ersten Zeichen des Bezeichners (z. B. Buchstaben eines Wortes).
2. Drücken Sie die Tastenkombination **STRG+Leerzeichen**.
   In einem Fenster werden die Auswahlmöglichkeiten angezeigt.
3. Erweitern oder verfeinern Sie die angezeigten Auswahlmöglichkeiten:
   - Geben Sie weitere Zeichen ein.
   - Löschen Sie Zeichen.
   - Verschieben Sie den Cursor mit den Pfeiltasten links/rechts.

4. Wählen Sie mit den Pfeiltasten auf/ab den gewünschten Bezeichner aus.
5. Übernehmen Sie den Bezeichner. Das aktuelle Wort wird überschrieben.

Hinweis

Wenn nur ein Bezeichner zur Auswahl steht, wird das Auswahlfenster nicht geöffnet und der Bezeichner sofort vervollständigt.

Funktionsweise

Es werden folgende Bezeichner angeboten, die mit dem vorgegebenen Zeichen beginnen:

- Schlüsselwörter der Sprache Structured Text
- Bezeichner aus der Befehlsbibliothek
- Technologieobjekte einschließlich deren Systemvariablen und Konfigurationsdaten
- Bezeichner der eigenen ST-Quelle:
  - Programmorganisationseinheiten (POE)
  - Datentypen
  - Variablen und Konstanten
  - Strukturelemente
- Bezeichner aus importierten Programmquellen

Hinweis

Bezeichner aus der eigenen ST-Quelle oder aus importierten Programmquellen werden nur korrekt angezeigt, wenn die entsprechende Programmquelle übersetzt wurde.

Die Anzeige erfolgt kontextsensitiv, d. h. es werden nur die Arten der Bezeichner angeboten, die an der jeweiligen Stelle der ST-Quelle sinnvoll sind:

- innerhalb eines Deklarationsblocks nur Datentypen und Schlüsselwörter
- innerhalb einer Programmorganisationseinheit (POE) keine Datentypen
- bei einer Struktur (z. B. var_struct.xx) nur Strukturkomponenten

3.3.4.13 Aufgerufenen Baustein öffnen

Vom ST-Editor aus können Sie eine in der ST-Quelle aufgerufene anwenderdefinierte Programmorganisationseinheit (POE) direkt öffnen und editieren, z. B eine Funktion oder einen Funktionsbaustein.

1. Setzen Sie den Cursor in den Bezeichner einer Programmorganisationseinheit (POE), z. B beim Aufruf einer Funktion oder bei der Instanzdeklaration eines Funktionsbausteins.
2. Wählen Sie das Kontextmenü **Aufgerufenen Baustein öffnen**.

Das Verhalten ist abhängig von der Programmiersprache, in der die Quelle der aufgerufenen POE verfasst ist:

- Programmiersprache SIMOTION ST:
  Die entsprechende ST-Quelle wird gegebenenfalls geöffnet. Der Cursor wird an den Anfang der entsprechenden POE im Implementationsabschnitt gesetzt.
  Innerhalb derselben ST-Quelle springt der Cursor zum Anfang der POE im Implementationsabschnitt.
- Programmiersprachen SIMOTION MCC oder SIMOTION KOP/FUP
  Die entsprechende POE (MCC-Chart, KOP/FUP-Programm) wird im zugehörigen Editor geöffnet.

3.3.4.14 Weitere Hilfsmittel des ST-Editors

Klammernpaare anzeigen

Die beiden Klammern eines Klammernpaars können optisch hervorgehoben werden.

Stellen Sie hierzu den Cursor neben eine Klammer. Der Editor versucht, die zugehörige Klammer des Paars zu finden und stellt gegebenenfalls beide Klammern hervorgehoben dar. Dies erleichtert Ihnen das Erkennen von Klammerpaaren, insbesondere bei Verschachtelungen.

So schalten Sie diese Funktion ein oder aus:

- Für die aktive ST-Quelle:
  - Wählen Sie das Kontextmenü **Ansicht > Klammernpaare anzeigen**.

  Diese Einstellung wird beim Schließen der ST-Quelle nicht gespeichert.
- Für alle geöffneten ST-Quellen:
  - Aktivieren oder deaktivieren Sie die Checkbox **Klammernpaare anzeigen** in den Einstellungen des ST-Editors (Seite 35).

  Diese Einstellung gilt auch für alle ST-Quellen, die anschließend geöffnet werden.

Zeilennummerierung ein- und ausblenden

Im ST-Editor können die Zeilennummern angezeigt werden:

So schalten Sie diese Funktion ein oder aus:

- Für die aktive ST-Quelle:
  - Wählen Sie das Kontextmenü **Ansicht > Zeilennummerierung**.

  Diese Einstellung wird beim Schließen der ST-Quelle nicht gespeichert.
- Für alle geöffneten ST-Quellen:
  - Aktivieren oder deaktivieren Sie die Checkbox **Zeilennummerierung anzeigen** in den Einstellungen des ST-Editors (Seite 35).

  Diese Einstellung gilt auch für alle ST-Quellen, die anschließend geöffnet werden.

Groß- und Kleinschreibung ändern

Sie können die Schreibweise eines markierten Textes in Großschreibung oder Kleinschreibung ändern:

1. Markieren Sie den Text, dessen Schreibweise Sie ändern wollen, siehe Text markieren.
2. Wählen Sie das Kontextmenü **Selektion in Großschreibung** bzw. **Selektion in Kleinschreibung**.

Tooltip-Anzeige

Der ST-Editor unterstützt Sie bei einigen Bezeichnern durch die Anzeige von Tooltips. Diese werden eingeblendet, wenn Sie mit dem Mauszeiger kurze Zeit über dem Bezeichner verweilen.

Beispielsweise wird bei Bezeichnerm von Variablen der zugehörige Datentyp im Tooltip angezeigt. Weitere Tooltips sind in Vorbereitung.

3.3.4.15 Befehlsbibliothek verwenden

Die Befehlsbibliothek ist ein Register des Projektnavigators. Sie enthält die verfügbaren Systemfunktionen, Systemfunktionsbausteine und Operatoren.

Diese Elemente können Sie mit Drag&Drop aus der Befehlsbibliothek in das Fenster des ST-Editors ziehen.

3.3.4.16 Funktionsleiste ST-Editor

Diese Funktionsleiste enthält wichtige Bedienhandlungen zum Programmieren:

Tabelle 3-3 Funktionsleiste des ST-Editors

Symbol	Bedeutung
	Übernehmen und übersetzen Klicken Sie auf dieses Symbol, um die aktive ST-Quelle in das Projekt zu übernehmen und in ausführbaren Code zu übersetzen. Siehe: Compiler starten (Seite 63).
	ST-Quelle einfügen Klicken Sie auf dieses Symbol, um eine neue ST-Quelle zu erstellen. Die Schaltfläche ist nur aktiv, wenn im Projektnavigator der Ordner **PROGRAMME** markiert ist, in dem die ST-Quelle abgelegt werden soll. Siehe: ST-Quelle einfügen (Seite 29).

Symbol	Bedeutung
	Status Programm Klicken Sie auf dieses Symbol, um den Testmodus **Status Programm** zu starten. Während des Programmdurchlaufs können Sie die Werte der in der ST-Quelle markierten Variablen beobachten. Folgende Voraussetzungen sind nötig: 1. Das Programm muss hierzu mit der entsprechenden Compileroption übersetzt sein. 2. Das Projekt und das Programm müssen in das Zielsystem geladen sein. 3. Es muss eine ONLINE-Verbindung zum Zielsystem existieren. Klicken Sie erneut auf dieses Symbol, um **Status Programm** zu beenden. Siehe: Status Programm einsetzen (Seite 342).
	Beobachten der Programmvariablen anhalten Klicken Sie auf dieses Symbol, um im Testmodus **Status Programm** das Beobachten der Programmvariablen anzuhalten. Siehe Status Programm einsetzen (Seite 342).
	Beobachten der Programmvariablen fortfahren Klicken Sie auf dieses Symbol, um im Testmodus **Status Programm** das Beobachten der Programmvariablen fortzusetzen. Siehe: Status Programm einsetzen (Seite 342).
	Aktualisieren Klicken Sie auf dieses Symbol, um in Testmodus **Status Programm** die Aktualisierung der angezeigten Werte zu erzwingen. Das Beobachten der Programmvariablen muss eingeschaltet sein. Siehe: Status Programm einsetzen (Seite 342).
	Aktuelle Auswahl formatieren Klicken Sie auf dieses Symbol, um im markierten Textbereich die Blöcke gemäß ihrer Hierarchie um jeweils eine Tabulatorweite einzurücken. Siehe Einzüge und Tabulatoren (Seite 37).
	Formatierungssymbole an/aus Klicken Sie auf dieses Symbol, um in der aktiven ST-Quelle Leerzeichen und Tabulatoren ein- oder ausblenden. Siehe Leerzeichen und Tabulatoren anzeigen (Seite 44).
	Lesezeichen einfügen/entfernen Klicken Sie auf dieses Symbol, um in der aktuellen Zeile der aktiven ST-Quelle ein Lesezeichen zu setzen bzw. ein vorhandenes Lesezeichen zu löschen. Siehe: Lesezeichen verwenden (Seite 51).
	Nächstes Lesezeichen Klicken Sie auf dieses Symbol, um zum nächsten Lesezeichen in der aktiven ST-Quelle zu springen. Siehe: Lesezeichen verwenden (Seite 51).
	Vorheriges Lesezeichen Klicken Sie auf dieses Symbol, um zum vorherigen Lesezeichen in der aktiven ST-Quelle zu springen. Siehe: Lesezeichen verwenden (Seite 51).

Symbol	Bedeutung
	Alle Lesezeichen entfernen Klicken Sie auf dieses Symbol, um alle Lesezeichen aus der aktiven ST-Quelle zu entfernen. Siehe: Lesezeichen verwenden (Seite 51).
	Gehe zu Blockanfang Klicken Sie auf dieses Symbol, um den Cursor an den Beginn des aktuellen bzw. übergeordneten Blocks zu verschieben.
	Gehe zu Blockende Klicken Sie auf dieses Symbol, um den Cursor an das Ende des aktuellen Blocks zu verschieben.

3.3.4.17 Kontextmenü ST-Editor

Das Kontextmenü ST-Editor wird eingeblendet, wenn Sie in einer geöffneten ST-Quelle die rechte Maustaste drücken.

Abhängig von der aktiven Applikation/Editor oder vom Modus (ONLINE/OFFLINE) werden bestimmte Befehle nicht angezeigt oder können nicht gewählt werden.

Dieses Kontextmenü ist, soweit es Editorfunktionen betrifft, auch für den Skript-Editor gültig.

Folgende Funktionen können Sie wählen:

Tabelle 3-4 Kontextmenü ST-Editor und Skript-Editor

Funktion	Bedeutung/Hinweis
Aufgerufenen Baustein öffnen [1]	Wenn der Cursor im Bezeichner einer anwenderdefinierten Programmorganisationseinheit (POE) steht, z. B. beim Aufruf einer Funktion oder bei der Instanzdeklaration eines Funktionsbausteins): Wählen Sie diesen Befehl, um die Quelle dieser POE (ST-Quelle, MCC-Chart, KOP/FUP-Programm) zu öffnen und zu editieren. Siehe: Aufgerufenen Baustein öffnen (Seite 53).
Ausschneiden	Wählen Sie diesen Befehl, um den markierten Bereich in der ST-Quelle auszuschneiden und in der Zwischenablage zu speichern.
Kopieren	Wählen Sie diesen Befehl, um den markierten Bereich in die Zwischenablage zu kopieren.
Einfügen	Wählen Sie diesen Befehl, um den Inhalt der Zwischenablage an der aktuellen Cursorposition in die ST-Quelle einzufügen.
Löschen	Wählen Sie diesen Befehl, um den markierten Bereich bzw. das Zeichen rechts vom Cursor zu löschen.
Alles markieren	Wählen Sie diesen Befehl, um den gesamten Text in der ST-Quelle zu markieren. Siehe: Text markieren (Seite 46).
Aufruf parametrieren	In Vorbereitung.
Ansicht	

Funktion		Bedeutung/Hinweis
	Zeilennummerierung an/aus	Wählen Sie diesen Befehl, um in der aktiven ST-Quelle die Zeilennummern ein- oder ausblenden. Siehe: Weitere Hilfsmittel des ST-Editors (Seite 54).
	Einrückhilfe	Wählen Sie diesen Befehl, um in der aktiven ST-Quelle die Ein- und Ausrückungen bei Blöcken durch senkrechte Hilfslinien (entsprechend der eingestellten Tabulatorweite) optisch hervorzuheben. Siehe: Einzüge und Tabulatoren (Seite 37).
	Klammernpaare anzeigen	Wählen Sie diesen Befehl, um in der aktiven ST-Quelle die beiden Klammern eines Klammernpaares optisch hervorzuheben, wenn der Cursor an einer der beiden Klammern steht. Siehe: Weitere Hilfsmittel des ST-Editors (Seite 54).
	Formatierungssymbole	Wählen Sie diesen Befehl, um in der aktiven ST-Quelle Leerzeichen und Tabulatoren ein- oder ausblenden. Siehe: Leerzeichen und Tabulatoren anzeigen (Seite 44).
	Folding	Wählen Sie diesen Befehl, um in der aktiven ST-Quelle das Falten zu aktivieren oder zu deaktivieren. Die Faltungsinformation in der aktiven ST-Quelle werden dabei ein- oder ausgeblendet. Siehe: Falten (Blöcke ein- und ausblenden) (Seite 40).
	Aktuelle Auswahl formatieren	Wählen Sie diesen Befehl, um im markierten Textbereich die Blöcke gemäß ihrer Hierarchie um jeweils eine Tabulatorweite einzurücken. Siehe: Einzüge und Tabulatoren (Seite 37).
	Fenster teilen	Wählen Sie diesen Befehl, um das aktive Fenster des ST-Editors horizontal in zwei Segmente zu teilen. Dies ermöglicht zwei Sichten auf dieselbe ST-Quelle. Siehe: Editorfenster teilen (Seite 43).
Lesezeichen		
	Lesezeichen einfügen/entfernen	Wählen Sie diesen Befehl, um in der aktuellen Zeile der aktiven ST-Quelle ein Lesezeichen zu setzen bzw. ein dort gesetztes Lesezeichen zu löschen. Siehe: Lesezeichen verwenden (Seite 51).
	Nächstes Lesezeichen	Wählen Sie diesen Befehl, um zum nächsten Lesezeichen in der aktiven ST-Quelle zu springen. Siehe: Lesezeichen verwenden (Seite 51).
	Vorheriges Lesezeichen	Wählen Sie diesen Befehl, um zum vorherigen Lesezeichen in der aktiven ST-Quelle zu springen. Siehe: Lesezeichen verwenden (Seite 51).
	Alle Lesezeichen entfernen	Wählen Sie diesen Befehl, um alle Lesezeichen aus der aktiven ST-Quelle zu entfernen. Siehe: Lesezeichen verwenden (Seite 51).
Selektion in Großschreibung		Wählen Sie diesen Befehl, um den markierten Text in Großbuchstaben zu ändern. Siehe: Weitere Hilfsmittel des ST-Editors (Seite 54).
Selektion in Kleinschreibung		Wählen Sie diesen Befehl, um den markierten Text in Kleinbuchstaben zu ändern. Siehe: Weitere Hilfsmittel des ST-Editors (Seite 54).

Funktion		Bedeutung/Hinweis
Markierten Bereich einrücken		• Wenn kein Text markiert ist: Wählen Sie diesen Befehl, um den Text rechts vom Cursor zur nächsten Tabulatorposition zu verschieben. • Wenn Text in einer einzigen Zeile markiert ist: Wählen Sie diesen Befehl, um den markierten Text zu löschen und den nachfolgenden Text zur nächsten Tabulatorposition zu verschieben. • Wenn Text in mehreren Zeilen markiert ist: Wählen Sie diesen Befehl, um den markierten Bereich einzurücken (Seite 37). Gemäß den Einstellungen des ST-Editors (Seite 35) wird ein Tabulatorzeichen ($09) oder die entsprechende Anzahl Leerzeichen ($20) eingefügt.
Markierten Bereich rückgängig		• Wenn kein Text markiert ist: Wählen Sie diesen Befehl, um den Cursor an die vorhergehende Tabulatorposition zu verschieben. • Wenn Text in einer einzigen Zeile markiert ist: Wählen Sie diesen Befehl, um die Markierung aufzuheben und den Cursor an die vorhergehende Tabulatorposition zu verschieben. • Wenn Text in mehreren Zeilen markiert ist: Wählen Sie diesen Befehl, um den markierten Bereich auszurücken (Seite 37).
Gehe zu Blockanfang		Wählen Sie diesen Befehl, um den Cursor an den Beginn des aktuellen bzw. übergeordneten Blocks zu verschieben.
Gehe zu Blockende		Wählen Sie diesen Befehl, um den Cursor an das Ende des aktuellen Blocks zu verschieben.
Gehe zu Blockanfang Ebene 0		Wählen Sie diesen Befehl, um den Cursor an den Beginn des übergeordneten Blocks 1. Ordnung zu verschieben.
Gehe zu Blockanfang Ebene 1		Wählen Sie diesen Befehl, um den Cursor an den Beginn des übergeordneten Blocks 2. Ordnung zu verschieben.
Haltepunkt setzen/entfernen [1]		Wählen Sie diesen Befehl, um an der ausgewählten Codestelle einen Haltepunkt zu setzen bzw. einen vorhandenen Haltepunkt zu entfernen. Siehe: Haltepunkte setzen (Seite 351).
Haltepunkt aktivieren/deaktivieren [1]		Wählen Sie diesen Befehl, um den Haltepunkt an der ausgewählten Codestelle zu aktivieren bzw. zu deaktivieren. Siehe: Haltepunkte aktivieren (Seite 361).
In Watchtabelle aufnehmen [1]		
	Neue Watchtabelle	Wenn der Cursor innerhalb einer Variablen steht bzw. wenn die Variable markiert ist: Wählen Sie diesen Befehl, um eine neue Watchtabelle anzulegen und in diese die Variable aufzunehmen.
	(Name einer Watchtabelle)	Wenn der Cursor innerhalb einer Variablen steht bzw. wenn die Variable markiert ist: Wahlen sie diesen Befehl, um die Variable in die ausgewählte Watchtabelle aufzunehmen
Gehe zu [1]		

Funktion		Bedeutung/Hinweis
	Lokale Verwendung >>	Wenn der Cursor innerhalb einer Variablen steht bzw. wenn die Variable markiert ist: Wählen Sie diesen Befehl, um zur nächsten Verwendung der Variablen in der ST-Quelle zu springen.
	Lokale Verwendung <<	Wenn der Cursor innerhalb einer Variablen steht bzw. wenn die Variable markiert ist: Wählen Sie diesen Befehl, um zur vorherigen Verwendung der Variablen in der ST-Quelle zu springen.
	Deklarationsstelle	Wenn der Cursor innerhalb einer Variablen steht bzw. wenn die Variable markiert ist: Wählen Sie diesen Befehl, um zur Deklarationsstelle der Variablen zu springen. Wenn die Variable in einer importierten Quelle oder Bibliothek deklariert ist, wird diese geöffnet.
	Verwendungsstellen	Wenn der Cursor innerhalb einer Variablen steht bzw. wenn die Variable markiert ist: Wählen Sie diesen Befehl, um alle Verwendungsstellen der Variablen in der Detailanzeige aufzulisten.

[1] Nicht im Kontextmenü des Skript-Editors.

3.3.4.18 Tastenkürzel (Shortcuts)

Der ST-Editor stellt Ihnen auch Tastenkürzel (Shortcuts) zur Verfügung. Die Befehle sind z. T. auch über die Menüs **Bearbeiten** oder **ST-Editor** aufrufbar:

Die Tastenkürzel, die Editorfunktionen betreffen, sind auch für den Skript-Editor gültig.

Tabelle 3-5 Tastenkürzel (Shortcuts) ST-Editor und Skript-Editor

Tastenkürzel/Shortcut	Beschreibung
F2	Zum nächsten Lesezeichen springen.
F3	Weitersuchen (Menü **Bearbeiten > Weitersuchen**).
F9 [1]	Einen Haltepunkt setzen oder entfernen (Menü **Debug > Haltepunkt setzen / entfernen**).
F12 [1]	Einen gesetzten Haltepunkt aktivieren oder deaktivieren (Menü **Debug > Haltepunkt aktivieren / deaktivieren**).
BACK	Löschen des Zeichens links vom Cursor.
EINFG	Zwischen Eingefügemodus und Überschreibmodus wechseln.
ENTF	Löschen des markierten Bereichs bzw. des Zeichens rechts vom Cursor (Menü **Bearbeiten > Löschen**).
Pfeiltaste	Cursor verschieben.
POS1	Cursor an den Anfang der Zeile verschieben.
ENDE	Cursor an das Ende der Zeile verschieben.
BILD AUF	Bildlauf um eine Seite nach oben. Der Cursor wird nachgeführt.
BILD AB	Bildlauf um eine Seite nach unten. Der Cursor wird nachgeführt.

Tastenkürzel/Shortcut	Beschreibung
TAB	• Wenn kein Text markiert ist: Text rechts vom Cursor zur nächsten Tabulatorposition verschieben. • Wenn Text in einer einzigen Zeile markiert ist: Markierten Text löschen und den nachfolgenden Text zur nächsten Tabulatorposition verschieben. • Wenn Text in mehreren Zeilen markiert ist: Markierten Bereich einrücken. Gemäß den Einstellungen des ST-Editors wird ein Tabulatorzeichen ($09) oder die entsprechende Anzahl Leerzeichen ($20) eingefügt.
SHIFT+F2	Zum vorherigen Lesezeichen springen.
SHIFT+BACK	Löschen des Zeichens links vom Cursor.
SHIFT+EINFG	Einfügen des Inhalts der Zwischenablage (Menü **Bearbeiten > Einfügen**).
SHIFT+ENTF	Ausschneiden des markierten Bereichs (Menü **Bearbeiten > Ausschneiden**).
SHIFT+Pfeiltaste	Text markieren zeilenweise.
SHIFT+POS1	Text markieren bis zum Anfang der Zeile.
SHIFT+ENDE	Text markieren bis zum Ende der Zeile.
SHIFT+BILD AUF	Bildlauf um eine Seite nach oben. Text zeilenweise markieren bis zur neuen Position des Cursors.
SHIFT+BILD AB	Bildlauf um eine Seite nach unten. Text zeilenweise markieren bis zur neuen Position des Cursors.
SHIFT+TAB	• Wenn kein Text markiert ist: Zur vorhergehenden Tabulatorposition springen. • Wenn Text in einer einzigen Zeile markiert ist: Zur vorhergehenden Tabulatorposition springen. • Wenn Text in mehreren Zeilen markiert ist: Markierten Bereich ausrücken.
STRG+A	Gesamten Text markieren (Menü **Bearbeiten > Alles markieren**).
STRG+B [1]	ST Quelle übernehmen und übersetzen (Menü **ST Quelle > Übernehmen und Übersetzen**).
STRG+C	Kopieren des markierten Bereichs in die Zwischenablage (Menü **Bearbeiten > Kopieren**).
STRG+D	Duplizieren der aktuellen Zeile oder des markierten Bereichs.
STRG+F	Suchen nach Texten innerhalb der ST-Quelle (Menü **Bearbeiten > Suchen**). Wenn Text in einer einzigen Zeile markiert ist, wird dieser in die Suchmaske übernommen.
STRG+H	Ersetzen von Texten innerhalb der ST-Quelle (Menü **Bearbeiten > Ersetzen**).
STRG+J	Anzeige des nächsten Suchergebnisses bei der projektweiten Suche (Menü **Bearbeiten > Nächste Stelle anzeigen**).
STRG+L	Kopieren der aktuellen Zeile oder der markierten Zeilen in die Zwischenablage.
STRG+U	Markierten Text in Kleinbuchstaben ändern.
STRG+V	Einfügen des Inhalts der Zwischenablage (Menü **Bearbeiten > Einfügen**).
STRG+X	Ausschneiden des markierten Bereichs (Menü **Bearbeiten > Ausschneiden**).
STRG+Y	Wiederholen der letzten Aktion (Menü **Bearbeiten > Wiederholen**).
STRG+Z	Rückgängig machen der letzen Aktion (Menü **Bearbeiten > Rückgängig**).
STRG+Leerzeichen	Automatisches Vervollständigen.
STRG+F2	Lesezeichen setzen oder entfernen.
STRG+F4	Aktives Fenster schließen (z. B. Menü **ST-Quelle > Schließen**).

Tastenkürzel/Shortcut	Beschreibung
STRG+F5 [1]	Alle Haltepunkte (in allen Programmquellen) des SIMOTION Geräts entfernen (Menü **Debug > Alle Haltepunkte entfernen**).
STRG+F7 [1]	Ein- oder Ausschalten der Funktion Status Programm (Menü **ST-Quelle > Status Programm Ein/Aus**).
STRG+F8 [1]	Programmablauf am aktivierten Haltepunkt fortsetzen (Menü **Debug > Fortsetzen**).
STRG+BACK	Wort links vom Cursor löschen.
STRG+EINFG	Kopieren des markierten Bereichs in die Zwischenablage (Menü **Bearbeiten > Kopieren**).
STRG+ENTF	Wort rechts vom Cursor löschen.
STRG+Pfeiltaste links/rechts	Cursor um ein Wort nach links bzw. rechts verschieben.
STRG+Pfeiltaste auf/ab	Bildlauf um eine Zeile nach oben bzw. unten. Die Position des Cursors bleibt unverändert, solange er im Fenster sichtbar ist.
STRG+POS1	Cursor an den Anfang der ST-Quelle verschieben.
STRG+ENDE	Cursor an das Ende der ST-Quelle verschieben.
STRG+SHIFT+B	Klammernpaare in der aktuellen ST-Quelle hervorheben.
STRG+SHIFT+F	Suchen nach Texten innerhalb des Projekts (Menü **Bearbeiten > Suchen im Projekt**).
STRG+SHIFT+G	Ersetzen von Texten innerhalb des Projekts (Menü **Bearbeiten > Ersetzen im Projekt**).
STRG+SHIFT+H	Aufruf parametrieren. In Vorbereitung.
STRG+SHIFT+U	Markierten Text in Großbuchstaben ändern.
STRG+SHIFT+F2	Alle Lesezeichen aus der ST-Quelle entfernen.
STRG+SHIFT+F3	Fenster nebeneinander anordnen.
STRG+SHIFT+F5	Fenster untereinander anordnen.
STRG+SHIFT+F7	Im ausgewählten Bereich einfache Zahlenfolgen (Sequenz) generieren.
STRG+SHIFT+F8	Ausgewählten Bereich formatieren.
STRG+SHIFT+F9	Cursor an den Beginn des aktuellen bzw. übergeordneten Blocks verschieben.
STRG+SHIFT+F10	Cursor an das Ende des aktuellen Blocks verschieben.
STRG+SHIFT+F11	Cursor an den Beginn des übergeordneten Blocks 1. Ordnung verschieben.
STRG+SHIFT+F12	Cursor an den Beginn des übergeordneten Blocks 2. Ordnung verschieben.
STRG+SHIFT+BACK	Text links vom Cursor bis zum Anfang der Zeile löschen.
STRG+SHIFT+ENTF	Text rechts vom Cursor bis zum Ende der Zeile löschen.
STRG+SHIFT+Pfeiltaste links/rechts	Wort links bzw. rechts vom Cursor markieren.
STRG+SHIFT+POS1	Text zeilenweise markieren bis zum Anfang der ST-Quelle.
STRG+SHIFT+ENDE	Text zeilenweise markieren bis zum Ende der ST-Quelle.
STRG+ALT+C	Falten: Alle Blöcke der aktuellen ST-Quelle ausblenden.
STRG+ALT+D	Falten: Alle Blöcke der aktuellen ST-Quelle einblenden.
STRG+ALT+F	Falten: Faltungsinformation in der aktuellen ST-Quelle ein- oder ausblenden.
STRG+ALT+I	Einrückebene in der aktuellen ST-Quelle anzeigen.
STRG+ALT+L	Zeilennummern in der aktuellen ST-Quelle ein- oder ausblenden.
STRG+ALT+O [1]	Wenn der Cursor im Bezeichner einer Programmorganisationseinheit (POE) steht: Aufgerufenen Baustein öffnen, d. h. Programmquelle der POE öffnen und Cursor positionieren.
STRG+ALT+R	Falten: Alle untergeordneten Blöcke einblenden.
STRG+ALT+S	Fenster teilen oder Teilung aufheben (Menü **ST-Quelle > Fenster teilen**).
STRG+ALT+T	Falten: Block ein- oder ausblenden.
STRG+ALT+V	Falten: Alle untergeordneten Blöcke ausblenden.

Tastenkürzel/Shortcut	Beschreibung
STRG+ALT+W	Leerzeichen und Tabulatoren in der aktuellen ST-Quelle ein- oder ausblenden.
STRG+ADD (Ziffernblock)	Schriftgröße in der aktuellen ST-Quelle vergrößern.
STRG+MINUS (Ziffernblock)	Schriftgröße in der aktuellen ST-Quelle verkleinern.
STRG+DIV (Ziffernblock)	Schriftgröße in der aktuellen ST-Quelle 100 %.
ALT+SHIFT+L	Markierten Text in Großbuchstaben ändern.
ALT+SHIFT+U	Markierten Text in Kleinbuchstaben ändern.
ALT+SHIFT+Pfeiltaste	Text markieren spaltenweise.
ALT+SHIFT+POS1	Text spaltenweise markieren bis zum Anfang der Zeile.
ALT+SHIFT+ENDE	Text spaltenweise markieren bis zum Ende der Zeile.
ALT+SHIFT+BILD AUF	Bildlauf um eine Seite nach unten. Text spaltenweise markieren bis zur neuen Position des Cursors.
ALT+SHIFT+BILD AB	Bildlauf um eine Seite nach unten. Text spaltenweise markieren bis zur neuen Position des Cursors.

[1] Tastenkürzel nicht für den Skript-Editor gültig.

Tabelle 3-6 Kombinierte Tasten- und Mausaktionen ST-Editor und Skript-Editor

Tastatur	Maus	Beschreibung
	Einfachklick mit linker Taste in Text	Cursor setzen.
	Doppelklick mit linker Taste in Text	Wort markieren.
	Linke Taste drücken und Maus ziehen	Text markieren zeilenweise.
	Einfachklick mit linker Taste auf Zeilennummer	Zeile markieren.
	Mausrad drehen	Bildlauf vertikal. Cursor bleibt unverändert.
SHIFT	Einfachklick mit linker Taste in Text	Text markieren zeilenweise.
SHIFT	Mausrad drehen	Bildlauf horizontal.
STRG	Einfachklick mit linker Taste auf Zeilennummer	Gesamten Text markieren (Menü **Bearbeiten > Alles markieren**).
STRG	Einfachklick mit linker Taste in Lesezeichenspalte	Lesezeichen setzen oder löschen.
STRG	Mausrad drehen	Schriftgröße verändern.
ALT	Linke Taste drücken und Maus ziehen	Text markieren spaltenweise.
ALT+SHIFT	Einfachklick mit linker Taste in Text	Text markieren spaltenweise.

3.3.5 Compiler starten

Voraussetzung

Die ST-Quelle ist mit dem ST-Editor geöffnet.

Vorgehensweise

1. Klicken Sie in das Fenster mit dem ST-Editor. Das dynamische Menü ST-Quelle wird eingeblendet.
2. Wählen Sie Menü **ST-Quelle > Übernehmen und Übersetzen**.

Hinweis

Das Menü ST-Quelle ist dynamisch. Es wird nur eingeblendet, wenn das Fenster des ST-Editors aktiv ist.

Der Compiler überprüft die Syntax der ST-Quelle. Im Register "Ausgabe übersetzen/prüfen" der Detailanzeige werden die erfolgreiche Übersetzung des Quelltextes bzw. Compiler-Fehler angezeigt. Die Fehlerangabe umfasst: Name der ST-Quelle, Nummer der Zeile, in welcher der Fehler aufgetreten ist, Fehlernummer und Fehlerbeschreibung.

3.3.5.1 Hilfe zur Fehlerkorrektur

Sie werden bei der Fehlerkorrektur unterstützt:

- Doppelklicken Sie auf die Fehlermeldung im Register **Ausgabe übersetzen/prüfen** der Detailanzeige.

Die Schreibmarke wird in der betreffenden Zeile der ST-Quelle positioniert.

3.3.6 Einstellungen für den Compiler vornehmen

Sie können die Einstellungen des Compilers (Compileroptionen) folgendermaßen festlegen:

- global für das SIMOTION Projekt, gültig alle Programmiersprachen, siehe Globale Einstellungen des Compilers (Seite 64)
- lokal für eine einzelne ST-Quelle innerhalb des SIMOTION Projekts, siehe Lokale Einstellungen des Compilers (Seite 67)

3.3.6.1 Globale Einstellungen des Compilers

Die globalen Einstellungen sind innerhalb des SIMOTION Projekts für alle Programmiersprachen gültig.

Vorgehensweise

1. Wählen Sie das Menü **Extras > Einstellungen**.
2. Wählen Sie das Register **Compiler**.
3. Nehmen Sie die Einstellungen entsprechend der nachfolgenden Tabelle vor.
4. Bestätigen Sie mit **OK**.

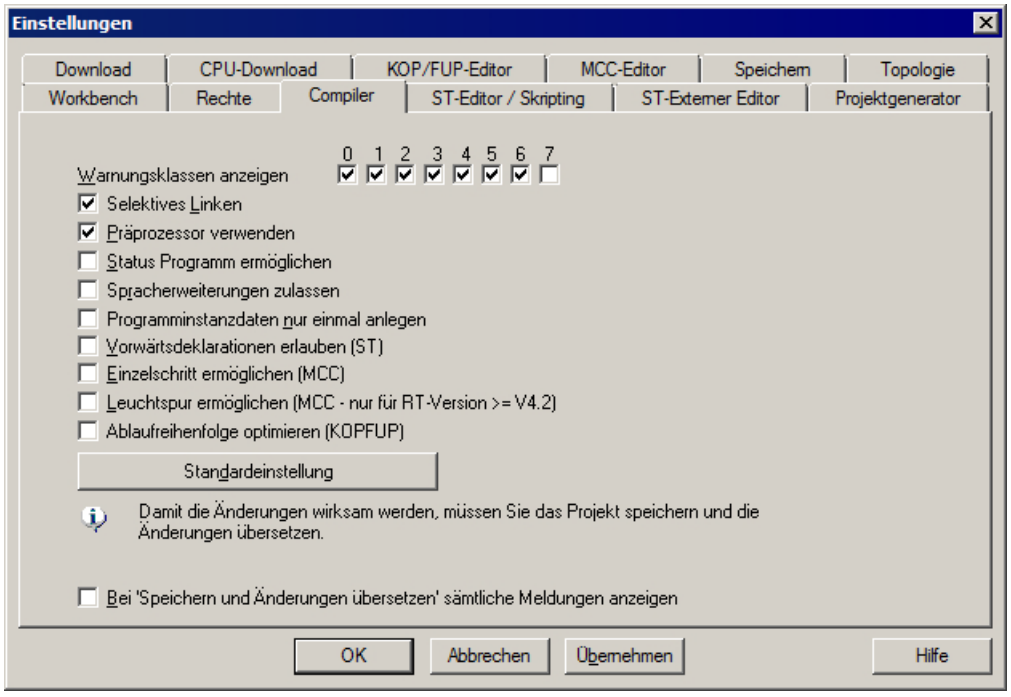


Bild 3-19 Globale Einstellungen des Compilers

Parameter

Tabelle 3-7 Parameter für globale Einstellungen des Compilers

Parameter	Beschreibung
Warnungsklassen anzeigen[1]	Zusätzlich zu den Fehlermeldungen kann der Compiler Warnungen und Informationen ausgeben. Sie können den Umfang der ausgegebenen Warnmeldungen einstellen: **Aktiv**: Der Compiler gibt Warnungen und Informationen der ausgewählten Klasse aus. **Inaktiv**: Der Compiler unterdrückt Warnungen und Informationen der jeweiligen Klasse. Siehe auch Bedeutung der Warnungsklassen (Seite 73).
Selektives Linken[1]	**Aktiv** (Standard): Unbenutzter Code wird aus dem lauffähigen Programm entfernt. **Inaktiv**: Unbenutzter Code bleibt im lauffähigen Programm erhalten..
Präprozessor verwenden[1]	**Aktiv**: Präprozessor wird verwendet (siehe Präprozessor steuern (Seite 307)). **Inaktiv** (Standard): Präprozessor wird nicht verwendet.
Status Programm ermöglichen[1]	**Aktiv**: Zusätzlicher Programmcode wird erzeugt, der die Beobachtung der Programmvariablen (auch lokaler Variablen) ermöglicht. **Inaktiv** (Standard): Status Programm nicht möglich. Siehe Eigenschaften von Status Programm (Seite 340).
Spracherweiterungen zulassen[1]	**Aktiv**: Es sind Sprachelemente zulässig, die nicht konform zur IEC 61131-3 sind. • Direkter Bitzugriff auf Variablen eines Bitdatentyps (Seite 152). • Außerhalb eines Funktionsbausteins auf dessen Eingangsparameter zugreifen (Seite 200). • Aufruf eines Programms innerhalb eines anderen Programms (Seite 206). **Inaktiv** (Standard): Nur Sprachelemente zulässig, die konform zur Norm IEC 61131-3 sind.

Parameter	Beschreibung
Programminstanzdaten nur einmal anlegen[1]	**Aktiv**: Die lokalen Variablen eines Programms werden nur einmal im Anwenderspeicher der Unit abgelegt. Diese Einstellung ist nötig beim Aufruf eines Programm innerhalb eines anderen Programms (Seite 206). **Inaktiv** (Standard): Die lokalen Variablen eines Programms werden entsprechend der Taskzuordnung im Anwenderspeicher der jeweiligen Task abgelegt. Siehe Speicherbereiche der Variablentypen (Seite 240).
Vorwärtsdeklarationen erlauben (ST)	Nur für Programmiersprache ST. Vorwärtsdeklarationen ermöglichen, dass Programmorganisationseinheiten (POE) vor ihrer vollständigen Definition verwendet werden können. Die Deklaration von Prototypen der POE vor deren Verwendung ist möglich, aber nur bei der Instanzdeklaration eines Funktionsbausteins erforderlich. **Aktiv**: Vorwärtsdeklarationen sind erlaubt. **Inaktiv** (Standard): Vorwärtsdeklarationen sind nicht erlaubt. Siehe Vorwärtsdeklarationen (Seite 318). Bei den Programmiersprachen MCC und KOP/FUP sind Vorwärtsdeklarationen immer erlaubt. Auch lokale Einstellung an der ST-Quelle (Seite 67) möglich. Siehe auch Beschreibung zur Wirksamkeit der globalen oder lokalen Einstellungen des Compilers (Seite 70).
Einzelschritt ermöglichen (MCC)	Nur für Programmiersprache MCC. **Aktiv**: Zusätzlicher Programmcode wird erzeugt, der die Beobachtung einzelner Programmschritte ermöglicht. **Inaktiv**: Einzelschritt nicht möglich. Diese Funktion erleichtert Ihnen die Fehlersuche (Debugging). Siehe "Einzelschritte im Programm verfolgen" im Programmierhandbuch MCC. Auch lokale Einstellung an der MCC-Quelle möglich. Siehe auch Beschreibung zur Wirksamkeit der globalen oder lokalen Einstellungen des Compilers (Seite 70).
Leuchtspur ermöglichen (MCC - nur für RT-Version >= 4.2)	Nur für Programmiersprache MCC und ab Version V4.2 des SIMOTION Kernels. **Aktiv**: Zusätzlicher Programmcode wird erzeugt, der die Beobachtung des Programmablaufs in zyklisch durchlaufenen Programmzweigen ermöglicht. **Inaktiv**: Leuchtspur nicht möglich. Diese Funktion erleichtert Ihnen die Fehlersuche (Debugging). Siehe "Programmablauf verfolgen per Leuchtspur" im Programmierhandbuch MCC. Auch lokale Einstellung an der MCC-Quelle möglich. Siehe auch Beschreibung zur Wirksamkeit der globalen oder lokalen Einstellungen des Compilers (Seite 70).
Ablaufreihenfolge optimieren (KOP/FUP)	Nur für Programmiersprache KOP/FUP. **Aktiv**: KOP/FUP-Netzwerke werden in der optimierten Ablaufreihenfolge berechnet. **Inaktiv**: KOP/FUP-Netzwerke werden in der nicht optimierten Ablaufreihenfolge berechnet.
Standardeinstellung	Klicken Sie auf diesen Button, um die Standardeinstellungen wiederherzustellen.
Bei "Speichern und Änderungen übersetzen" sämtliche Meldungen anzeigen[2]	Hier steuern Sie den Umfang des Meldungsprotokolls, das in der Detailanzeige der Workbench angezeigt wird, wenn Sie in SIMOTION SCOUT den Befehl **Speichern und Änderungen übersetzen** aufrufen. **Aktiv**: Ein ausführliches Meldungsprotokoll ähnlich der Einzelübersetzung einer ST-Quelle wird erstellt. **Inaktiv**: Ein komprimiertes Meldungsprotokoll wird erstellt.

[1] Auch lokale Einstellung möglich, siehe Lokale Einstellungen des Compilers (Seite 67). Siehe auch Beschreibung zur Wirksamkeit der globalen oder lokalen Einstellungen des Compilers (Seite 70).

[2] Benutzerspezifische Einstellung. Gültig für alle SIMOTION Projekte, welche der Benutzer bearbeitet.

Hinweis

Sie müssen gegebenenfalls das Projekt speichern und übersetzen, damit die Einstellungen wirksam werden.

3.3.6.2 Lokale Einstellungen des Compilers

Die lokalen Einstellungen werden für jede ST-Quelle einzeln vorgenommen und überschreiben die globalen Einstellungen.

Vorgehensweise

1. Öffnen Sie das Eigenschaftsfenster der ST-Quelle, siehe Eigenschaften einer ST-Quelle ändern (Seite 31):
   Markieren Sie im Projektnavigator die ST-Quelle und wählen Sie das Menü **Bearbeiten > Objekteigenschaften**.
2. Wählen Sie das Register **Compiler**.
3. Nehmen Sie die Einstellungen entsprechend der nachfolgenden Tabelle vor.
4. Bestätigen Sie mit **OK**.

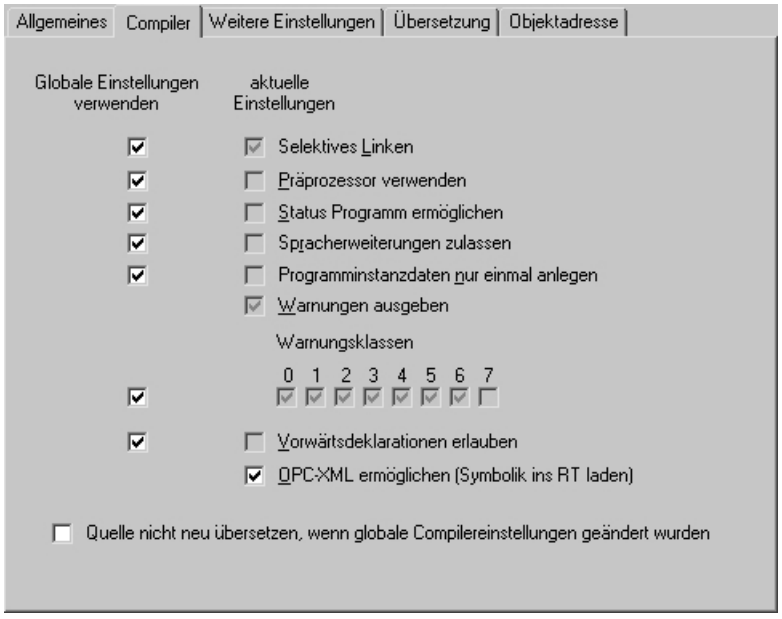


Bild 3-20 Lokale Einstellungen des Compilers an der ST-Quelle

Parameter

Tabelle 3-8 Parameter für lokale Einstellungen des Compilers an der ST-Quelle

Parameter	Beschreibung
Globale Einstellungen verwenden	Diese Checkbox ist für jeden Parameter verfügbar, zu dem es auch eine globale Einstellung gibt. Sie ist nur anwählbar, wenn die Checkbox " Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden" = Inaktiv ist. Hiermit legen Sie fest, ob die globalen Einstellungen übernommen werden oder die lokalen Einstellungen wirken. Siehe Beschreibung unter "Wirksamkeit der globalen oder lokalen Einstellungen (Seite 70)". Die lokalen Einstellungen legen Sie mit der zweiten Checkbox oder den weiteren Checkboxen zu den jeweiligen Parametern fest, die nachfolgend beschrieben sind.
Selektives Linken[1]	**Aktiv**: Unbenutzter Code wird aus dem lauffähigen Programm entfernt. **Inaktiv**: Unbenutzter Code bleibt im lauffähigen Programm erhalten. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv).
Präprozessor verwenden[1]	**Aktiv**: Präprozessor wird verwendet. **Inaktiv**: Präprozessor wird nicht verwendet. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv). Siehe Präprozessor steuern (Seite 307).
Status Programm ermöglichen[1]	**Aktiv**: Zusätzlicher Programmcode wird erzeugt, der die Beobachtung der Programmvariablen (auch lokaler Variablen) ermöglicht **Inaktiv**: Status Programm nicht möglich. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv). Siehe Eigenschaften von Status Programm (Seite 340).
Spracherweiterungen zulassen[1]	**Aktiv**: Es sind Sprachelemente zulässig, die nicht konform zur IEC 61131-3 sind. • Direkter Bitzugriff auf Variablen eines Bitdatentyps (Seite 152). • Außerhalb eines Funktionsbausteins auf dessen Eingangsparameter zugreifen (Seite 200). • Aufruf eines Programms innerhalb eines anderen Programms (Seite 206). **Inaktiv**: Nur Sprachelemente zulässig, die konform zur Norm IEC 61131-3 sind. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv).
Programminstanzdaten nur einmal anlegen[1]	**Aktiv**: Die lokalen Variablen eines Programms werden nur einmal im Anwenderspeicher der Unit abgelegt. Diese Einstellung ist nötig beim Aufruf eines Programm innerhalb eines anderen Programms (Seite 206). **Inaktiv**: Die lokalen Variablen eines Programms werden entsprechend der Taskzuordnung im Anwenderspeicher der jeweiligen Task abgelegt. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv). Siehe Speicherbereiche der Variablentypen (Seite 240). Weitere Informationen finden Sie im Funktionshandbuch SIMOTION Basisfunktionen.

Parameter	Beschreibung
Warnungen ausgeben Warnungsklassen[1]	Zusätzlich zu den Fehlermeldungen kann der Compiler Warnungen und Informationen ausgeben. Sie können den Umfang der ausgegebenen Warnmeldungen einstellen. Checkbox "Warnungen ausgeben": **Aktiv**: Der Compiler gibt Warnungen und Informationen entsprechend der nachstehenden Auswahl der Warnungsklassen aus. **Inaktiv**: Der Compiler unterdrückt alle Warnungen und Informationen zu dieser Quelle. Die Checkboxen für die Warnungsklassen werden ausgeblendet. **Grau hinterlegt** (nur Anzeige): Als globale Einstellung gibt der Compiler immer Warnungen und Informationen entsprechend der globalen Auswahl der Warnungsklassen aus, die nachfolgend angezeigt wird (bei "Globale Einstellungen verwenden" = Aktiv). Checkboxen "Warnungsklassen" (nur bei "Warnungen ausgeben" = Aktiv): **Aktiv**: Der Compiler gibt Warnungen und Informationen der ausgewählten Klasse aus. **Inaktiv**: Der Compiler unterdrückt Warnungen und Informationen der jeweiligen Klasse. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv). Siehe auch Bedeutung der Warnungsklassen (Seite 73).
Vorwärtsdeklarationen erlauben[1]	Vorwärtsdeklarationen ermöglichen, dass Programmorganisationseinheiten (POE) vor ihrer vollständigen Definition verwendet werden können. Die Deklaration von Prototypen der POE vor deren Verwendung ist möglich, aber nur bei der Instanzdeklaration eines Funktionsbausteins erforderlich. **Aktiv**: Vorwärtsdeklarationen sind erlaubt. **Inaktiv** (Standard): Vorwärtsdeklarationen sind nicht erlaubt. **Grau hinterlegt** (nur Anzeige): Die angezeigte globale Einstellung wird übernommen (bei "Globale Einstellungen verwenden" = Aktiv). Siehe Vorwärtsdeklarationen (Seite 318).
OPC-XML ermöglichen	**Aktiv:** Symbolinformationen der Unit-Variablen der ST-Quelle stehen im SIMOTION Gerät zur Verfügung. Dies ist notwendig für: • die Funktionen *_exportUnitDataSet* und *_importUnitDataSet*, siehe Funktionshandbuch SIMOTION Basisfunktionen. • die Watch-Funktion von IT DIAG. **Inaktiv:** Symbolinformationen werden nicht erzeugt.

Parameter	Beschreibung
Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden	**Aktiv:** Die globalen Einstellungen des Compilers sind für alle Parameter unwirksam. Die Checkboxen "Globale Einstellungen verwenden" sind nicht anwählbar und gegraut dargestellt. Bei Änderung der globalen Compilereinstellungen wird die ST-Quelle nicht neu übersetzt. Diese Einstellung ist nötig bei Bibliotheken, die mit einem Know-how-Schutz Programme (Seite 75) der Stufe "Hoch" versehen sind: • Für alle Programmquellen in dieser Bibliothek. • Für die Bibliothek selbst. **Inaktiv:** Die Checkboxen "Globale Einstellungen verwenden" sind für alle Parameter anwählbar und werden mit weißem Hintergrund dargestellt. Mit diesen Checkboxen legen Sie fest, ob für den entsprechenden Parameter die globalen Eigenschaften übernommen werden. Diese Einstellung bewirkt, dass im nachstehenden Fall die ST-Quelle übersetzt wird, obwohl alle Checkboxen "Globale Einstellungen übernehmen" inaktiv sind: • -Die globalen Einstellungen des Compilers wurden geändert und Menü **Projekt > Speichern und Änderungen übersetzen** wird gewählt. Siehe Beschreibung unter "Wirksamkeit der globalen oder lokalen Einstellungen (Seite 70)".
[1] Auch globale Einstellung möglich, siehe Globale Einstellungen des Compilers (Seite 64). Siehe auch Beschreibung zur Wirksamkeit der globalen oder lokalen Einstellungen des Compilers (Seite 70).	

3.3.6.3 Wirksamkeit der globalen oder lokalen Einstellungen des Compilers

Die Wirksamkeit der globalen und lokalen Einstellungen eines Parameters steuern Sie mit den lokalen Einstellungen des Compilers (Seite 67).

Checkbox "Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden"

Mit der Checkbox "Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden" legen Sie fest, ob die globalen Eigenschaften des Compilers Einfluss auf die Programmquelle haben.

- **Aktiv:** Die globalen Einstellungen des Compilers sind für alle Parameter unwirksam. Die Checkboxen "Globale Einstellungen verwenden" sind nicht anwählbar und gegraut dargestellt. Bei Änderung der globalen Compilereinstellungen wird die ST-Quelle nicht neu übersetzt.
  Diese Einstellung ist nötig bei Bibliotheken, die mit einem Know-how-Schutz Programme (Seite 75) der Stufe "Hoch" versehen sind:
  - Für alle Programmquellen in dieser Bibliothek.
  - Für die Bibliothek selbst.
- **Inaktiv:** Die Checkboxen "Globale Einstellungen verwenden" sind für alle Parameter anwählbar und werden mit weißem Hintergrund dargestellt. Mit diesen Checkboxen legen Sie fest, ob für den entsprechenden Parameter die globalen Eigenschaften übernommen werden.
  Diese Einstellung bewirkt, dass im nachstehenden Fall die Programmquelle übersetzt wird, obwohl alle Checkboxen "Globale Einstellungen übernehmen" inaktiv sind:
  - Die globalen Einstellungen des Compilers wurden geändert und Menü **Projekt > Speichern und Änderungen übersetzen** wird gewählt.

  Dies führt zu Problemen bei Bibliotheken, die mit einem Know-how-Schutz Programme (Seite 75) der Stufe "Hoch" versehen sind. Außerdem können bei Projekten, die von einer früheren Version des SIMOTION SCOUT konvertiert wurden, Online-Inkonsistenzen auftreten.

Checkboxen für die weiteren Parameter

Außerdem verfügt jeder Parameter, zu dem es auch eine globale Einstellung gibt, über mindestens 2 Checkboxen:

- Checkbox "Globale Einstellungen übernehmen":
  Mit dieser Checkbox legen Sie fest, ob die globalen Einstellungen übernommen werden. Sie ist nur anwählbar, wenn die Checkbox "Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden" = Inaktiv ist.
  Folgende Einstellungen sind möglich:
  - Aktiv:
    Für den entsprechenden Parameter werden die globalen Einstellungen übernommen. Die weiteren Checkboxen für diesen Parameter ist nicht anwählbar. Sie werden gegraut dargestellt und zeigen die globale Einstellung an.
  - Inaktiv:
    Für den entsprechenden Parameter werden die globalen Einstellungen ignoriert. Es sind nur die lokalen Einstellungen wirksam, die Sie mit den weiteren Checkboxen festlegen.
- Eine oder mehrere Checkboxen für die lokale Einstellung:
  Die Funktion dieser Checkboxen ist abhängig von der Checkbox "Globale Einstellungen übernehmen":
  - Bei aktiver Checkbox "Globale Einstellungen übernehmen":
    Für den entsprechenden Parameter werden die globalen Einstellungen übernommen. Die Checkboxen für die lokale Einstellung sind nicht anwählbar. Sie werden gegraut dargestellt und zeigen die globale Einstellung an.
  - Bei inaktiver Checkbox "Globale Einstellungen übernehmen":
    Die Checkboxen für die lokalen Einstellungen sind anwählbar und werden mit weißem Hintergrund dargestellt. Sie legen hiermit die lokalen Einstellungen für den entsprechenden Parameter fest. Die globalen Einstellungen werden ignoriert.

Dieses Verhalten gilt für folgende Compilereinstellungen

- Selektives Linken
- Präprozessor verwenden
- Status Programm ermöglichen
- Spracherweiterungen zulassen
- Programminstanzdaten nur einmal anlegen
- Warnungen anzeigen mit Warnungsklassen
- Vorwärtsdeklarationen erlauben (nur bei Programmiersprache ST)
- Einzelschritt ermöglichen (nur bei Programmiersprache MCC)
- Leuchtspur ermöglichen (nur bei Programmiersprache MCC und ab Version V4.2 des SIMOTION Kernels)

Hinweis

Die aktuellen Compileroptionen, die bei der nächsten Übersetzung der Programmquelle wirken, können Sie überprüfen:

- Wählen Sie das Register "Weitere Einstellungen" (Seite 73) im Eigenschaftsfenster der Programmquelle.

3.3.6.4 Bedeutung der Warnungsklassen

In der Tabelle sind die Warnungsklassen und ihre Bedeutung aufgelistet.

Tabelle 3-9 Bedeutung der Warnungsklassen

Warnungsklasse	Bedeutung
0	Warnungen zu nicht referenzierten oder verwendeten Codeteilen bzw. Daten
1	Warnungen zu verdeckten Bezeichnern
2	Warnungen bei Datentypkonvertierung, z. B. bei Datenänderung
3	Warnungen zu gesetzten Compileroptionen
4	Warnungen zu Semaphore (potentiell fehlerhafte Funktionen)
5	Warnungen zu Alarmfunktionen
6	Warnungen zu Konstrukten in Bibliotheken (Unit-Variablen deklariert)
7	Meldungen des Präprozessors

Bei der ausführlichen Beschreibung der Fehlermeldungen des Compilers ist angegeben, welche Warnungsklassen den einzelnen Warnungen (Seite 437) bzw. Informationen (Seite 443) zugeordnet sind.

3.3.6.5 Anzeige der Compileroptionen

Sie können für eine Programmquelle ansehen:

- Die aktuellen Compileroptionen gemäß den globalen oder lokalen Einstellungen des Compilers.
- Die beim letzten Übersetzen der Programmquelle verwendeten Compileroptionen.

Voraussetzung

Das Eigenschaftsfenster der Programmquelle (Seite 31) ist geöffnet.

Vorgehensweise

Zum Anzeigen der aktuellen Compileroptionen gemäß den globalen oder lokalen Einstellungen des Compilers (Seite 64):

- Wählen Sie das Register **Weitere Einstellungen**.
  Die aktuellen Compileroptionen für die Programmquelle werden angezeigt. Sie sind gültig für eine künftige Übersetzung.

Zum Anzeigen der beim letzten Übersetzen der Programmquelle verwendeten Compileroptionen:

- Wählen Sie das Register **Übersetzung**.
  Für die letzte Übersetzung der Programmquelle wird angezeigt:
  - Die Version des verwendeten Compilers.
  - Die verwendeten Compileroptionen.

Bedeutung der Compileroptionen

Compileroption	Bedeutung
-c[2]	Keine Erzeugung von Debug- und Symbolinformationen.
-C lang_ext	"Spracherweiterungen zulassen"[1] aktiv.
-C lang_iec	"Spracherweiterungen zulassen" inaktiv.
-C opcsym	"OPC-XML ermöglichen"[1] aktiv.
-C no_opcsym	"OPC-XML ermöglichen" inaktiv.
-C preproc	"Präprozessor verwenden"[1] aktiv.
-C no_preproc	"Präprozessor verwenden" inaktiv.
-C prog_once	"Programminstanzdaten nur einmal anlegen"[1] aktiv.
-C prog_multi	"Programminstanzdaten nur einmal anlegen" inaktiv.
-C scan_twice	"Vorwärtsdeklaration erlauben"[1] aktiv.
-C scan_once	"Vorwärtsdeklaration erlauben" inaktiv.
-D *text*	Präprozessor-Definition (Seite 75).
-e user[2]	Nur globale Einstellungen wirksam.
-e local	"Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden"[1] aktiv. Nur lokale Einstellungen wirksam. Keine Angabe (Voreinstellung): "Quelle nicht neu übersetzen, wenn globale Compilereinstellungen geändert wurden" inaktiv. Globale Einstellungen werden mit lokalen Einstellungen ergänzt.
-l[2]	Übernahme der Paketeinstellungen von Gerät bzw. Bibliothek.
-l sel	"Selektives Linken"[1] aktiv.
-l no_sel	"Selektives Linken" inaktiv.
-s	"Status Programm ermöglichen"[1] aktiv.
-s_off	"Status Programm ermöglichen" inaktiv.
-w no_warn	"Warnungen unterdrücken"[1] aktiv.
-w all_warn	Alle Warnungen anzeigen.
-w *n*_off	Warnungsklasse *n* aktiv[1].
-w *n*_on	Warnungsklasse *n* inaktiv[1].

Compileroption	Bedeutung
Weitere Optionen	Interne Optionen des SIMOTION Compilers.
[1] Zur Bedeutung der Compileroption: siehe "Lokale Einstellungen des Compilers" (Seite 67).	
[2] Nur bei Aufruf des Compilers über die Kommandozeile, z. B. mittels Scripting.	

Hinweis

Die Compileroptionen können auch beim Aufruf des Compilers über die Kommandozeile angegeben werden, z. B. mittels Scripting.

3.3.7 Know-how-Schutz für ST-Quellen

Sie können ST-Quellen gegen unbefugten Zugriff Dritter schützen. Geschützte ST-Quellen können Sie nur öffnen oder als Klartextdateien exportieren, wenn Sie das Kennwort angeben.

Weitere Informationen zum Know-how-Schutz Programme finden Sie in der SIMOTION Online-Hilfe.

Hinweis

Beim Export im XML-Format werden die ST-Quellen verschlüsselt exportiert. Beim Import der verschlüsselten XML-Dateien bleibt der Know-how-Schutz einschließlich Login und Passwort erhalten.

Siehe auch

Know-how-Schutz für Bibliotheken (Seite 288)

3.3.8 Präprozessor-Definitionen vornehmen

Definitionen für den Präprozessor (siehe Präprozessor steuern (Seite 307)) können Sie im Eigenschaftsdialog der ST-Quelle vornehmen. Sie können so den Präprozessor auch bei ST-Quellen mit Know-how-Schutz (siehe Know-how-Schutz für ST-Quellen (Seite 75)) steuern.

Präprozessor-Definitionen im Eigenschaftsdialog vornehmen

1. Öffnen Sie das Eigenschaftsfenster der ST-Quelle (siehe Eigenschaften einer ST-Quelle ändern (Seite 31)): Markieren Sie im Projektnavigator die ST-Quelle und wählen Sie das Menü **Bearbeiten > Objekteigenschaften**.
2. Wählen Sie das Register **Weitere Einstellungen**.
3. Geben Sie die Präprozessor-Definitionen ein (Syntax gemäß nachstehender Tabelle).
4. Bestätigen Sie mit **OK**.

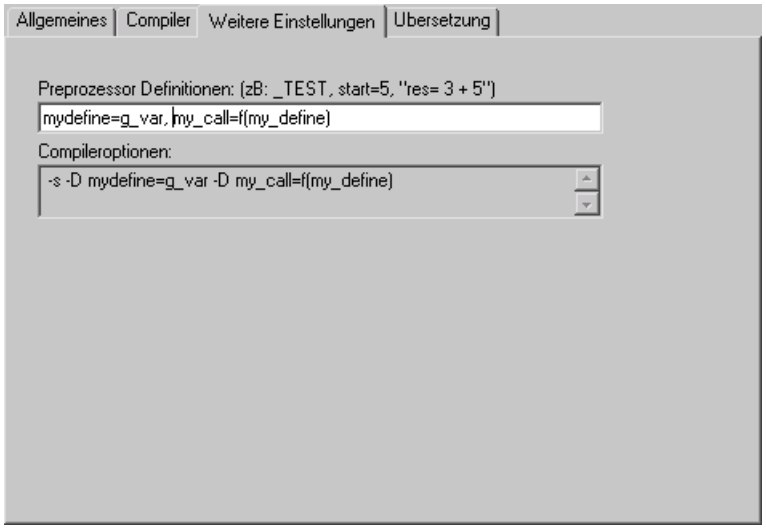


Bild 3-21 Präprozessor-Definitionen

Tabelle 3-10 Syntax der Präprozessor-Definitionen

Syntax	Bedeutung
Bezeichner=Text	Der angegebene *Bezeichner* wird definiert und in der ST-Quelle durch den angegebenen *Text* ersetzt. Erlaubte Zeichen: Siehe Fußteil der Tabelle. Wenn innerhalb des Ausdrucks Leerzeichen enthalten sind (z. B. im Text), muss die Syntax *"Bezeichner=Text"* verwendet werden
'Bezeichner=Text'	
"Bezeichner=Text"	
Bezeichner	Der angegebene *Bezeichner* wird definiert und in der ST-Quelle durch leeren Text ersetzt. Erlaubte Zeichen: Siehe Fußteil der Tabelle.
Mehrere Präprozessor-Definitionen werden durch Komma getrennt: *Definition_1, Definition_2, …* Erlaubte Zeichen: • für *Bezeichner*: Entsprechend den Regeln für Bezeichner: Folge aus Buchstaben (A … Z, a … z), Ziffern (0 … 9) oder einzelnen Unterstrichen (_) in beliebiger Reihenfolge, wobei das erste Zeichen ein Buchstabe oder ein Unterstrich sein muss. Zwischen Groß- und Kleinbuchstaben wird nicht unterschieden. • Für *Text*: Folge beliebiger Zeichen außer \ (Backslash), ' (einfaches Hochkomma - single quote), " (doppeltes Hochkomma - double quote). Die Schlüsselwörter USES, USELIB und USEPACKAGE sind nicht erlaubt.	

Hinweis

Präprozessor-Definitionen, die innerhalb einer ST-Quelle mit Pragmas vorgenommen werden, überschreiben die Definitionen im Eigenschaftsdialog.

Beachten Sie die Hinweise bei Präprozessor-Anweisung (Seite 308)!

3.3.9 ST-Quelle exportieren, importieren und drucken

Hier erhalten Sie einen Überblick über das Exportieren, Importieren und Drucken einer ST-Quelle.

3.3.9.1 ST-Quelle als Textdatei (ASCII) exportieren

Sie können eine ST-Quelle als Textdatei im ASCII-Format exportieren. Sie können diese Datei wieder als ST-Quelle importieren oder in jedem ASCII-Editor bearbeiten

Vorgehensweise

So exportieren Sie eine ST-Quelle als ASCII-Datei:

1. Öffnen Sie die ST-Quelle (Seite 31) , gegebenenfalls unter Angabe des Passworts (bei ST-Quellen, die mit Know-how-Schutz (Seite 75) versehen sind).
2. Stellen Sie sicher, dass sich der Cursor im ST-Editor befindet.
3. Wählen Sie den Menüpunkt **ST-Quelle > Exportieren**.
4. Geben Sie den Pfad und Dateinamen für die ASCII-Datei an und bestätigen Sie mit **Speichern**.

Die ST-Quelle wird als ASCII-Datei gespeichert; standardmäßig erhält der Dateiname die Erweiterung *.st.

Alternativ können Sie auch so vorgehen:

1. Markieren Sie im Projektnavigator die ST-Quelle.
2. Wählen Sie das Kontextmenü **Exportieren**.
3. Nur bei ST-Quellen, die mit Know-how-Schutz (Seite 75) versehen und nicht bereits geöffnet sind:
   Wenn das der ST-Quelle zugeordnete Login noch nicht angemeldet ist:
   - Geben Sie zum angezeigten Login das zugehörige Passwort ein.
     Der Know-how-Schutz wird für diese Quelle temporär (für diesen Export) aufgehoben.
   - Aktivieren Sie gegebenenfalls die Checkbox **Login als Standard-Login verwenden**.
     Sie werden mit diesem Login angemeldet und können weitere Quellen, denen dasselbe Login zugeordnet ist, ohne erneute Eingabe des Passworts exportieren oder öffnen.
4. Geben Sie den Pfad und Dateinamen für die ASCII-Datei an und bestätigen Sie mit **Speichern**.

Hinweis

Faltungsinformationen (Seite 40) und Lesezeichen (Seite 51) werden nicht exportiert.

Eine mit Know-how-Schutz versehene ST-Quelle wird ungeschützt exportiert.

3.3.9.2 ST-Quelle im XML-Format exportieren

So exportieren Sie eine ST-Quelle im XML-Format:

1. Markieren Sie im Projektnavigator die ST-Quelle.
2. Wählen Sie das Kontextmenü **Experte > Projekt speichern und Objekt exportieren**.
3. Geben Sie den Pfad für den XML-Export an und bestätigen Sie mit **OK**.

Unter dem angegebenen Pfad werden eine XML-Datei mit dem Namen der ST-Quelle und ein Ordner mit weiteren zugehörigen XML-Dateien gespeichert.

Hinweis

Auch mit Know-how-Schutz versehene ST-Quellen können im XML-Format exportiert werden. Die ST-Quellen werden verschlüsselt exportiert. Beim Import der verschlüsselten XML-Dateien bleibt der Know-how-Schutz einschließlich Login und Passwort erhalten.

Faltungsinformationen (Seite 40) und Lesezeichen (Seite 51) werden nicht exportiert.

3.3.9.3 Textdatei (ASCII) als ST-Quelle importieren

So importieren Sie eine ASCII-Datei als ST-Quelle:

1. Markieren Sie im Projektnavigator unter dem gewünschten SIMOTION Gerät den Ordner **PROGRAMME**.
2. Wählen Sie das Menü **Einfügen > Externe Quelle > ST-Quelle**.
3. Wählen Sie die zu importierende ASCII-Datei aus und bestätigen Sie mit **Öffnen**.
   Das Dialogfenster zum Einfügen einer ST-Quelle öffnet.
4. Geben Sie den Namen der ST-Quelle ein und wählen Sie die weiteren Optionen (siehe ST-Quelle einfügen (Seite 29)).

Die ASCII-Datei wird als ST-Quelle in das aktuelle Projektverzeichnis übernommen und kann von Ihnen geöffnet werden.

3.3.9.4 XML-Daten in ST-Quelle importieren

So importieren Sie XML-Daten in eine ST-Quelle:

1. Fügen Sie gegebenenfalls eine neue ST-Quelle ein (siehe ST-Quelle einfügen (Seite 29)).
2. Markieren Sie im Projektnavigator die ST-Quelle.
3. Wählen Sie das Kontextmenü **Experte > Objekt importieren**.
4. Wählen Sie die zu importierenden XML-Daten aus.
   Die importierten XML-Daten überschreiben vorhandene Daten in der markierten ST-Quelle.
   Das gesamte Projekt wird gespeichert und neu übersetzt.

Alternativ:

1. Markieren Sie im Projektnavigator den Ordner **PROGRAMME**.
2. Wählen Sie das Kontextmenü **Objekt importieren**.
3. Wählen Sie die zu importierenden XML-Daten aus.
   Eine neue ST-Quelle wird angelegt und die XML-Daten importiert. Diese ST-Quelle erhält den in den XML-Daten gespeicherten Namen; bei einem Namenskonflikt wird sie automatisch umbenannt. Das gesamte Projekt wird gespeichert und neu übersetzt.

Hinweis

Beachten Sie, wenn die zu importierenden XML-Daten aus einer ST-Quelle exportiert wurden, die mit Know-how-Schutz versehen war: Beim Import der verschlüsselten XML-Daten bleibt der Know-how-Schutz einschließlich Login und Passwort erhalten.

3.3.9.5 ST-Quelle drucken

So drucken Sie eine ST-Quelle:

1. Öffnen Sie die ST-Quelle.
2. Stellen Sie sicher, dass sich der Cursor im ST-Editor befindet.
3. Wählen Sie das Menü **Projekt > Drucken**.

Das Programm wird mit Angabe von Name und Datum gedruckt.

3.3.10 Externen Editor verwenden

Welche externen Editoren können verwendet werden?

Sie können statt des standardmäßigen ST-Editors jeden beliebigen ASCII-Editor verwenden, der über nachstehende Funktion verfügt:

- Externe Programme (z. B. Compiler) sind aufrufbar und wirken auf das aktive Fenster.

Außerdem sollte der Editor in der Lage sein, bestimmte Textstellen der ST-Quelle farblich hervorzuheben (Syntaxcoloring).

Hinweis

Bei Verwenden eines externen Editors stehen das dynamische Menü ST-Quelle und die darin enthaltenen Einträge nicht zur Verfügung. Auch die entsprechende Funktionsleiste ist inaktiv.

Das Übersetzen der ST-Quelle muss daher vom externen Editor aus gestartet werden können.

Status Programm (Seite 342) erfolgt weiterhin mit dem ST-Editor.

Einstellung zur Verwendung eines externen Editors

Die Einstellungen nehmen Sie in der SCOUT-Workbench vor:

1. Wählen Sie das Menü **Extras > Einstellungen**.
2. Wählen Sie das Register **ST-Externer Editor** (siehe Bild).
3. Markieren Sie die Checkbox **Externen ST-Editor verwenden**.
4. Geben Sie den Pfad des externen Editors an:
   – Klicken Sie auf **Durchsuchen** und wählen Sie Pfad und Dateinamen des Editors aus.

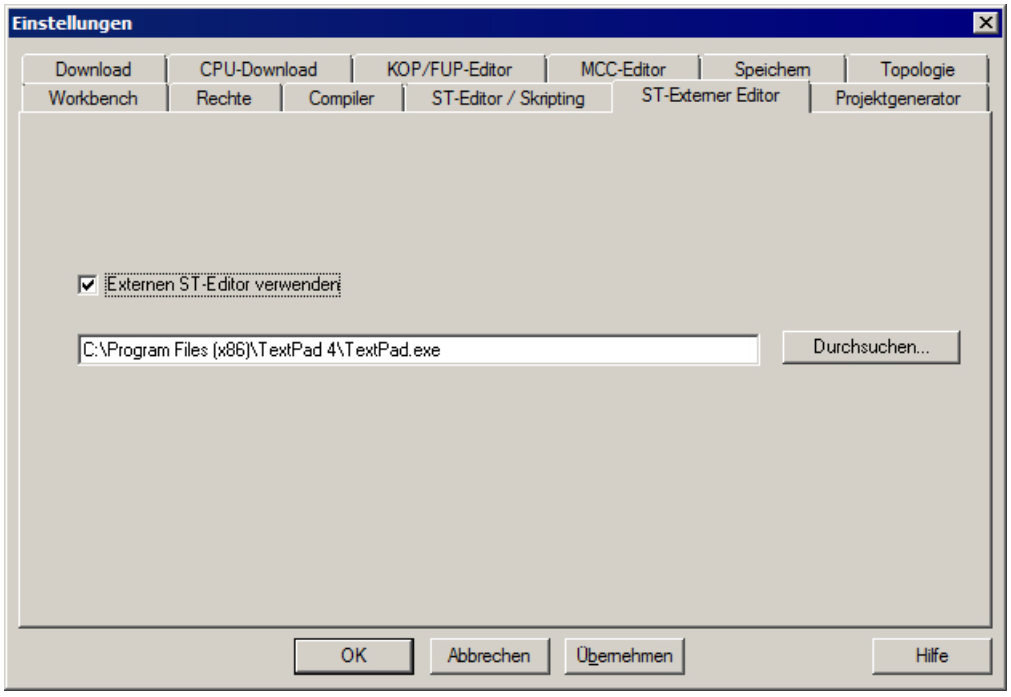


Bild 3-22 Einstellung zur Verwendung eines externen Editors

Einstellungen im externen Editor vornehmen

Die folgenden Hinweise sind allgemeiner Art. Vergleichen Sie die Bedienungsanleitung des externen Editors.

1. Konfigurieren Sie im externen Editor den Pfad zum ST-Compiler. Der Compiler befindet sich im STEP7-Installationsverzeichnis s7bin\u7wstcax.exe.
2. Für verschiedene Editoren werden Syntaxdateien mitgeliefert. Diese setzen den Editor in die Lage, Textstellen farbig hervorzuheben (Syntaxcoloring). Kopieren Sie die Syntaxdatei in das entsprechende Verzeichnis und konfigurieren Sie den Editor entsprechend.

Beachten Sie unbedingt Folgendes beim Verwenden eines externen Editors:

ACHTUNG
Datenverlust möglich Wenn Sie ein Projekt schließen oder SIMOTION SCOUT beenden, bevor Sie alle Fenster des externen Editord geschlossen haben, besteht die Gefahr von Datenverlusten! Schließen Sie alle Fenster des externen Editors, bevor Sie ein Projekt schließen oder SIMOTION SCOUT beenden.

3.3.11 ST-Quelle Menüs

3.3.11.1 Menü ST-Quelle

Abhängig von der aktiven Applikation/Editor oder vom Modus (ONLINE/OFFLINE) werden bestimmte Befehle nicht angezeigt oder können nicht gewählt werden. Das Menü wird nur angezeigt, wenn der ST-Editor im Arbeitsbereich aktiv ist.

Folgende Funktionen können Sie wählen:

Tabelle 3-11 Menü ST-Quelle

Funktion	Bedeutung/Hinweis
Schließen	Wählen Sie diesen Befehl, um die aktive ST-Quelle schließen. Bei Änderungen können entscheiden, ob Sie die geänderte Quelle in das Projekt übernehmen wollen.
Eigenschaften	Wählen Sie diesen Befehl, um die Eigenschaften der aktiven ST-Quelle anzuzeigen In mehreren Registern können Sie lokale Einstellungen für diese Quelle vornehmen. Siehe: Eigenschaften einer ST-Quelle ändern (Seite 31).
Übernehmen und Übersetzen	Wählen Sie diesen Befehl, um die aktuelle ST-Quelle in das Projekt zu übernehmen und in ausführbaren Code zu übersetzen. Siehe: Compiler starten (Seite 63).
Präprozessor ausführen	Der Präprozessor scannt optional eine ST-Quelle vor dem Kompilieren und kann darin z. B. Zeichenfolgen ersetzen, die dann beim Kompilieren berücksichtigt werden. Mit diesem Menübefehl können Sie die Präprozessor-Anweisungen gezielt ausführen.
Exportieren	Wählen Sie diesen Befehl, um die aktive ST-Quelle als Textdatei (ASCII) zu exportieren. Siehe: ST-Quelle als Textdatei (ASCII) exportieren (Seite 77).
Fenster teilen	Wählen Sie diesen Befehl, um das aktive Fenster des ST-Editors horizontal in zwei Segmente zu teilen. Dies ermöglicht zwei Sichten auf dieselbe ST-Quelle. Siehe: Editorfenster teilen (Seite 43).

Funktion	Bedeutung/Hinweis
Status Programm Ein/Aus	Wählen Sie diesen Befehl, um den Testmodus **Status Programm** zu starten. Während des Programmdurchlaufs können Sie die Werte der in der ST-Quelle markierten Variablen beobachten. Folgende Voraussetzungen sind nötig: 1. Das Programm muss hierzu mit der entsprechenden Compileroption übersetzt sein. 2. Das Projekt und das Programm müssen in das Zielsystem geladen sein. 3. Es muss eine ONLINE-Verbindung zum Zielsystem existieren. Wählen Sie den Befehl erneut, um **Status Programm** zu beenden. Siehe: Status Programm einsetzen (Seite 342).
Variablen sichern	Mit diesem Menübefehl können Sie Retain-, Unit- und Globale Variablen sichern. Sie können diese Variablen aus dem RAM oder ROM Speicher des Zielgeräts sichern und auf einen Datenträger als XML-Datei speichern. Beim Wiederherstellen können Sie diese Variablen aus dem Datenträger in den RAM bzw. ROM Speicher des Zielgeräts schreiben.
Variablen wiederherstellen	Mit diesem Menübefehl können Sie Retain-, Unit- und Globale Variablen aus den vorher exportierten Variablen wieder herstellen. Beim Wiederherstellen können Sie diese Variablen aus dem Datenträger in den RAM bzw. ROM Speicher des Zielgeräts schreiben.

3.3.11.2 Kontextmenü ST-Quelle

Das Kontextmenü ST-Quelle wird eingeblendet, wenn Sie im Projektnavigator eine ST-Quelle markieren und die rechte Maustaste drücken.

Abhängig von der aktiven Applikation/Editor oder vom Modus (ONLINE/OFFLINE) werden bestimmte Befehle nicht angezeigt oder können nicht gewählt werden.

Folgende Funktionen können Sie wählen:

Tabelle 3-12 Kontextmenü ST-Quelle

Funktion	Bedeutung/Hinweis
Öffnen	Wählen Sie diesen Befehl, um die markierte ST-Quelle öffnen. Siehe: Vorhandene ST-Quelle öffnen (Seite 31).
Ausschneiden	Die markierten ST-Quellen werden entfernt und in der Zwischenablage abgelegt.
Kopieren	Die markierten ST-Quellen werden in die Zwischenablage kopiert.
Einfügen	Der Inhalt der Zwischenablage wird in dem markierten Ordner eingefügt.
Löschen	Die markierte ST-Quelle wird einschließlich aller Daten gelöscht.
Umbenennen	Wählen Sie diesen Befehl, um den Namen der markierten ST-Quelle zu ändern. Beachten Sie bei Namensänderungen, dass Referenzen auf diesen Namen nicht mit geändert werden und der neue Name den Regeln für Bezeichner (Seite 100) entsprechen muss.
Variablen sichern	Mit diesem Menübefehl können Sie Retain-, Unit- und Globale Variablen sichern. Sie können diese Variablen aus dem RAM oder ROM Speicher des Zielgeräts sichern und auf einen Datenträger als XML-Datei speichern. Beim Wiederherstellen können Sie diese Variablen aus dem Datenträger in den RAM bzw. ROM Speicher des Zielgeräts schreiben.
Variablen wiederherstellen	Mit diesem Menübefehl können Sie Retain-, Unit- und Globale Variablen aus den vorher exportierten Variablen wieder herstellen. Beim Wiederherstellen können Sie diese Variablen aus dem Datenträger in den RAM bzw. ROM Speicher des Zielgeräts schreiben.

Funktion		Bedeutung/Hinweis
Experte		
	Objekt importieren	Wählen Sie diesen Befehl, um XML-Daten einer ST-Quelle, die Sie in einem anderen Projekt exportiert hatten, in die markierte ST-Quelle zu importieren. Die in der importierenden ST-Quelle vorhandenen Daten werden überschrieben. Siehe: XML-Daten in ST-Quelle importieren (Seite 78).
	Projekt speichern und Objekt exportieren	Wählen Sie diesen Befehl, um die markierte ST-Quelle im XML-Format zu exportieren. Die exportierten Daten können Sie in andere Projekte importieren. Siehe: ST-Quelle im XML-Format exportieren (Seite 78).
Übernehmen und Übersetzen		Wählen Sie diesen Befehl, um die aktuelle ST-Quelle in das Projekt zu übernehmen und in ausführbaren Code zu übersetzen. Siehe: Compiler starten (Seite 63).
Präprozessor ausführen		Der Präprozessor scannt optional eine ST-Quelle vor dem Kompilieren und kann darin z. B. Zeichenfolgen ersetzen, die dann beim Kompilieren berücksichtigt werden. Mit diesem Menübefehl können Sie die Präprozessor-Anweisungen gezielt ausführen.
Status Programm Ein/Aus		Wählen Sie diesen Befehl, um den Testmodus **Status Programm** zu starten. Während des Programmdurchlaufs können Sie die Werte der in der ST-Quelle markierten Variablen beobachten. Folgende Voraussetzungen sind nötig: 1. Das Programm muss hierzu mit der entsprechenden Compileroption übersetzt sein. 2. Das Projekt und das Programm müssen in das Zielsystem geladen sein. 3. Es muss eine ONLINE-Verbindung zum Zielsystem existieren. Wählen Sie den Befehl erneut, um **Status Programm** zu beenden. Siehe: Status Programm einsetzen (Seite 342).
Exportieren		Wählen Sie diesen Befehl, um die markierte ST-Quelle als Textdatei (ASCII) zu exportieren. Siehe: ST-Quelle als Textdatei (ASCII) exportieren (Seite 77).
Know-how-Schutz		
	Setzen	Wählen Sie diesen Befehl, um die markierte ST-Quelle gegen unbefugten Zugriff Dritter schützen. Geschützte ST-Quellen können Sie nur unter Angabe eines Passworts öffnen oder als Klartextdateien exportieren. Siehe: Know-how-Schutz für ST-Quellen (Seite 75).
	Löschen	Wählen Sie diesen Befehl, um den Know-how-Schutz der markierten Quelle permanent aufzuheben. Die Eingabe des Passworts ist hierfür erforderlich. Siehe: Know-how-Schutz für ST-Quellen (Seite 75).
Referenzdaten anzeigen		Für eine korrekte, konsistente Anzeige der Referenzdaten ist eine fehlerfreie Übersetzung Voraussetzung. Übersetzen Sie ggf. das Projekt, die CPU, das Programm oder die Bibliothek vorher. Siehe: Referenzdaten (Seite 298), Querverweisliste erzeugen (Seite 298).

Funktion		Bedeutung/Hinweis
	Querverweise	Die Querverweisliste der markierten ST-Quelle wird gebildet und angezeigt. Die Querverweisliste enthält die Deklaration und Verwendung aller Bezeichner der markierten ST-Quelle. Siehe: Inhalt der Querverweisliste (Seite 299).
	Programmstruktur	Die Programmstruktur der markierten ST-Quelle wird gebildet und angezeigt. Die Programmstruktur enthält alle Unterprogrammaufrufe und deren Schachtelung innerhalb der markierten ST-Quelle. Siehe: Inhalt der Programmstruktur (Seite 303).
	Codeattribute	Die Codeattribute der markierten ST-Quelle werden gebildet und angezeigt. Die Codeattribute enthalten Informationen über den Speicherbedarf verschiedener Datenbereiche der markierten ST-Quelle. Siehe: Inhalt der Codeattribute (Seite 304).
Drucken		Wählen Sie diesen Befehl, um die markierte ST-Quelle zu drucken. Sie können wählen, ob Sie den Text der ST-Quelle und/oder deren Eigenschaften drucken wollen.
Druckvorschau		Wählen Sie diesen Befehl, um eine Vorschau auf das zu erwartende Druckergebnis zu erhalten.
Eigenschaften		Wählen Sie diesen Befehl, um die Eigenschaften der ausgewählten ST-Quelle anzuzeigen In mehreren Registern können Sie lokale Einstellungen für diese Quelle vornehmen. Siehe: Eigenschaften einer ST-Quelle ändern (Seite 31).

3.4 Ein Beispielprogramm erstellen

Wir wollen ein kurzes Programm erstellen, das Ihnen beispielhaft alle Arbeitsschritte von der Erstellung bis zum Starten und Testen eines Programms aufzeigt. Das Testen ist in Programmtest (Seite 325) beschrieben.

Aufgabenstellung

Das Programm *Flash* setzt ein Bit an einem Ausgangsbyte Ihres Zielsystems und rotiert es in diesem Byte. Dadurch wird jedes Bit des Ausgangsbytes nacheinander gesetzt und wieder zurückgesetzt. Nach dem letzten Bit des Bytes soll wieder das erste Bit gesetzt werden. Das Ergebnis des Programms sehen Sie an Ausgängen Ihres Zielsystems.

3.4.1 Voraussetzungen

Um das Beispielprogramm zu erstellen benötigen Sie

- ein SIMOTION Projekt und
- innerhalb des Projekts ein SIMOTION Gerät (z. B. SIMOTION C240), dessen Ausgang auf Adresse 62 konfiguriert ist.

3.4.2 Projekt öffnen oder erstellen

Projekte enthalten alle Informationen über Hardware und Konfiguration. Darin sind auch Programme enthalten, mit denen Sie diese Hardware steuern.

Vorgehensweise

Ist kein Projekt vorhanden, gehen Sie wie folgt vor:

1. Wählen Sie in der Menüleiste **Projekt** an.
2. Wählen Sie **Neu** oder **Öffnen**.
3. Geben Sie einen Namen für ein neues Projekt an und bestätigen Sie mit **OK**.

Einzelheiten siehe in der Online-Hilfe.

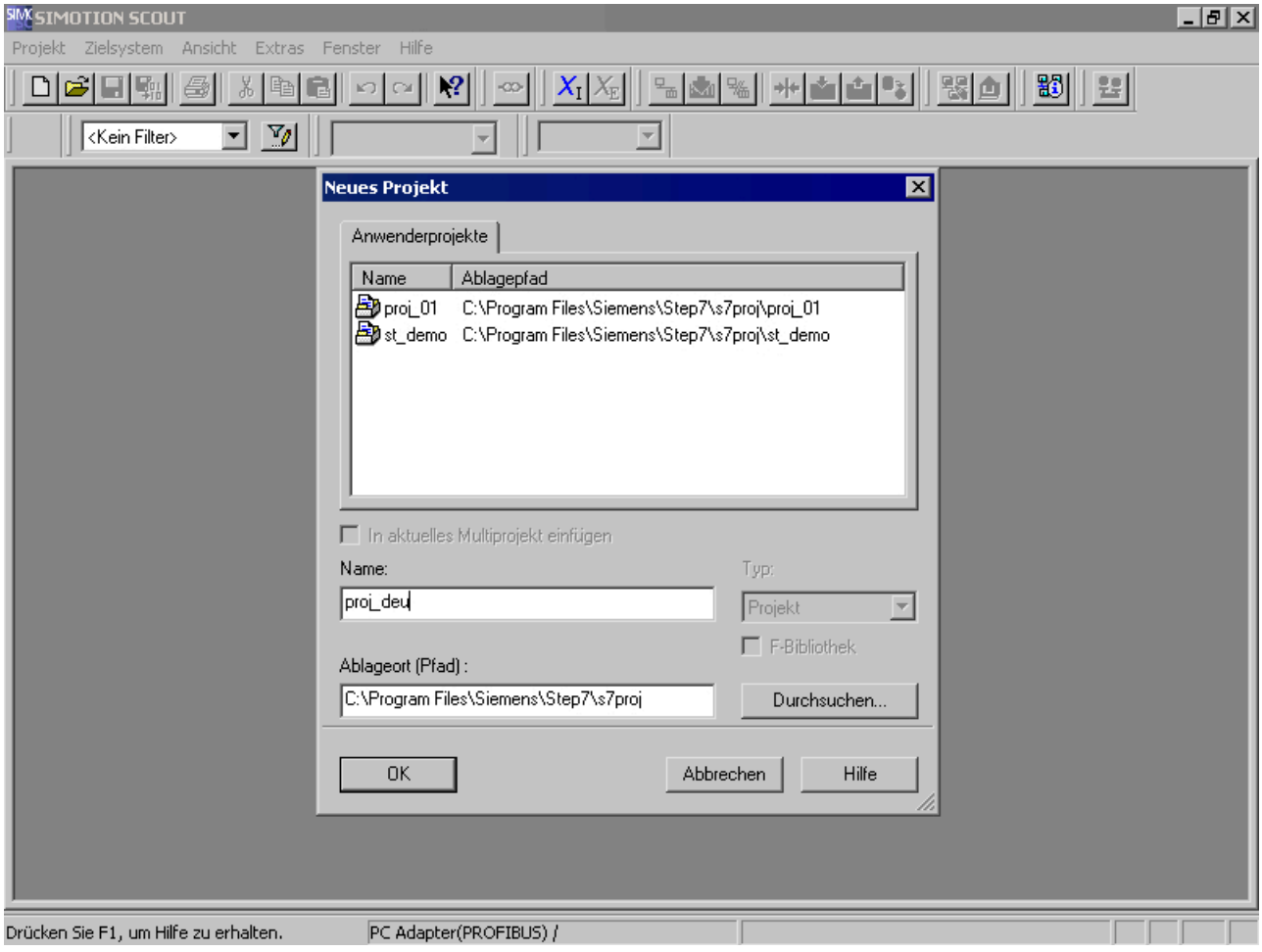


Bild 3-23 Neues Projekt erstellen

3.4.3 Hardware bekanntmachen

Dieser Arbeitsschritt umfasst:

1. Neues SIMOTION Gerät (z. B. C240 V4.2) anlegen und konfigurieren.
2. Ausgang in der HW-Konfig auf Adresse 62 konfigurieren.

Einzelheiten zu den Arbeitsschritten 1 und 2 siehe Online-Hilfe.

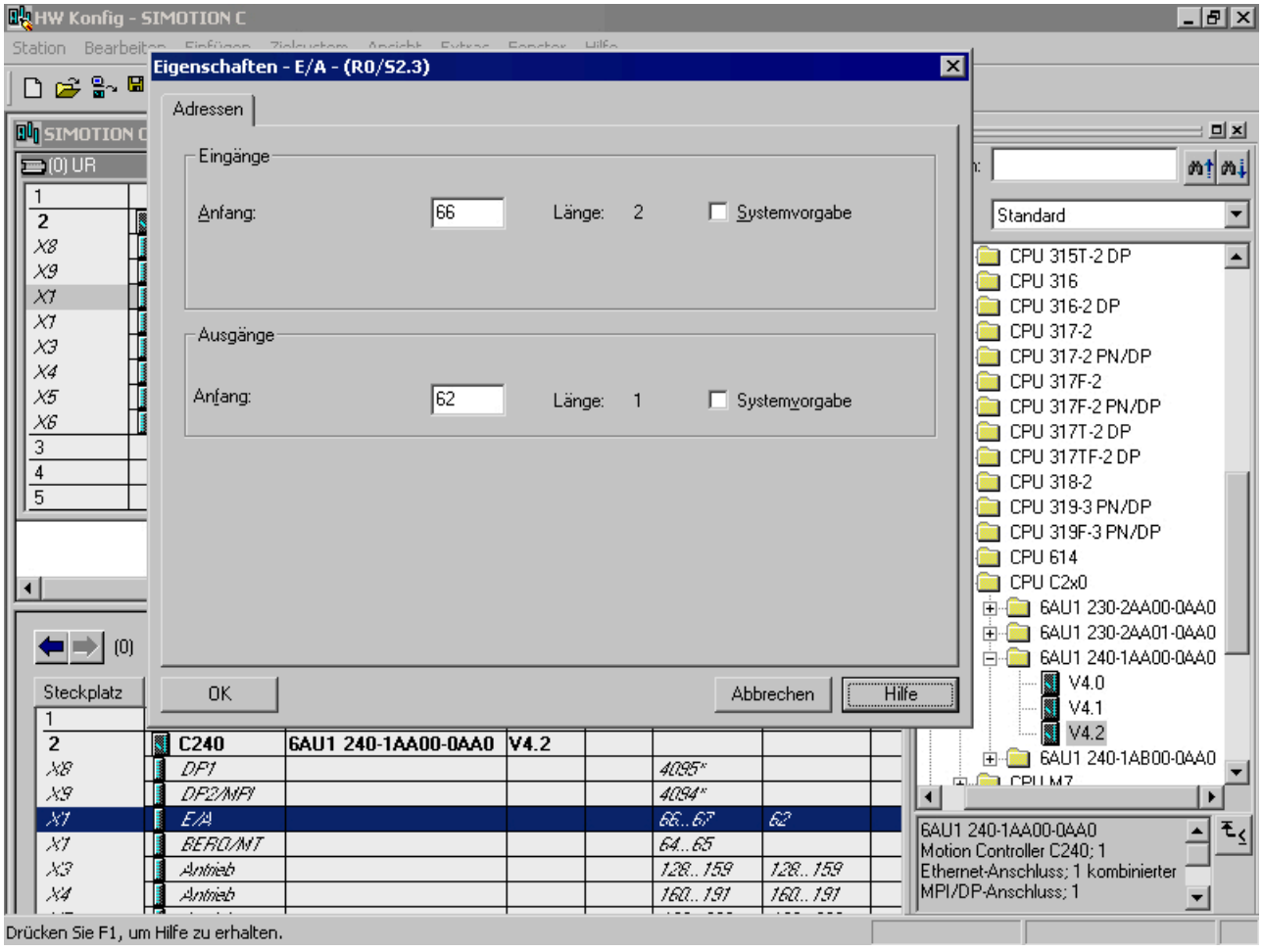


Bild 3-24 Änderung in der HW-Konfig

3.4.4 Quelltext mit dem ST-Editor eingeben

Vorgehensweise

1. Öffnen Sie im Projektnavigator den Baum Ihres SIMOTION Geräts (Programme sind dem SIMOTION Gerät zugeordnet, auf dem Sie später laufen sollen.).
2. Markieren Sie den Ordner **Programme** und wählen Sie den Menüpunkt **Einfügen > Programm > ST-Quelle**.
3. Geben Sie einen Namen mit maximal 128 Zeichen für die ST-Quelle ein (siehe Bild), z. B. **ST_1**, und bestätigen Sie Ihre Eingabe mit **OK**.
   Der ST-Editor erscheint im Arbeitsbereich. Im Navigator wird die ST-Quelle **ST_1** eingefügt.

4. Geben Sie den Quelltext aus Quelltext des Beispielprogramms (Seite 89) ein, am besten mit eingerückten Zeilen. Drücken Sie dazu die TAB-Taste.
   Die Möglichkeiten des ST-Editors finden Sie in Mit dem ST-Editor umgehen (Seite 33); der Aufbau einer ST-Quelle ist in Gliederung der ST-Quelle (Seite 113) und in Quelldatei-Abschnitte (Seite 212) ausführlich beschrieben.
5. Verwenden Sie so oft wie möglich Kommentare. Geben Sie Ihren Kommentar nach den Zeichen **//** ein, wenn es sich um einzeiligen Text handelt. Wenn der Kommentar über mehrere Zeilen geht, schließen Sie ihn zwischen den Zeichenpaaren (***** und *****) ein.
6. Speichern Sie das gesamte Projekt mit **Projekt > Speichern**.

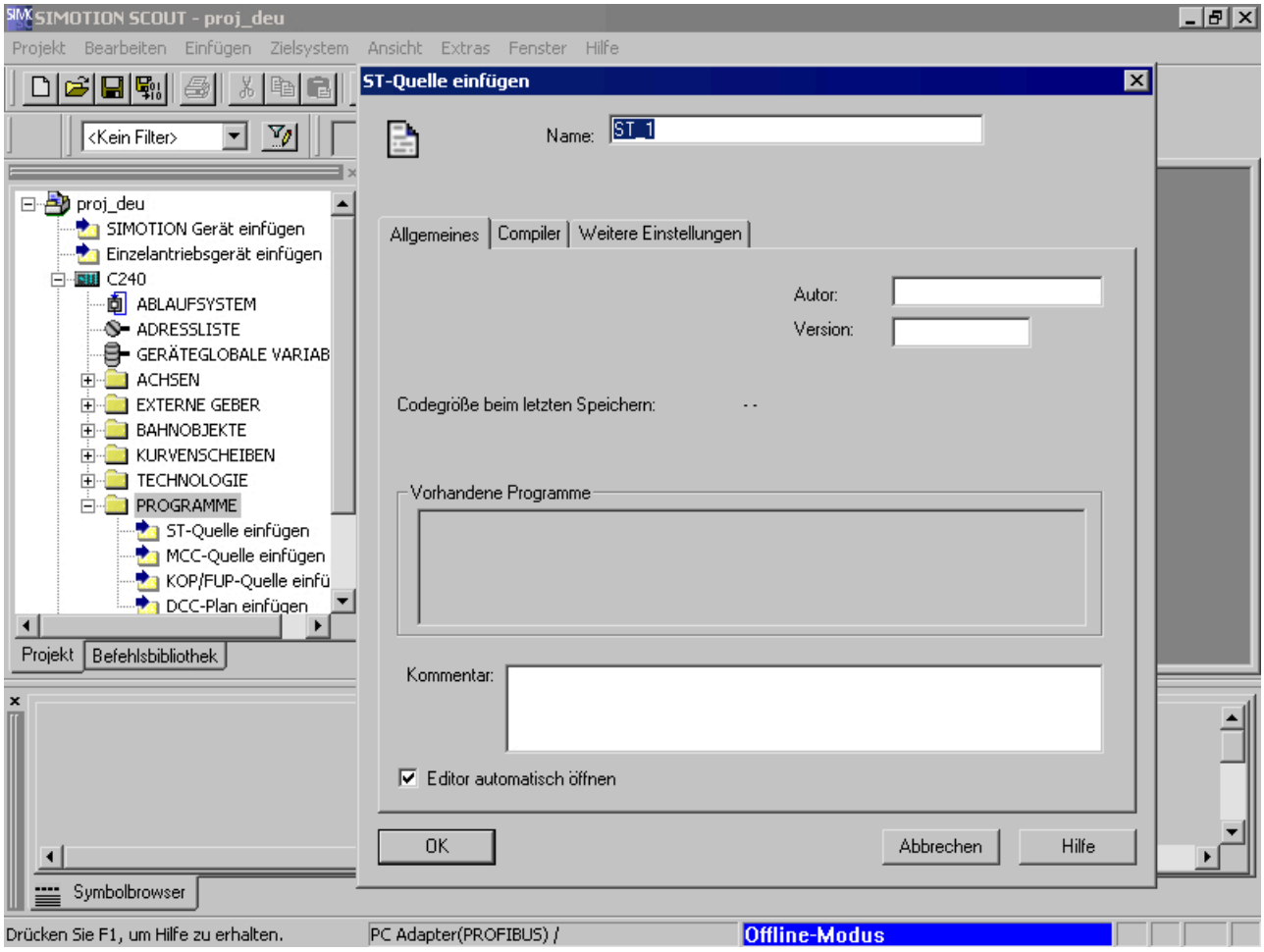


Bild 3-25 Name für die ST-Quelle vergeben

3.4.4.1 Funktionen des Editors

Der ST-Editor stellt Ihnen außer einfacher Texteingabe folgende Zusatz- bzw. Komfortfunktionen zur Verfügung, unter anderem, um Ihnen die Funktionalität Ihres Quelltextes zu dokumentieren:

- In Windows übliche Bedienungsmöglichkeiten (z. B. Rückgängig mit CTRL + Z oder Wiederholen mit CTRL + Y)
- Syntaxcoloring (unterschiedliche Färbung für die verschiedenen Elemente der Sprache)

- Drucken der Quelldatei in ansprechender Form mit Seitennummer, Quelldateiname und Druckdatum
- Export/Import der Quelldatei
- Archivieren der Quelldatei (über das Projekt)

Eine ausführliche Beschreibung der Funktionen finden Sie in Mit dem ST-Editor umgehen (Seite 33) und in Einstellungen für den Compiler vornehmen (Seite 64).

3.4.4.2 Quelltext des Beispielprogramms

Die Tabelle zeigt Ihnen den Quelltext des Beispielprogramms, so wie Sie ihn eingeben müssen, um lauffähigen Code erzeugen zu können.

Tabelle 3-13 Beispielprogramm Flash

```
INTERFACE
    VAR_GLOBAL
        counterVar : INT  := 1; // Zählervariable
        outputVar  : BYTE := 1; // Hilfsvariable
    END_VAR
    PROGRAM Flash;
END_INTERFACE

IMPLEMENTATION
    PROGRAM Flash
        IF counterVar >= 500 THEN // in jedem 500. Durchlauf
            %QB62 := outputVar;   // setze Ausgangsbyte
            outputVar := ROL (in := outputVar, n := 1);
            (* rotiere Bit in Byte
              nach links um eine Stelle *)
            counterVar := 0; // Zähler zurücksetzen
        END_IF;
        counterVar := counterVar + 1; // Zähler erhöhen
    END_PROGRAM
END_IMPLEMENTATION
```

3.4.5 Beispielprogramm übersetzen

Bevor Sie Ihr Programm ablaufen lassen oder testen können, müssen Sie es in ausführbaren Maschinencode übersetzen. Diese Aufgabe übernimmt der Compiler.

3.4.5.1 Compiler starten

Bevor Sie Ihr Programm ablaufen lassen oder testen können, müssen Sie es in ausführbaren Maschinencode übersetzen. Diese Aufgabe übernimmt der ST-Compiler.

So starten Sie den Compiler:

1. Klicken Sie in das Fenster mit dem ST-Editor, um das Menü **ST-Quelle** einzublenden. Dieses Menü ist ein dynamisches Menü und wird nur eingeblendet, wenn das Fenster des ST-Editors aktiv ist.
2. Starten Sie den Compiler mit Anwahl des Menüpunktes **ST-Quelle > Übernehmen und Übersetzen**.

3.4.5.2 Fehler korrigieren

Der Compiler überprüft die Syntax der ST-Quelle. Im Register **Ausgabe übersetzen/prüfen** der Detailanzeige werden die erfolgreiche Übersetzung des Quelltextes bzw. Compiler-Fehler angezeigt. Die Fehlerangabe umfasst: Name der ST-Quelle, Zeilennummer, in welcher der Fehler aufgetreten ist, Fehlernummer und Fehlerbeschreibung.

Gehen Sie folgendermaßen vor, um einen Fehler im Beispielprogramm zu korrigieren:

1. Doppelklicken Sie auf die Fehlermeldung. Die Schreibmarke wird in der betreffenden Zeile der ST-Quelle positioniert. Siehe Beispiel für Fehlermeldungen (Seite 91).
2. Beginnen Sie die Fehlerkorrektur mit dem ersten Fehler.
3. Starten Sie den Übersetzungsvorgang erneut.
4. Wiederholen Sie den gesamten Vorgang so lange, bis keine Fehler mehr angezeigt werden (**0 errors**).

Nach erfolgreicher Übersetzung haben Sie ein Anwenderprogramm mit dem Namen **flash** erstellt. Dieses wird im Projektnavigator unterhalb der Programmquelle **ST_1** angezeigt.

3.4.5.3 Beispiel für Fehlermeldungen

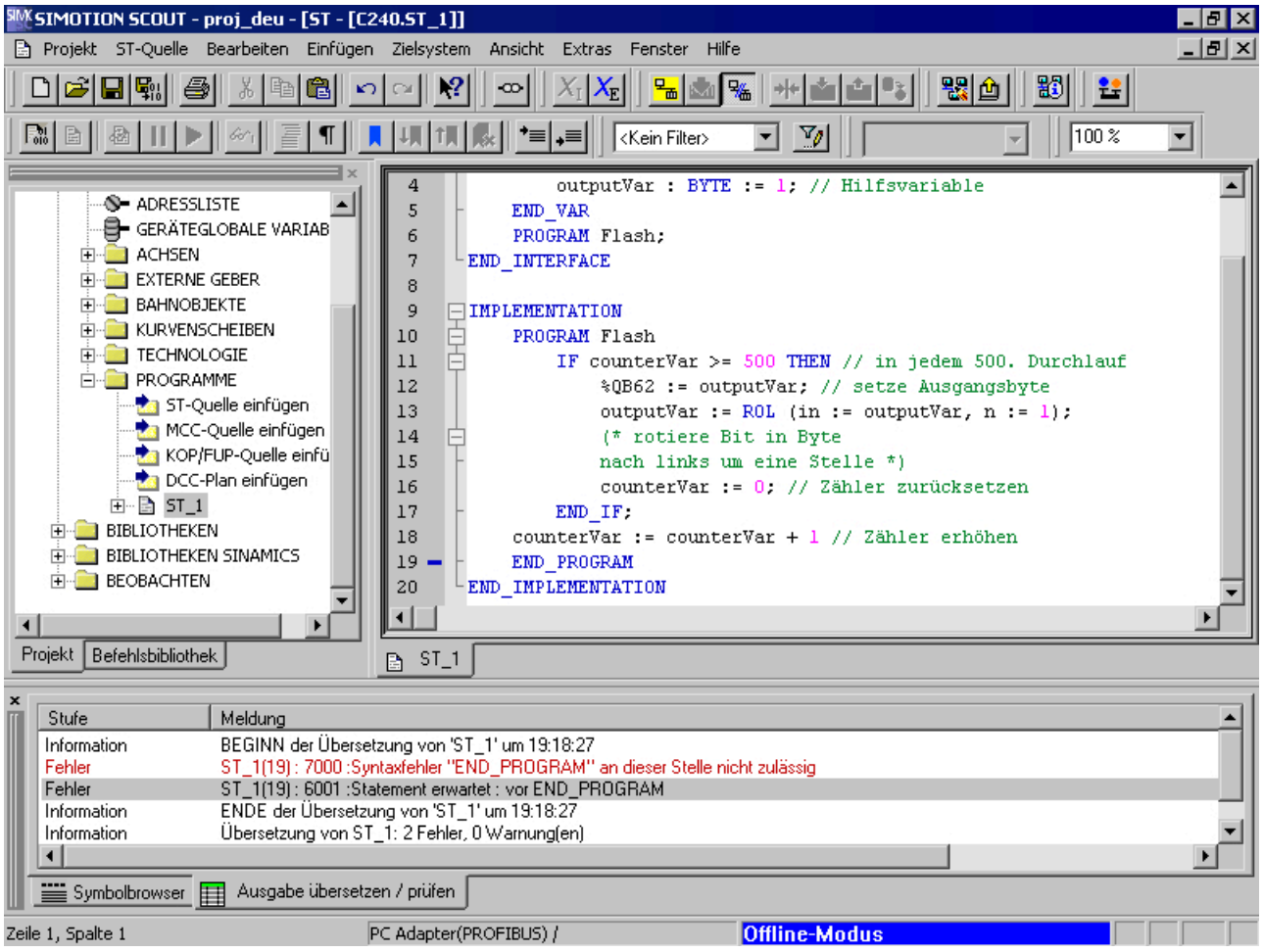


Bild 3-26 Fehlermeldung beim Übersetzen einer ST-Quelle

Das Bild zeigt als Beispiel die Übersetzung der ST-Quelle ST_1 (siehe Quelltext des Beispielprogramms (Seite 89)), in der folgende Veränderung vorgenommen wurde: Am Ende der Zeile 18 mit der Anweisung "counterVar := counterVar + 1" fehlt der Strichpunkt.

Der Compiler erkennt den Fehler erst in Zeile 19, da er nach dem fehlenden Strichpunkt mit der Übersetzung fortfährt.

Nach Ergänzen des fehlenden Strichpunkts wird die ST-Quelle fehlerfrei übersetzt.

Eine ausführliche Auflistung aller Compiler-Fehlermeldungen finden Sie in Fehlermeldungen des Compilers und deren Behebung (Seite 424).

3.4.6 Beispielprogramm ausführen

Bevor Sie das Beispielprogramm ausführen können, muss es einer Ablaufebene bzw. einer Task zugeordnet sein. Wenn diese Voraussetzung erfüllt ist, können Sie die Verbindung zum Zielsystem herstellen, das Programm in das Zielsystem laden und es anschließend starten.

3.4.6.1 Beispielprogramm einer Ablaufebene zuordnen

Die Ablaufebenen legen die zeitliche Abfolge von Programmen fest. Jede Ablaufebene enthält eine oder mehrere Tasks, denen Sie Programme zuordnen können.

Die Zuordnung eines Programmes zu einer Task kann nur nach dem Compilieren stattfinden und muss vor dem Laden des Programmes ins Zielsystem geschehen.

Das Beispielprogramm ordnen Sie der *BackgroundTask* zu. Die *BackgroundTask* ist vorgesehen für die Programmierung von zyklischen Abläufen ohne festes Zeitraster. Sie wird in der Round-Robin-Ablaufebene zyklisch ausgeführt, d. h. sie wird nach ihrer Beendigung automatisch erneut gestartet.

So ordnen Sie das Beispielprogramm der *BackgroundTask* zu:

1. Mit Doppelklick auf das Element **Ablaufsystem** im Projektnavigator öffnet sich im Arbeitsbereich das Fenster mit dem Ablaufsystem und der Programmzuordnung.
2. Klicken Sie auf die **BackgroundTask**, um sie für die Programmzuordnung zu markieren. In der Programmzuordnung auf der linken Seite sehen Sie alle compilierten Programme, die Sie Tasks zuordnen können.
3. Klicken Sie im Listenfeld **Programme** auf unser Beispielprogramm **ST_1.flash** und auf die Schaltfläche **>>**, um das Programm der BackgroundTask zuzuordnen.
   Das Ergebnis der Zuordnung zeigt das folgende Bild; im Listenfeld **Verwendete Programme** wird das Programm **ST_1.flash** angezeigt.

Näheres zum Ablaufsystem und der Programmzuordnung zu Tasks siehe Funktionsbeschreibung *SIMOTION Motion Contol Basisfunktionen*.

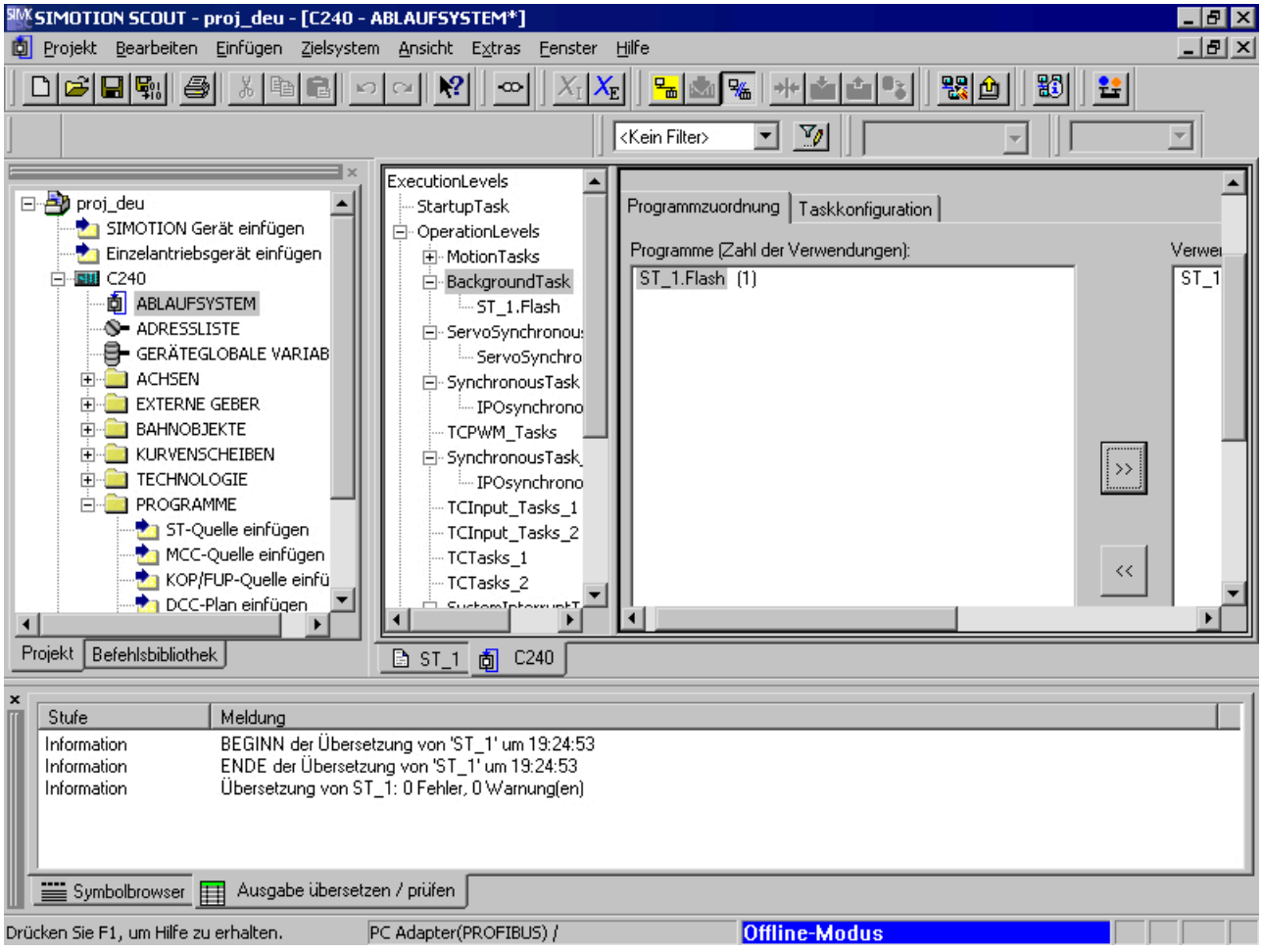


Bild 3-27 Beispielprogramm der BackgroundTask zuordnen

3.4.6.2 Verbindung mit dem Zielsystem herstellen

Voraussetzung ist, dass die Schnittstellenkarte des PC konfiguriert und an das Zielsystem angeschlossen ist.

So stellen Sie eine Verbindung mit dem Zielsystem her:

1. Wählen Sie in der Menüleiste **Projekt > Mit ausgewählten Zielgeräten verbinden**.
   In der Detailanzeige wird das Register **Diagnoseübersicht** geöffnet. Die Diagnoseübersicht zeigt Ihnen den Betriebszustand, die Speicherbelegung und die CPU-Auslastung für das Gerät an, mit dem Sie verbunden sind. Am rechten unteren Bildschirmrand sehen Sie, dass Sie mit dem Zielsystem verbunden sind.

Hinweis

Nähere Informationen siehe Projektierungshandbuch SIMOTION SCOUT und Online-Hilfe SIMOTION SCOUT.

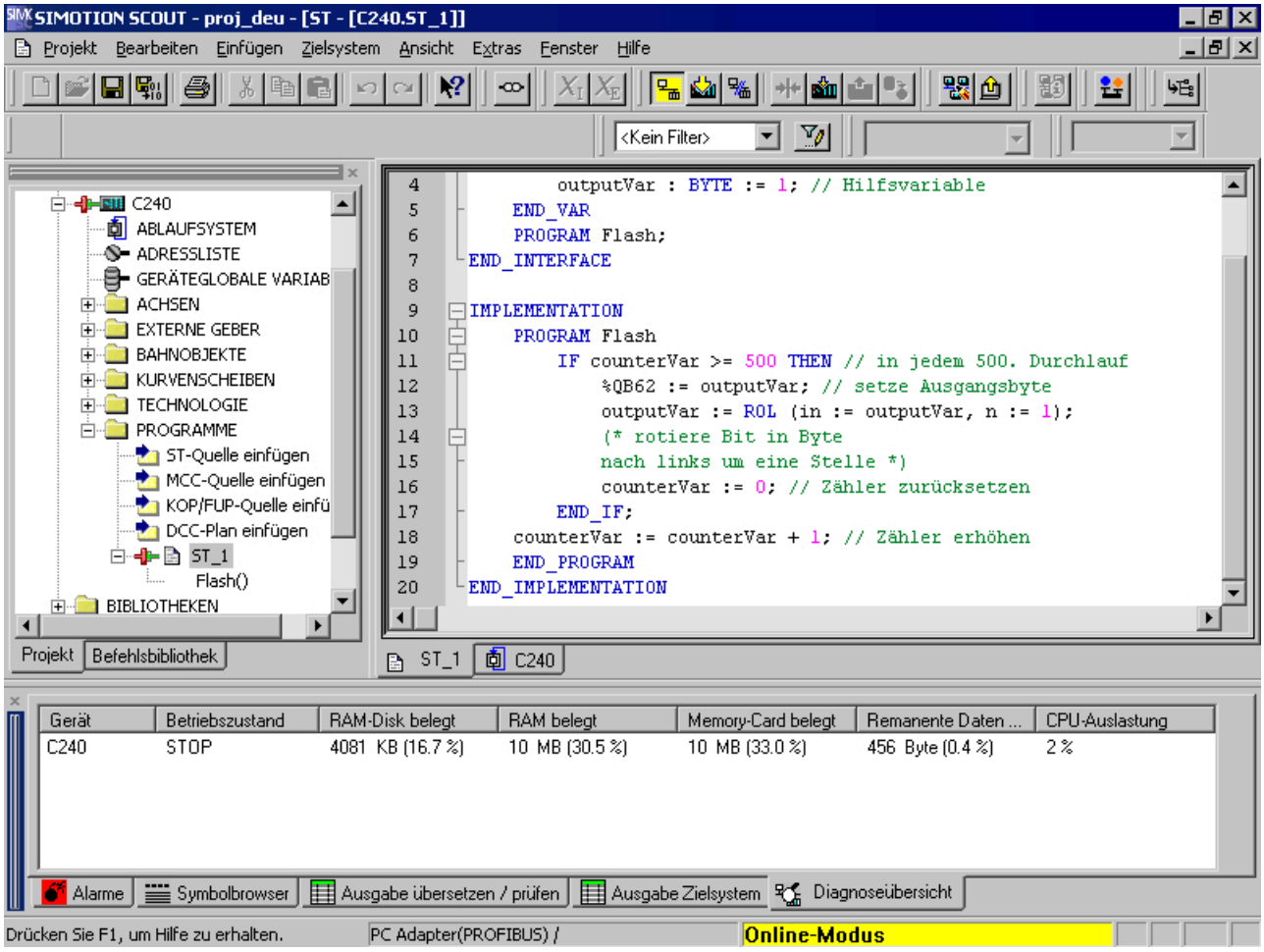


Bild 3-28 Verbindung mit dem Zielsystem herstellen

3.4.6.3 Beispielprogramm in das Zielsystem laden (Download)

So laden Sie das Beispielprogramm in das Zielsystem:

1. Schalten Sie das Zielsystem auf **STOP**.
2. Wählen Sie in der Menüleiste **Zielsystem > Laden > Projekt ins Zielsystem laden**.
3. Bestätigen Sie alle weiteren Abfragen.

Das Fenster Ausgabe Zielsystem in der Detailanzeige wird geöffnet und zeigt Ihnen das Ergebnis des Downloads an.

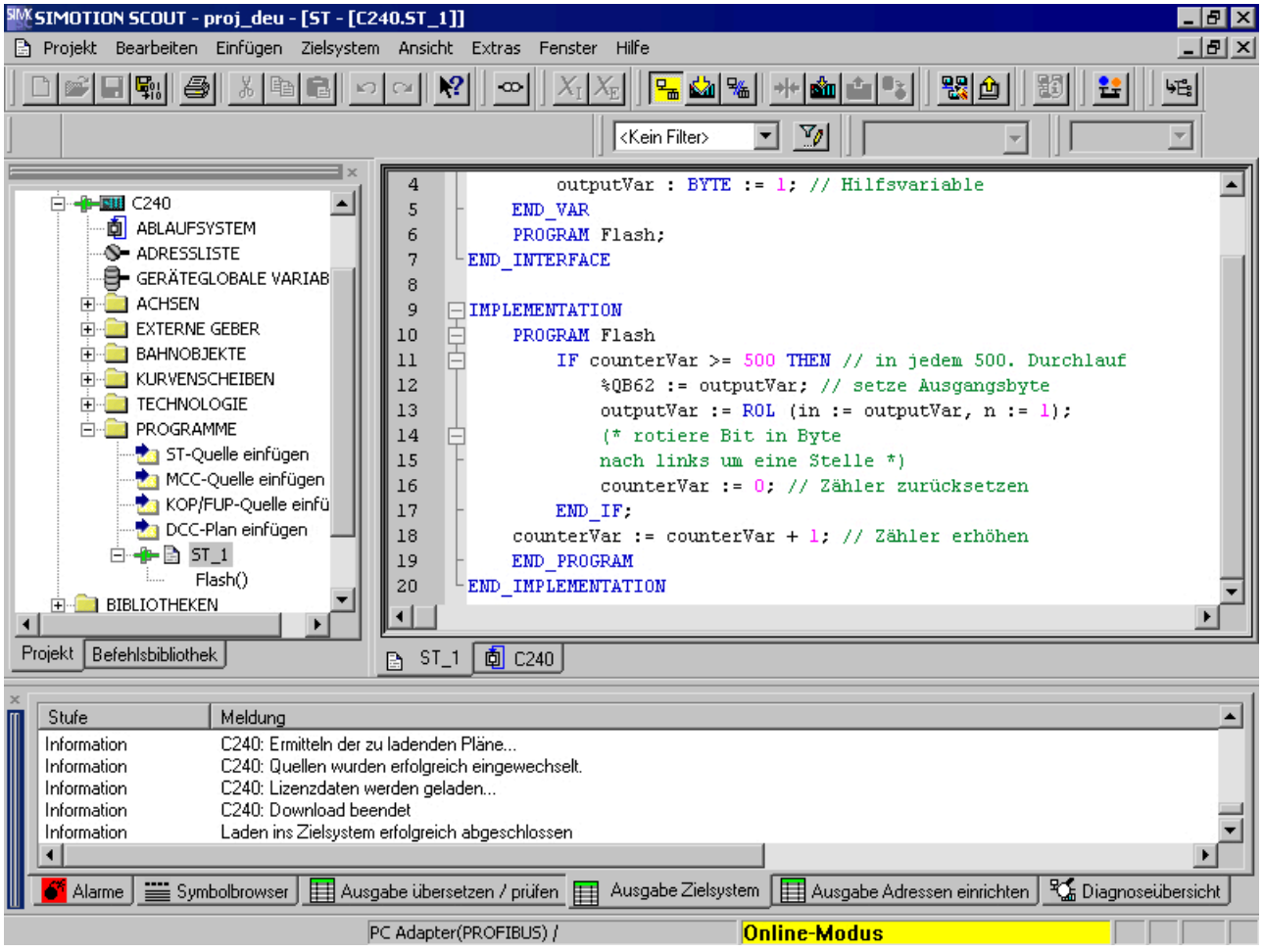


Bild 3-29 Beispielprogramm in das Zielsystem laden (Download)

3.4.6.4 Beispielprogramm starten und testen

Beispielprogramm starten

So starten Sie das Beispielprogramm:

- Schalten Sie Ihr Zielsystem auf RUN (siehe Hardwarebeschreibung).

An den Ausgängen Ihres Zielsystems blinken die Lämpchen in sequentieller Folge.

Beispielprogramm testen

Siehe Programmtest (Seite 325).

ST-Grundlagen

4

Sie erfahren in diesem Kapitel, welche Sprachmittel Ihnen ST zur Verfügung stellt und wie Sie mit diesen Sprachmitteln umgehen können. Beachten Sie, dass Funktionen, Funktionsbausteine und die Tasksteuerung in den nächsten Kapiteln behandelt werden. Eine vollständige formale Sprachbeschreibung mit allen Syntaxdiagrammen siehe Anhang Regeln (Seite 381).

4.1 Hilfen für die Sprachbeschreibung

Basis für die Sprachbeschreibung in den folgenden Kapiteln sind Syntaxdiagramme. Sie geben Ihnen einen guten Einblick in den syntaktischen (d. h. grammatikalischen) Aufbau von ST.

4.1.1 Syntaxdiagramm

Das Syntaxdiagramm ist eine grafische Darstellung der Struktur der Sprache. Die Struktur wird durch eine Folge von Regeln beschrieben. Dabei kann eine Regel auf bereits eingeführte Regeln aufbauen.

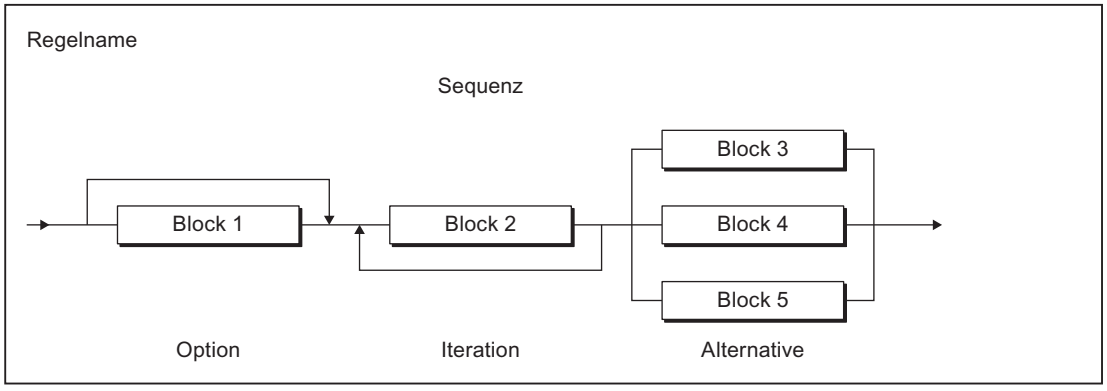


Bild 4-1 Syntaxdiagramm

Das Syntaxdiagramm aus dem vorhergehenden Bild wird von links nach rechts gelesen. Dabei sind die folgenden Regelstrukturen zu beachten:

- Sequenz: Folge von Blöcken
- Option: Überspringbare Anweisung(en)
- Iteration: Wiederholung einer oder mehrerer Anweisungen
- Alternative: Verzweigung

4.1.2 Blöcke in Syntaxdiagrammen

Ein Block ist ein Grundelement oder ein Element, das wiederum aus Blöcken zusammengesetzt ist. Das Bild zeigt die Symbolarten, die den Blöcken entsprechen:

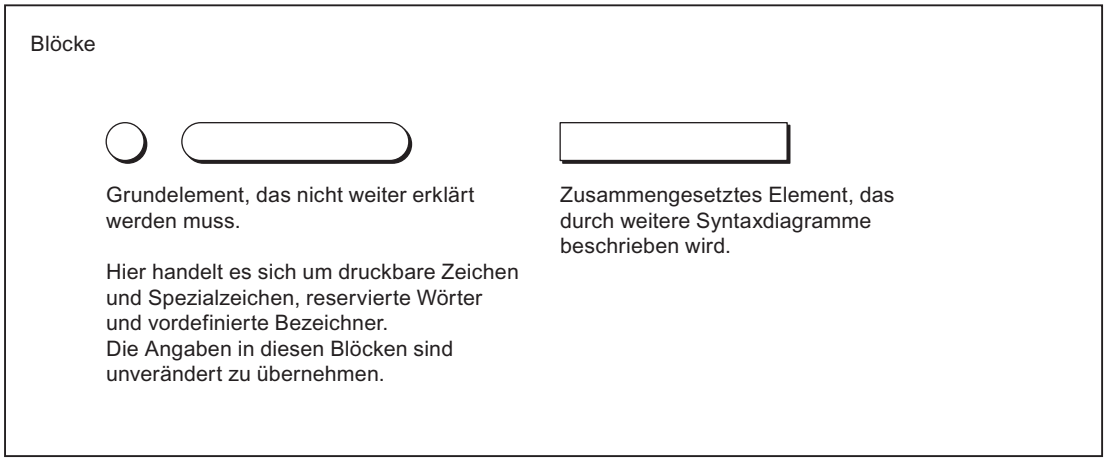


Bild 4-2 Blöcke

Bei der Eingabe von Quelltexten, d. h., wenn Sie Blöcke oder Elemente eines Syntaxdiagrammes in Quelltext umsetzen, sind sowohl die formatpflichtigen Regeln als auch die formatfreien Regeln zu beachten (siehe Hilfen für die Sprachbeschreibung (Seite 367)).

Siehe auch

Formale Sprachbeschreibung (Seite 367)

4.1.3 Bedeutung der Regeln (Semantik)

In den Regeln kann nur der formale Aufbau der Sprache dargestellt werden. Die Bedeutung, d. h. Semantik, geht nicht immer daraus hervor. Deshalb wird an wichtigen Stellen Zusatzinformation neben die Regeln geschrieben. Beispiele dafür sind:

- Bei gleichartigen Elementen mit unterschiedlicher Bedeutung wird ein Zusatzname angegeben. Beispielsweise wird in der Regel *Datumsangabe* bei jedem Element *Dezimalziffernfolge* ein Zusatz angegeben – entweder *Jahr, Monat* oder *Tag* (siehe Literale (Seite 382)). Der Name deutet auf die Verwendung hin.
- Wichtige Einschränkungen werden neben den Regeln vermerkt. Beispielsweise wird in Regel *Ganzzahl* bei **-** (Minus) vermerkt, dass es nur vor Dezimalziffernfolge der Datentypen SINT, INT und DINT stehen darf (siehe Literale (Seite 382)).

Siehe auch

Formale Sprachbeschreibung (Seite 367)

4.2 Basiselemente der Sprache

Zu den Basiselementen der Sprache ST gehören der ST-Zeichensatz, die daraus gebildeten reservierten Bezeichner (z. B. Sprachbefehle), selbstdefinierte Bezeichner sowie Zahlen.

Der ST-Zeichensatz und die reservierten Bezeichner sind Grundelemente (Terminale), da sie nicht durch eine weitere Regel, sondern verbal erklärt werden. Selbstdefinierte Bezeichner und Zahlen sind keine Terminale, da sie durch weitere Regeln erklärt werden.

In den Syntaxdiagrammen werden Terminale durch Kreise oder Ovale dargestellt, zusammengesetzte Elemente durch Rechtecke (siehe Blöcke in Syntaxdiagrammen (Seite 99)). Hier erhalten Sie eine Auswahl der wichtigsten Terminale, eine vollständige Übersicht finden Sie in Grundelemente (Terminale) (Seite 369).

4.2.1 Der ST-Zeichensatz

ST benutzt aus dem ASCII-Zeichensatz folgende **Buchstaben und Ziffern**:

- die Klein- und Großbuchstaben von A bis Z
- die arabischen Ziffern von 0 bis 9

Buchstaben und Ziffern sind die hauptsächlich verwendeten Zeichen. Der Bezeichner (siehe Bezeichner im ST (Seite 100)) besteht z. B. aus Buchstaben, Ziffern und Unterstrich. Letzteres gehört zu den sonstigen Zeichen.

Sonderzeichen haben im ST eine festgelegte Bedeutung (siehe Formale Sprachbeschreibung (Seite 367), Grundelemente (Terminale) (Seite 369)).

4.2.2 Bezeichner im ST

Bezeichner sind Namen im ST. Dies können vom System vorgegebene Namen sein, wie z. B. Sprachbefehle. Es können aber auch Namen sein, die Sie selbst vergeben, z. B. für eine Konstante, Variable oder Funktion.

4.2.2.1 Regeln für Bezeichner

Gültige Bezeichner

Bezeichner sind aus Buchstaben (A … Z, a … z), Ziffern (0 … 9) oder einzelnen Unterstrichen (_) in beliebiger Reihenfolge zusammengesetzt, wobei das erste Zeichen ein Buchstabe oder ein Unterstrich sein muss.

Zwischen Groß- und Kleinbuchstaben wird nicht unterschieden (z. B. werden Anna und AnNa vom Compiler als identisch betrachtet).

Ein Bezeichner kann durch das folgende Syntaxdiagramm formal dargestellt werden:

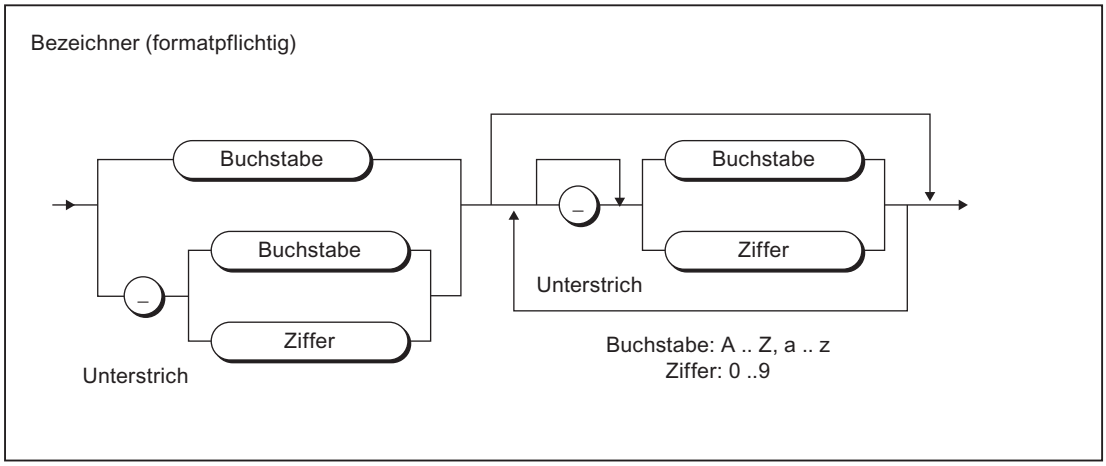


Bild 4-3 Syntax: Bezeichner

Dieses Syntaxdiagramm besagt, dass das erste Zeichen eines Bezeichners ein Buchstabe oder ein Unterstrich sein muss. Nach einem Unterstrich kommt zwingend ein Buchstabe oder eine Zahl, d. h. mehrere Unterstriche sind nicht erlaubt. Danach können Sie beliebig oft und in beliebiger Reihenfolge Unterstriche, Buchstaben oder Ziffern verwenden. Einzige Ausnahme ist auch hier der Unterstrich, der nicht mehrmals hintereinander verwendet werden darf.

Zulässige Bezeichner, die Sie definieren können (anwenderdefinierte Bezeichner)

Bezeichner, die Sie selbst definieren, z. B. in Variablendeklarationen, Datentypdeklaration, Funktionsnamen) dürfen nicht mit reservierten Bezeichnern (Seite 102) übereinstimmen. Bei der Namensvergabe wählen Sie am besten eindeutige und aussagekräftige Namen, die zur Verständlichkeit des Programms beitragen.

4.2.2.2 Beispiele für Bezeichner

Beispiele für gültige Bezeichner

Die folgenden Namen sind gültige Bezeichner:

```
x           y12         sum         temperature    P_CONTROLLER
name        area        myFB        table
```

Beispiele für ungültige Bezeichner

Die folgenden Namen sind **keine** gültigen Bezeichner:

Ungültiger Bezeichner	Grund
`4ter`	Das erste Zeichen muss ein Buchstabe oder ein Unterstrich sein.
`*#AB`	Sonderzeichen (außer Unterstrich) sind nicht erlaubt.
`RR__20`	Zwei aufeinander folgende Unterstriche sind nicht erlaubt.

Ungültiger Bezeichner	Grund
`S Wert`	Leerzeichen sind nicht erlaubt, da Sonderzeichen.
`Array`	ARRAY, REAL und _sizeOf sind zwar formal gültige, aber reservierte Bezeichner (Seite 102) , d. h. sie dürfen nur wie vorbestimmt benutzt werden. Somit dürfen Sie diese Namen nicht selbst vergeben, z. B. für eine Variable.
`REAL`	
`_sizeOf`	

4.2.3 Reservierte Bezeichner

Reservierte Bezeichner dürfen Sie nur wie vorbestimmt verwenden. Sie dürfen z. B. keine Variable und keinen Datentyp mit dem Namen eines reservierten Bezeichners deklarieren.

Eine Unterscheidung zwischen Groß- und Kleinschreibung findet nicht statt.

- Geschützte Bezeichner der Programmiersprache ST (Seite 102)
- Reservierte Bezeichner der Programmiersprache ST (Seite 107)
- Weitere reservierte Bezeichner (Seite 108), wie z. B. Standardfunktionen, Systemfunktionen, Systemvariablen

Eine Auflistung aller Bezeichner mit vordefinierter Bedeutung finden Sie im Funktionshandbuch SIMOTION Basisfunktionen.

4.2.3.1 Geschützte Bezeichner der Programmiersprache ST

In der Tabelle sehen Sie die geschützten Bezeichner der Sprache ST. Sie können nur wie vorbestimmt verwendet werden. Eine kurze Erklärung finden Sie im Anhang unter Reservierte Wörter (Seite 373). Die zugehörigen Syntaxdiagramme (Seite 98) finden Sie im Anhang unter Regeln (Seite 381).

Tabelle 4-1 Geschützte Bezeichner der Sprache ST

A	
ABS	ANYTYPE_TO_LITTLEBYTEARRAY
ACOS	ARRAY
AND	AS
ANYOBJECT	ASIN
ANYOBJECT_TO_OBJECT	AT
ANYTYPE_TO_BIGBYTEARRAY	ATAN
B	

BIGBYTEARRAY_TO_ANYTYPE	BY
BOOL	BYTE
BOOL_TO_BYTE	BYTE_TO_BOOL
BOOL_TO_DWORD	BYTE_TO_DINT
BOOL_TO_WORD	BYTE_TO_DWORD
BOOL_VALUE_TO_DINT	BYTE_TO_INT
BOOL_VALUE_TO_INT	BYTE_TO_SINT
BOOL_VALUE_TO_LREAL	BYTE_TO_UDINT
BOOL_VALUE_TO_REAL	BYTE_TO_UINT
BOOL_VALUE_TO_SINT	BYTE_TO_USINT
BOOL_VALUE_TO_UDINT	BYTE_TO_WORD
BOOL_VALUE_TO_UINT	BYTE_VALUE_TO_LREAL
BOOL_VALUE_TO_USINT	BYTE_VALUE_TO_REAL
C	
CASE	CTD_UDINT
CONCAT	CTU
CONCAT_DATE_TOD	CTU_DINT
CONSTANT	CTU_UDINT
COS	CTUD
CTD	CTUD_DINT
CTD_DINT	CTUD_UDINT
D	
DATE	DO
DATE_AND_TIME	DT
DATE_AND_TIME_TO_DATE	DT_TO_DATE
DATE_AND_TIME_TO_TIME_OF_DAY	DT_TO_TOD
DELETE	DWORD
DINT	DWORD_TO_BOOL
DINT_TO_BYTE	DWORD_TO_BYTE
DINT_TO_DWORD	DWORD_TO_DINT
DINT_TO_INT	DWORD_TO_INT
DINT_TO_LREAL	DWORD_TO_REAL
DINT_TO_REAL	DWORD_TO_SINT
DINT_TO_SINT	DWORD_TO_UDINT
DINT_TO_STRING	DWORD_TO_UINT
DINT_TO_UDINT	DWORD_TO_USINT
DINT_TO_UINT	DWORD_TO_WORD
DINT_TO_USINT	DWORD_VALUE_TO_LREAL
DINT_TO_WORD	DWORD_VALUE_TO_REAL
DINT_VALUE_TO_BOOL	
E	

ELSE	END_REPEAT
ELSIF	END_STRUCT
END_CASE	END_TYPE
END_EXPRESSION	END_VAR
END_FOR	END_WAITFORCONDITION
END_FUNCTION	END_WHILE
END_FUNCTION_BLOCK	ENUM_TO_DINT
END_IF	EXIT
END_IMPLEMENTATION	EXP
END_INTERFACE	EXPD
END_LABEL	EXPRESSION
END_PROGRAM	EXPT
F	
F_TRIG	FOR
FALSE	FUNCTION
FIND	FUNCTION_BLOCK
G	
GOTO	
I	
IF	INT_TO_SINT
IMPLEMENTATION	INT_TO_TIME
INSERT	INT_TO_UDINT
INT	INT_TO_UINT
INT_TO_BYTE	INT_TO_USINT
INT_TO_DINT	INT_TO_WORD
INT_TO_DWORD	INT_VALUE_TO_BOOL
INT_TO_LREAL	INTERFACE
INT_TO_REAL	
L	
LABEL	LREAL_TO_REAL
LEFT	LREAL_TO_SINT
LEN	LREAL_TO_STRING
LIMIT	LREAL_TO_UDINT
LITTLEBYTEARRAY_TO_ANYTYPE	LREAL_TO_UINT
LN	LREAL_TO_USINT
LOG	LREAL_VALUE_TO_BOOL
LREAL	LREAL_VALUE_TO_BYTE
LREAL_TO_DINT	LREAL_VALUE_TO_DWORD
LREAL_TO_INT	LREAL_VALUE_TO_WORD
M	
MAX	MOD
MID	MUX
MIN	
N	

NOT	
O	
OF	OVERLAP
OR	
P	
PROGRAM	
R	
R_TRIG	REAL_VALUE_TO_BYTE
REAL	REAL_VALUE_TO_DWORD
REAL_TO_DINT	REAL_VALUE_TO_WORD
REAL_TO_DWORD	REPEAT
REAL_TO_INT	REPLACE
REAL_TO_LREAL	RETAIN
REAL_TO_SINT	RETURN
REAL_TO_STRING	RIGHT
REAL_TO_TIME	ROL
REAL_TO_UDINT	ROR
REAL_TO_UINT	RS
REAL_TO_USINT	RTC
REAL_VALUE_TO_BOOL	
S	
SEL	SINT_TO_WORD
SHL	SINT_VALUE_TO_BOOL
SHR	SQRT
SIN	SR
SINT	STRING
SINT_TO_BYTE	STRING_TO_DINT
SINT_TO_DINT	STRING_TO_LREAL
SINT_TO_DWORD	STRING_TO_REAL
SINT_TO_INT	STRING_TO_UDINT
SINT_TO_LREAL	STRUCT
SINT_TO_REAL	StructAlarmId
SINT_TO_UDINT	STRUCTALARMID_TO_DINT
SINT_TO_UINT	StructTaskId
SINT_TO_USINT	
T	
TAN	TOD
THEN	TOF
TIME	TON
TIME_OF_DAY	TP
TIME_TO_INT	TRUE
TIME_TO_REAL	TRUNC
TO	TYPE
U	

UDINT	UINT_TO_UDINT
UDINT_TO_BYTE	UINT_TO_USINT
UDINT_TO_DINT	UINT_TO_WORD
UDINT_TO_DWORD	UINT_VALUE_TO_BOOL
UDINT_TO_INT	UNIT
UDINT_TO_LREAL	UNTIL
UDINT_TO_REAL	USELIB
UDINT_TO_SINT	USEPACKAGE
UDINT_TO_STRING	USES
UDINT_TO_UINT	USINT
UDINT_TO_USINT	USINT_TO_BYTE
UDINT_TO_WORD	USINT_TO_DINT
UDINT_VALUE_TO_BOOL	USINT_TO_DWORD
UINT	USINT_TO_INT
UINT_TO_BYTE	USINT_TO_LREAL
UINT_TO_DINT	USINT_TO_REAL
UINT_TO_DWORD	USINT_TO_SINT
UINT_TO_INT	USINT_TO_UDINT
UINT_TO_LREAL	USINT_TO_UINT
UINT_TO_REAL	USINT_TO_WORD
UINT_TO_SINT	USINT_VALUE_TO_BOOL
V	
VAR	VAR_OUTPUT
VAR_GLOBAL	VAR_TEMP
VAR_IN_OUT	VOID
VAR_INPUT	
W	
WAITFORCONDITION	WORD_TO_INT
WHILE	WORD_TO_SINT
WITH	WORD_TO_UDINT
WORD	WORD_TO_UINT
WORD_TO_BOOL	WORD_TO_USINT
WORD_TO_BYTE	WORD_VALUE_TO_LREAL
WORD_TO_DINT	WORD_VALUE_TO_REAL
WORD_TO_DWORD	
X	
XOR	

4.2.3.2 Reservierte Bezeichner der Programmiersprache ST

Die Tabelle enthält weitere reservierte Bezeichner, die künftigen Erweiterungen der Programmiersprache ST vorbehalten sind.

Tabelle 4-2 Reservierte Bezeichner der Programmiersprache ST

A	
ACTION ADD ADD_DT_TIME	ADD_TIME ADD_TOD_TIME
B	
BCD_TO_BYTE BCD_TO_DINT BCD_TO_DWORD BCD_TO_INT	BCD_TO_LWORD BCD_TO_SINT BCD_TO_WORD BYTE_TO_BCD
C	
CONFIGURATION CTD_LINT CTD_ULINT CTU_LINT	CTU_ULINT CTUD_LINT CTUD_ULINT
D	
DINT_TO_BCD DIV	DIVTIME DWORD_TO_BCD
E	
EN END_ACTION END_CONFIGURATION END_RESOURCE	END_STEP END_TRANSITION ENO EQ
F	
F_EDGE	FROM
G	
GE	GT
I	
INITIAL_STEP	INT_TO_BCD
L	
LE LINT LT	LWORD LWORD_TO_BCD
M	
MUL	MULTIME
N	
NE	
R	
R_EDGE	RESOURCE

S	
SEMA	SUB_DT_DT
SINT_TO_BCD	SUB_DT_TIME
STEP	SUB_TIME
SUB	SUB_TOD_TIME
SUB_DATE_DATE	SUB_TOD_TOD
T	
TRANSITION	
U	
ULINT	
V	
VAR_ACCESS	VAR_EXTERNAL
VAR_ALIAS	VAR_OBJECT
W	
WORD_TO_BCD	

4.2.3.3 Weitere reservierte Bezeichner

In Anwenderdefinitionen sind neben den geschützten Bezeichnern (Seite 102) und den reservierten Bezeichnern (Seite 107) der Programmiersprache ST auch Bezeichner zu vermeiden, die in SIMOTION eine definierte Bedeutung haben oder für künftige Erweiterungen vorgesehen sind.

Bezeichner mit definierter Bedeutung in SIMOTION

Deklarieren Sie (z. B. in Variablendeklarationen oder Datentypdeklarationen) keine Bezeichner, die bereits in SIMOTION definiert sind. Andernfalls können die in SIMOTION definierten Bezeichner durch Ihre Deklarationen verdeckt und möglicherweise nicht mehr erreicht werden. Unter Umständen kann der Compiler keine entsprechende Warnung absetzen, z. B: wenn das zugehörige Technologiepaket nicht importiert ist.

In SIMOTION sind als Bezeichner definiert:

- Allgemeine Standardfunktionen einschließlich der zugehörigen Datentypen
  (siehe auch Funktionshandbuch SIMOTION Basisfunktionen)
- Allgemeine Standardfunktionsbausteine einschließlich der zugehörigen Datentypen
  (siehe auch Funktionshandbuch SIMOTION Basisfunktionen)
- Funktionen zur Tasksteuerung , Laufzeitmessung und Meldungsprogrammierung einschließlich der zugehörigen Datentypen
  (siehe auch Funktionshandbuch SIMOTION Basisfunktionen)
- Systemfunktionen und Systemvariablen der SIMOTION Geräte einschließlich der zugehörigen Datentypen
  (siehe auch Listenhandbücher der SIMOTION Geräte)
- Systemfunktionen, Systemvariablen und Konfigurationsdaten der Technologieobjekte einschließlich der zugehörigen Datentypen
  (siehe auch Listenhandbücher der Technologiepakete)
- Geschützte und reservierte Bezeichner anderer Programmiersprachen

Eine Auflistung aller Bezeichner mit definierter Bedeutung finden Sie im Funktionshandbuch SIMOTION Basisfunktionen.

Bezeichner, die für künftige Erweiterungen von SIMOTION vorgesehen sind

Vermeiden Sie anwenderdefinierte Bezeichner, die für zukünftige Erweiterungen von SIMOTION vorgesehen sind. Sie treffen dadurch Vorsorge, dass auch in zukünftigen Versionen Ihre anwenderdefinierten Bezeichner keine in SIMOTION definierten Bezeichner verdecken.

Für künftige Erweiterungen von SIMOTION sind insbesondere alle Bezeichner vorgesehen, die mit folgenden Zeichenfolgen beginnen:

- _ (Unterstrich)
- **Command**
- **Enum**
- **Struct**

Bei der Deklaration eines Bezeichners, der mit diesen Zeichenfolgen beginnt, wird eine entsprechende Warnung (z. B. Warnungsnummer 16026) ausgegeben.

4.2.4 Zahlen und Boolesche Werte

Zahlen können im ST unterschiedlich geschrieben werden. Eine Zahl kann wahlweise ein Vorzeichen, einen Dezimalpunkt oder einen Exponenten beinhalten. Die folgenden Aussagen gelten für alle Zahlen:

- Kommata und Leerzeichen dürfen nicht innerhalb einer Zahl vorhanden sein.
- Zur optischen Trennung ist der Unterstrich (_) erlaubt.
- Der Zahl kann wahlweise ein Plus (+) oder Minus (–) vorangestellt werden. Falls kein Vorzeichen bei der Zahl steht, wird sie als positiv angenommen.
- Zahlen dürfen bestimmte Maximal- und Minimalwerte nicht über- bzw. unterschreiten.

4.2.4.1 Ganzzahlen

Eine Ganzzahl enthält weder einen Dezimalpunkt noch einen Exponenten. Somit ist eine Ganzzahl einfach eine Folge von Ziffern, die wahlweise mit einem Vorzeichen beginnt.

Einige gültige Ganzzahlen:

```
0                1               +1              -1
743              -5280           60_000          -32_211_321
```

Die folgenden Ganzzahlen sind aus den angeführten Gründen **falsch**:

`123,456`	Kommata sind nicht erlaubt.
`36.`	In einer Ganzzahl darf kein Dezimalpunkt stehen.
`10 20 30`	Leerzeichen sind nicht erlaubt.

Im ST können Sie Ganzzahlen in unterschiedlichen Zahlensystemen darstellen. Das erfolgt durch Voranstellen eines Schlüsselwortes für das Zahlensystem.

Dabei steht:

- 2# für das Binärsystem,
- 8# für das Oktalsystem und
- 16# für das Hexadezimalsystem.

Gültige Darstellungen der Dezimalzahl 15 sind:

```
2#1111          8#17          16#F
```

4.2.4.2 Gleitpunktzahlen

Eine Gleitpunktzahl kann einen Dezimalpunkt oder einen Exponent (oder beides) enthalten. Ein Dezimalpunkt muss zwischen zwei Ziffern stehen. Somit kann eine Gleitpunktzahl nicht mit einem Dezimalpunkt anfangen oder enden.

Einige gültige Gleitpunktzahlen:

```
0.0             1.3             -0.2            827.602
0000.0          +0.000743       60_000.15       -315.0066
```

Die folgenden Gleitpunktzahlen sind **falsch**:

`1.`	Auf beiden Seiten des Dezimalpunktes muss eine Ziffer stehen.
`1,000.0`	Kommata sind nicht erlaubt.
`1.333.333`	Zwei Punkte sind nicht erlaubt.

4.2.4.3 Exponenten

Ein Exponent kann enthalten sein, um die Lage des Dezimalpunktes festzulegen. Falls kein Dezimalpunkt vorhanden ist, wird angenommen, dass er auf der rechten Seite der Ziffer steht. Der Exponent selbst muss entweder eine positive oder negative Ganzzahl sein. Die Basis 10 wird mittels des Buchstabens E ausgedrückt.

Die Größe 3 x 10^8 kann im ST durch folgende richtige Gleitpunktzahlen dargestellt werden:

```
3.0E+8          3.0E8           3e+8            3E8             0.3E+9
0.3e9           30.0E+7         30e7
```

Die folgenden Gleitpunktzahlen sind **falsch**:

`3.E+8`	Auf beiden Seiten des Dezimalpunkts muss eine Ziffer stehen.
`8e2.3`	Der Exponent muss eine Ganzzahl sein.
`.333e-3`	Auf beiden Seiten des Dezimalpunkts muss eine Ziffer stehen.
`30 E8`	Leerzeichen sind nicht erlaubt.

4.2.4.4 Boolesche Werte

Boolesche Werte sind Bit-Konstanten. Sie müssen mit dem Wert Null (0) oder Eins (1) dargestellt werden oder durch die Schlüsselwörter FALSE bzw. TRUE.

Beispiel:

```
a := 1;         // entspricht a := TRUE
b := FALSE;     // entspricht b := 0
```

4.2.4.5 Datentypen von Zahlen

Der Compiler wählt automatisch den zur Zahl passenden elementaren Datentyp in Abhängigkeit von deren Wert und der Verwendung (in Ausdruck oder Wertzuweisung).

Darüber hinaus können Sie den Datentyp direkt spezifizieren: Stellen Sie den Datentyp (numerischer Datentyp oder Bitdatentyp) und das Zeichen "#" der Zahl voran.

Beispiele:

INT#255	INT#16#FF	INT#8#377
WORD#255	WORD#16#FF	WORD#8#377
REAL#255	REAL#16#FF	REAL#8#377
REAL#255.0	REAL#2.55E2	LREAL#255.0

Hinweis

Gleitpunktzahlen können nur den Datentypen REAL und LREAL zugeordnet werden.

4.2.5 Zeichenstrings

Was ist ein Zeichenstring?

Ein Zeichenstring ist ein Folge von 0 oder mehr Zeichen, die durch ein Apostroph (Hochkomma) am Anfang und Ende eingeschlossen wird. Jedes Zeichen wird im String mit 1 Byte (8 Bit) verschlüsselt.

Ein Zeichen kann folgendermaßen eingeben werden:

- als druckbares Zeichen (ASCII-Code $20 bis $7E, $80 bis $FF), außer Dollarzeichen (ASCII-Code $24) und Apostroph (ASCII-Code $27), da diese innerhalb von Strings eine Sonderfunktion haben,
- durch den zweistelligen hexadezimalen ASCII-Code des betreffenden Zeichens mit vorangestelltem Dollarzeichen ($),
- als Kombination aus zwei Zeichen entsprechend der folgenden Tabelle:

Tabelle 4-3 Zwei-Zeichen-Kombinationen für Sonderzeichen in Strings

Zeichenkombination			Bedeutung
$$			Dollarzeichen $ ($24)
$'			Apostroph (Hochkomma) ' ($27)
$L	oder	$l	Line Feed LF ($0A)
$N	oder	$n	Carriage Return + Line Feed CR + LF ($0D$0A)
$P	oder	$p	Form Feed FF ($0C)
$R	oder	$r	Carriage Return CR ($0D)
$T	oder	$t	Horizontal Tab (HT) ($09)

Beispiele:

''	Leerer String (Länge 0).
'A'	String der Länge 1, der den Buchstaben A enthält.
' '	String der Länge 1, der ein Leerzeichen enthält.
'$''	String der Länge 1, der ein Apostroph (Hochkomma) enthält.
'RL' '$0D$0A'	Zwei gleichwertige Darstellungen für einen String der Länge 2, der die Zeichen CR und LF enthält.
'$$1.00'	String der Länge 5, der $1.00 enthält.
'Text$R$L'	String der Länge 6, der das Wort Text gefolgt von den Zeichen CR und LF enthält.
'ÄÖÜ' '$C4$D6$FC'	Zwei gleichwertige Darstellungen für einen String der Länge 3, der die deutschen Umlaute ÄÖü (A, O, u mit Diaresis) enthält.

4.3 Gliederung der ST-Quelle

Eine ST-Quelldatei besteht prinzipiell aus fortlaufendem Text. Dieser Text kann durch logische Abschnitte gegliedert und strukturiert werden. Die ausführlichen Regeln hierzu finden Sie in Quelldatei-Abschnitte (Seite 212).

Hier kurz eine Zusammenfassung:

- Eine **ST-Quelle** ist eine logische Einheit, die Sie in Ihrem Projekt anlegen und die mehrmals vorkommen kann. Sie wird oft auch als Unit bezeichnet.
- Die logischen Teile einer ST-Quelle werden als **Abschnitte** bezeichnet (siehe Tabelle).
- Ein **Anwenderprogramm** ist die Summe aller Programmquellen (z. B. ST-Quellen, MCC-Quellen).

Jeder logische Abschnitt der ST-Quelle hat einen Anfang und ein Ende, die mit spezifischen Schlüsselwörtern gekennzeichnet sind:

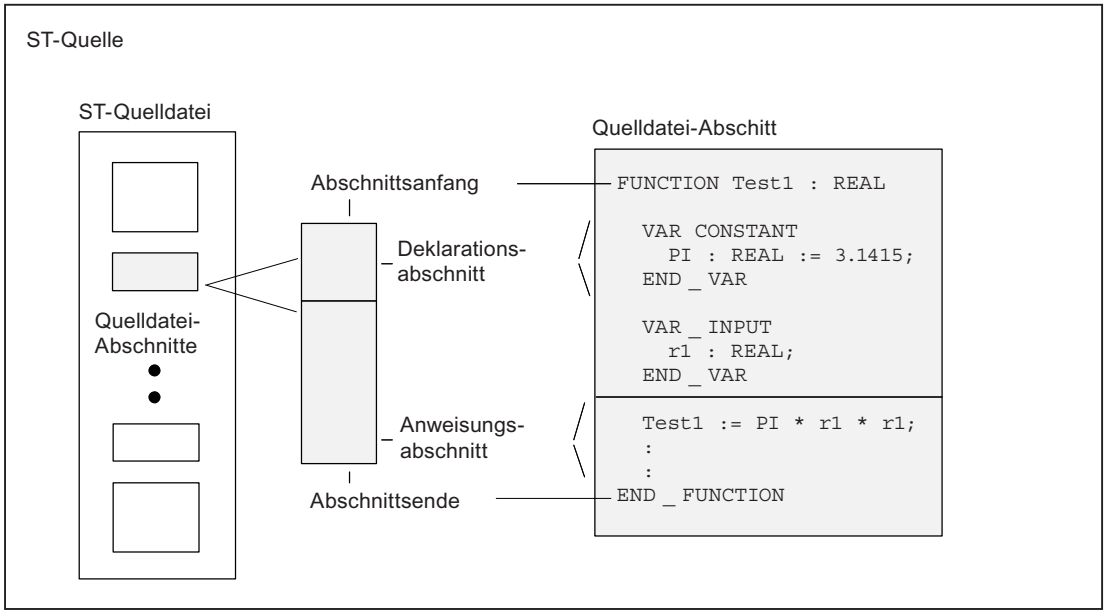


Bild 4-4 Aufbau einer ST-Quelle

Nicht jede Funktion müssen Sie selbst programmieren. Sie können auch auf SIMOTION Systemkomponenten zurückgreifen. Das sind vorgefertigte Abschnitte wie Systemfunktionen bzw. Funktionen der Technologieobjekte (TO-Funktionen).

Tabelle 4-4 Wichtige Abschnitte einer ST-Quelle

Quelldatei-Abschnitt	Beschreibung
Unit Anweisung (optional)	Enthält den Namen der ST
Interfaceabschnitt	Enthält Anweisungen für Import und Export von Variablen, Typen, Programmorganisationseinheiten (POE).
Implementationsabschnitt	Enthält die ausführbaren Abschnitte der ST-Quelle.
POE (Programmorganisationseinheit)	Einzelner ausführbarer Abschnitt der ST-Quelle (Programm, Funktion, Funktionsbaustein)

Quelldatei-Abschnitt	Beschreibung
Deklarationsabschnitt	Enthält Deklarationen (z. B. von Variablen- und Typen), kann im Interface- und Implementationsabschnitt sowie in einer POE enthalten sein.
Anweisungsabschnitt	Enthält die ausführbaren Anweisungen einer POE.

Hinweis

Über die Online-Hilfe ist ein ausführlich kommentiertes *Template für Beispiel-Unit* verfügbar. Sie können es als Vorlage für eine neue ST-Quelle verwenden.

Rufen Sie die Hilfe zum ST-Editor auf und klicken Sie auf die entsprechende Verknüpfung. Kopieren Sie den Text in das geöffnete Fenster des ST-Editors und passen Sie das Template Ihren Erfordernissen an.

Template für Beispiel-Unit enthält einen Abdruck dieses Templates.

4.3.1 Anweisungen

Der Anweisungsteil einer Programmorganisationseinheit (POE – Programm, Funktion, Funktionsbaustein) besteht aus wiederholten einzelnen Anweisungen. Er folgt dem Deklarationsabschnitt einer POE und endet mit dem Ende der POE. Er hat keine expliziten Schlüsselwörter für Anfang und Ende.

Es gibt im ST drei grundlegende Anweisungen:

- Wertzuweisungen
  Zuordnung eines Ausdrucks zu einer Variablen, siehe Variablendeklaration (Seite 139)
- Kontrollanweisungen
  Wiederholung oder Verzweigung von Anweisungen, siehe Kontrollanweisungen (Seite 167)
- Unterprogrammbearbeitung
  Funktionen (FC) und Funktionsbausteine (FB), siehe Funktionen, Funktionsbausteine, Programme (Seite 185)

Tabelle 4-5 Beispiele für Anweisungen

```
...
// Wertzuweisung
Status := 17;

// Kontrollanweisung
IF a = b THEN
    FOR c := 1 TO 10 DO
        b := b + c;
    END_FOR;
END_IF;

// Funktionsaufruf
retVal := Test1(10.0);
...
```

4.3.2 Kommentare

Kommentare dienen der Dokumentation und dem besseren Verständnis eines Quelldatei-Abschnitts. Sie sind nach dem Kompilieren für den Programmablauf ohne Bedeutung.

Es gibt zwei Kommentararten:

- den Zeilenkommentar und
- den Blockkommentar.

Den Zeilenkommentar leiten Sie mit // ein. Der darauf folgende Text wird bis zum Zeilenende vom Compiler als Kommentar behandelt.

Den Blockkommentar können Sie über mehrere Zeilen eingeben, wenn Sie ihn mit (* einleiten und mit *) abschließen.

Bitte beachten Sie beim Einbau von Kommentaren:

- Sie können den vollständigen, erweiterten ASCII Zeichensatz in Kommentaren benutzen.
- Innerhalb des Zeilenkommentars sind die Zeichenpaare (* und *) bedeutungslos.
- Die Schachtelung von Blockkommentaren ist nicht erlaubt. Möglich ist jedoch die Schachtelung von Zeilenkommentaren in Blockkommentaren.
- Der Einbau ist an beliebigen Stellen möglich, nicht jedoch in Regeln, die streng einzuhalten sind, wie z. B. in Namen von Bezeichnern. Mehr zu diesen Regeln erfahren Sie in Hilfen für die Sprachbeschreibung (Seite 367).

Tabelle 4-6 Beispiele für Kommentare

```
// Dies ist ein einzeiliger Kommentar.
a := 5;

// Dies ist ein Beispiel für einen mehrmals
// hintereinander verwendeten einzeiligen Kommentar.
b := 23;

(* Das Beispiel von oben ist als mehrzeiliger Kommentar leichter zu pflegen.
*)
c := 87;
```

4.4 Datentypen

Mit Hilfe eines Datentyps wird festgelegt, wie der Wert einer Variablen oder Konstanten in einer Programmquelle verwendet werden soll.

Dem Anwender stehen folgende Arten von Datentypen zur Verfügung:

- Elementare Datentypen (Seite 116)
- Anwenderdefinierte Datentypen (UDT) (Seite 120)
  - einfache Ableitungen
  - Felder (Array)
  - Aufzählungen (Enumeratoren)
  - Strukturen (Struct)
- Datentypen von Technologieobjekten (Seite 135)
- Systemdatentypen (Seite 138)

4.4.1 Elementare Datentypen

Elementare Datentypen definieren die Struktur von Daten, die nicht in kleinere Einheiten zerlegt werden können. Ein elementarer Datentyp beschreibt einen Speicherbereich mit fester Länge und steht für Bitdaten, Ganzzahlen, Gleitpunktzahlen, Zeitdauer, Uhrzeit, Datum und Zeichenstrings.

Eine Aufstellung aller elementaren Datentypen sehen Sie in der folgenden Tabelle:

Tabelle 4-7 Bitbreiten und Wertebereiche der elementaren Datentypen

Typ		Reserv. Wort	Bitbreite	Wertebereich
Bitdatentyp Daten dieses Typs umfassen entweder 1 Bit, 8 Bit, 16 Bit oder 32 Bit. Der Initialisierungswert einer Variablen dieses Datentyps ist 0.				
	Bit	BOOL	1	0, 1 oder FALSE, TRUE
	Byte	BYTE	8	16#0 bis 16#FF
	Wort	WORD	16	16#0 bis 16#FFFF
	Doppelwort	DWORD	32	16#0 bis 16#FFFF_FFFF
Numerische Typen Sie stehen für die Verarbeitung numerischer Werte zur Verfügung. Der Initialisierungswert einer Variablen dieses Datentyps ist 0 (alle Ganzzahlen) bzw. 0.0 (alle Gleitpunktzahlen).				

Typ		Reserv. Wort	Bitbreite	Wertebereich
	Kurze Ganzzahl	SINT	8	-128 bis 127 (-2**7 bis 2**7-1)
	Vorzeichenlose kurze Ganzzahl	USINT	8	0 bis 255 (0 bis 2**8-1)
	Ganzzahl (Integer)	INT	16	-32_768 bis 32_767 (-2**15 bis 2**15-1)
	Vorzeichenlose Ganzzahl	UINT	16	0 bis 65_535 (0 bis 2**16-1)
	Ganzzahl doppelter Breite	DINT	32	-2_147_483_648 bis 2_147_483_647 (-2**31 bis 2**31-1)
	Vorzeichenlose Ganzzahl doppelter Breite	UDINT	32	0 bis 4_294_967_295 (0 bis 2**32-1)
	Gleitpunktzahl (gemäß IEEE-754)	REAL	32	-3.402_823_466E+38 bis -1.175_494_351E-38, 0.0, +1.175_494_351E-38 bis +3.402_823_466E+38 Genauigkeit: Mantisse 23 Bit (entspricht 6 Dezimalstellen), Exponent 8 Bit, Vorzeichen 1 Bit.
	Lange Gleitpunktzahl (gemäß IEEE-754)	LREAL	64	-1.797_693_134_862_315_7E+308 bis -2.225_073_858_507_201_4E-308, 0.0, +2.225_073_858_507_201_4E-308 bis +1.797_693_134_862_315_7E+308 Genauigkeit: Mantisse 52 Bit (entspricht 15 Dezimalstellen), Exponent 11 Bit, Vorzeichen 1 Bit.
Zeittypen Daten dieses Typs repräsentieren die unterschiedlichen Zeit- und Datumswerte.				
	Zeitdauer in Schritten von 1 ms	TIME	32	T#0d_0h_0m_0s_0ms bis T#49d_17h_2m_47s_295ms Maximal 2 Stellen für die Werte Tag, Stunde, Minute, Sekunde; maximal 3 Stellen für Millisekunden. Initialisierung mit T#0d_0h_0m_0s_0ms
	Datum in Schritten von 1 Tag	DATE	32	D#1992-01-01 bis D#2200-12-31 Schaltjahre sind berücksichtigt, Jahreszahl ist vierstellig, Monat und Tag zweistellig. Initialisierung mit D#0001-01-01
	Uhrzeit in Schritten von 1 ms	TIME_OF_DAY (Kürzel TOD)	32	TOD#0:0:0.0 bis TOD#23:59:59.999 Maximal 2 Stellen für die Werte Stunde, Minute, Sekunde; maximal 3 Stellen für Millisekunden. Initialisierung mit TOD#0:0:0.0
	Datum und Uhrzeit	DATE_AND_TIME (Kürzel DT)	64	DT#1992-01-01-0:0:0.0 bis DT#2200-12-31-23:59:59.999 DATE_AND_TIME besteht aus den Datentypen DATE und TIME. Initialisierung mit DT#0001-01-01-0:0:0.0

Typ	Reserv. Wort	Bitbreite	Wertebereich
String-Typ Daten dieses Typs repräsentieren Zeichenketten, bei denen jedes Zeichen mit der angegebenen Anzahl von Bytes verschlüsselt wird. Die Länge des Strings kann bei der Deklaration vorgegeben werden Geben Sie hierzu die Länge in "[" und "]" an, z. B. STRING[100]. Die Voreinstellung beträgt 80 Zeichen. Die Anzahl der belegten (initialisierten) Zeichen kann kleiner als die deklarierte Länge sein.			
String mit 1 Byte/Zeichen	STRING	8	Alle Zeichen mit ASCII-Code $00 bis $FF sind zulässig. Vorbelegung mit ' ' (Leerer String)

Hinweis

Beim Export der Variablen in andere Systeme sind die Wertebereiche der entsprechenden Datentypen im Zielsystem zu beachten.

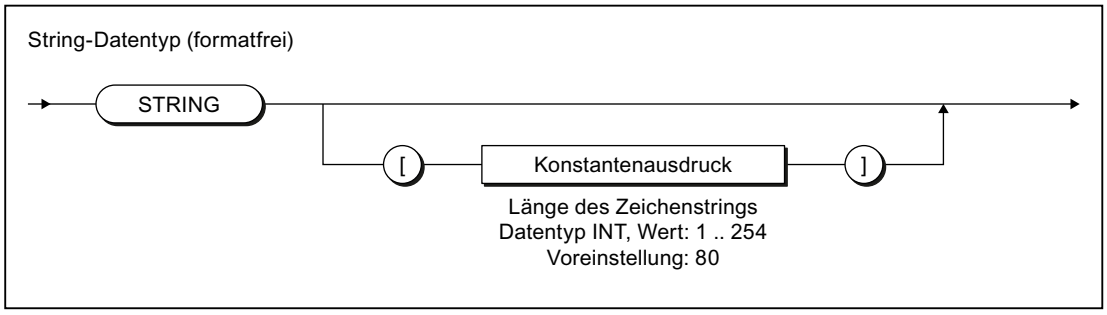


Bild 4-5 Syntax: Datentyp STRING

4.4.1.1 Wertebereichsgrenzen elementarer Datentypen

Die Grenzen der Wertebereiche einiger elementarer Datentypen sind als Konstanten verfügbar:

Tabelle 4-8 Symbolische Konstanten für die Wertebereichsgrenzen elementarer Datentypen

Symbolische Konstante	Datentyp	Wert	Hex-Darstellung
SINT#MIN	SINT	-128	16#80
SINT#MAX	SINT	127	16#7F
INT#MIN	INT	-32768	16#8000
INT#MAX	INT	32767	16#7FFF
DINT#MIN	DINT	-2147483648	16#8000_0000
DINT#MAX	DINT	2147483647	16#7FFF_FFFF
USINT#MIN	USINT	0	16#00
USINT#MAX	USINT	255	16#FF
UINT#MIN	UINT	0	16#0000
UINT#MAX	UINT	65535	16#FFFF
UDINT#MIN	UDINT	0	16#0000_0000

Symbolische Konstante	Datentyp	Wert	Hex-Darstellung
UDINT#MAX	UDINT	4294967295	16#FFFF_FFFF
T#MIN TIME#MIN	TIME	T#0ms	16#0000_0000[1]
T#MAX TIME#MAX	TIME	T#49d_17h_2m_47s_295ms	16#FFFF_FFFF[1]
TOD#MIN TIME_OF_DAY#MIN	TOD	TOD#00:00:00.000	16#0000_0000[1]
TOD#MAX TIME_OF_DAY#MAX	TOD	TOD#23:59:59.999	16#0526_5BFF[1]
[1] Nur interne Darstellung			

4.4.1.2 Allgemeine Datentypen

Allgemeine Datentypen werden oft bei den Eingangs- und Ausgangsparametern von Systemfunktionen und Systemfunktionsbausteinen verwendet. Das Unterprogramm kann mit Variablen jeden Datentyps aufgerufen werden, der im allgemeinen Datentyp enthalten ist.

Die nachfolgende Tabelle zeigt die verfügbaren allgemeinen Datentypen:

Tabelle 4-9 Allgemeine Datentypen

Allgemeiner Datentyp	Enthaltene Datentypen
ANY_BIT	BOOL, BYTE, WORD, DWORD
ANY_INT	SINT, INT, DINT, USINT, UINT, UDINT
ANY_REAL	REAL, LREAL
ANY_NUM	ANY_INT, ANY_REAL
ANY_DATE	DATE, TIME_OF_DAY (TOD), DATE_AND_TIME (DT)
ANY_ELEMENTARY	ANY_BIT, ANY_NUM, ANY_DATE, TIME, STRING
ANY	ANY_ELEMENTARY, anwenderdefinierte Datentypen (UDT), Systemdatentypen, Datentypen der Technologieobjekte

Hinweis

Allgemeine Datentypen können Sie **nicht** als Typbezeichner in Variablen- oder Typdeklarationen verwenden.

Der allgemeine Datentyp bleibt erhalten, wenn ein anwenderdefinierter Datentyp (UDT) direkt von einem elementaren Datentyp abgeleitet wird (nur bei der Programmiersprache SIMOTION ST möglich).

4.4.1.3 Elementare Systemdatentypen

Im SIMOTION System werden die in der Tabelle angegebenen Datentypen ähnlich den elementaren Datentypen behandelt. Sie werden bei vielen Systemfunktionen verwendet.

Tabelle 4-10 Elementare Systemdatentypen und ihre Verwendung

Bezeichner	Bitbreite	Verwendung
StructAlarmId	32	Datentyp der AlarmId für die projektweit eindeutige Kennung der Meldungen. Die AlarmId wird bei der Meldungsgenerierung verwendet. Siehe Funktionshandbuch *SIMOTION Basisfunktionen*. Initialisierung mit STRUCTALARMID#NIL
StructTaskId	32	Datentyp der TaskId für die projektweit eindeutige Kennung der Tasks im Ablaufsystem. Siehe Funktionshandbuch *SIMOTION Basisfunktionen*. Initialisierung mit STRUCTTASKID#NIL

Tabelle 4-11 Symbolische Konstanten für ungültige Werte elementarer Systemdatentypen

Symbolische Konstante	Datentyp	Bedeutung
STRUCTALARMID#NIL	StructAlarmId	Ungültige AlarmId
STRUCTTASKID#NIL	StructTaskId	Ungültige TaskId

4.4.2 Anwenderdefinierte Datentypen

Anwenderdefinierte Datentypen oder UDT (user defined data types), erstellen Sie mit dem Konstrukt TYPE / END_TYPE in den Deklarationsabschnitten nachstehender Quelldatei-Abschnitte (siehe Gliederung der ST-Quelle (Seite 113) und Quelldatei-Abschnitte (Seite 212)):

- des Interfaceabschnitts
- des Implementationsabschnitts
- einer POE

Die erstellten Datentypen können Sie im Deklarationsabschnitt weiter verwenden. Der Quelldatei-Abschnitt bestimmt die Reichweite der Typdeklaration.

Siehe auch

Syntax anwenderdefinierter Datentypen (Typdeklaration) (Seite 121)

Ableitung elementarer oder abgeleiteter Datentypen (Seite 123)

Abgeleiteter Datentyp ARRAY - Feld (Seite 123)

Abgeleiteter Datentyp Aufzählung - Enumerator (Seite 126)

Abgeleiteter Datentyp STRUCT - Struktur (Seite 128)

4.4.2.1 Syntax anwenderdefinierter Datentypen (Typdeklaration)

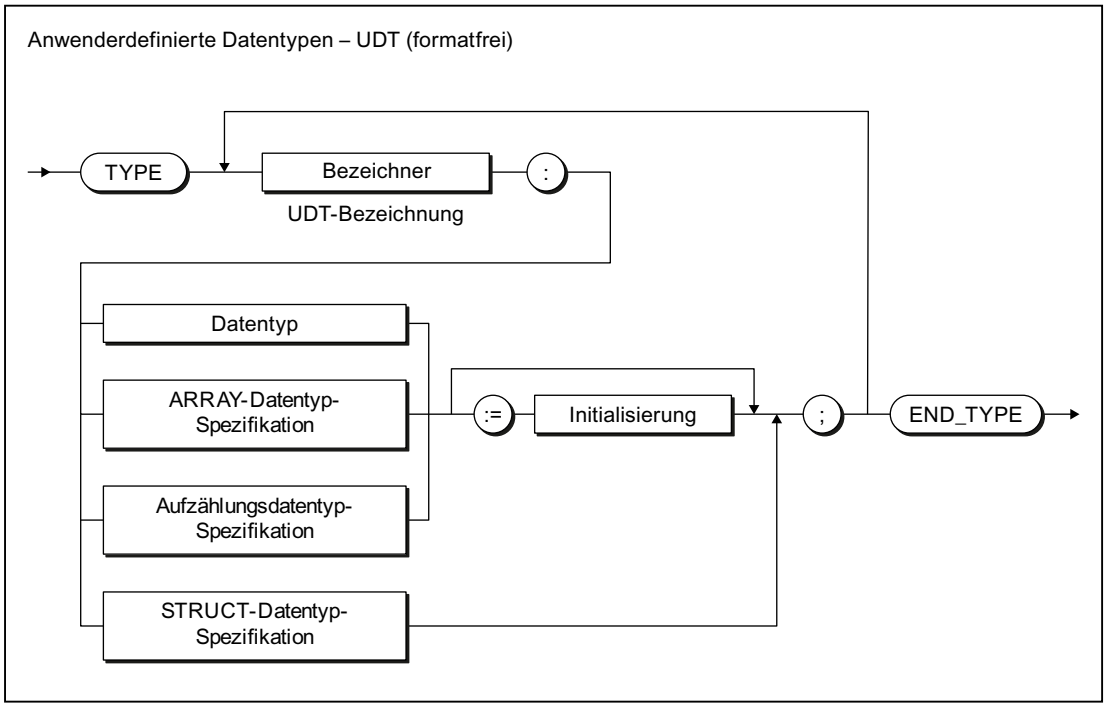


Bild 4-6 Syntax: Anwenderdefinierter Datentyp

Die Vereinbarung der UDT wird durch das Schlüsselwort TYPE eingeleitet.

Danach folgt für jeden zu vereinbarenden Datentyp, siehe Bild:

1. Bezeichnung (Name):
   Die Bezeichnung des Datentyps muss den Regeln für Bezeichner entsprechen.
2. Datentyp-Spezifikation:
   Unter dem Begriff *Datentyp* sind zusammengefasst, siehe Ableitung elementarer oder abgeleiteter Datentypen (Seite 123):
   - Elementare Datentypen
   - vorher vereinbarte UDT
   - TO-Datentypen
   - Systemdatentypen

   Weiterhin sind folgende Datentyp-Spezifikationen möglich:
   - ARRAY-Datentyp-Spezifikation, siehe Abgeleiteter Datentyp ARRAY – Feld (Seite 123)
   - Aufzählungsdatentyp-Spezifikation, siehe Abgeleiteter Datentyp Aufzählung – Enumerator (Seite 126)
   - STRUCT-Datentyp-Spezifikation, siehe Abgeleiteter Datentyp STRUCT – Struktur (Seite 128)

   Die Verweise in Klammern beziehen sich auf die nachfolgenden Abschnitte, in denen die jeweilige Datentyp-Spezifikation ausführlich beschrieben ist.
3. Optional Initialisierung:
   Sie können für den Datentyp einen Initialisierungswert festlegen. Wenn Sie später eine Variable dieses Datentyps deklarieren, wird der Initialisierungswert der Variablen zugewiesen.
   Ausnahme: Bei der STRUCT-Datentyp-Spezifikation erfolgt die Initialisierung für jede einzelne Komponente innerhalb der Datentyp-Spezifikation.
   Siehe auch Initialisierung von Variablen oder Datentypen (Seite 142).

Abgeschlossen wird die gesamte Vereinbarung der UDT mit dem Schlüsselwort END_TYPE. Innerhalb des Konstrukts TYPE / END_TYPE können Sie beliebig viele Datentypen erstellen. Die definierten Datentypen können Sie zur Deklaration von Variablen oder Parametern verwenden.

Es sind alle denkbaren Verschachtelungen zwischen den UDT möglich, so lange die Syntax aus dem Bild eingehalten wird. Beispielsweise können Sie als Datentyp-Spezifikation bereits definierte UDT oder geschachtelte Strukturen verwenden. Typdeklarationen selbst dürfen nur sequentiell und nicht verschachtelt angewendet werden.

Hinweis

Wie Sie Variablen und Parameter deklarieren, erfahren Sie in Übersicht aller Variablendeklarationen (Seite 141), wie Sie Wertzuweisungen mit UDT vornehmen, in Syntax der Wertzuweisung (Seite 148).

Nachfolgend werden die einzelnen Datentyp-Spezifikationen für den UDT erläutert und deren Anwendung anhand von Beispielen demonstriert.

4.4.2.2 Ableitung elementarer oder abgeleiteter Datentypen

Bei der Ableitung von Datentypen wird im Konstrukt TYPE / END_TYPE dem zu definierenden Datentyp ein elementarer bzw. anwenderdefinierter Datentyp (UDT) zugeordnet:

TYPE Bezeichner : elementarer Datentyp { := Initialisierung } ; END_TYPE

TYPE Bezeichner : anwenderdefinierter Datentyp { := Initialisierung } ; END_TYPE

Nach der Deklaration des Datentyps können Sie Variablen vom abgeleiteten Datentyp Bezeichner definieren. Dies entspricht der Deklaration von Variablen vom Datentyp *elementarer Datentyp*.

Tabelle 4-12 Beispiele für die Ableitung elementarer Datentypen

```
TYPE
    I1: INT;     // elementarer Datentyp
    R1: REAL;    // elementarer Datentyp
    R2: R1;      // abgeleiteter Datentyp (UDT)
END_TYPE
VAR
    // Diese Variablen können überall dort angewendet werden,
    // wo Variablen vom Typ INT angewendet werden können.
    myI1 : I1;
    myI2 : INT;   // kein abgeleiteter Datentyp!

    // Diese Variablen können überall dort angewendet werden,
    // wo Variablen vom Typ REAL angewendet werden können.
    myR1 : R1;
    myR2 : R2;
END_VAR
myI1 := 1;
myI2 := 2;
myR1 := 2.22;
myR2 := 3.33;
```

4.4.2.3 Abgeleiteter Datentyp ARRAY - Feld

Der abgeleitete Datentyp ARRAY oder Feld umfasst im Konstrukt TYPE / END_TYPE eine festgelegte Anzahl von Elementen eines einzigen Datentyps. Im Syntaxdiagramm des nächsten Bildes sehen Sie diesen Datentyp, der nach dem reservierten Bezeichner OF genauer spezifiziert wird.

TYPE Bezeichner : ARRAY-Datentyp-Spezifikation { := Initialisierung } ; END_TYPE

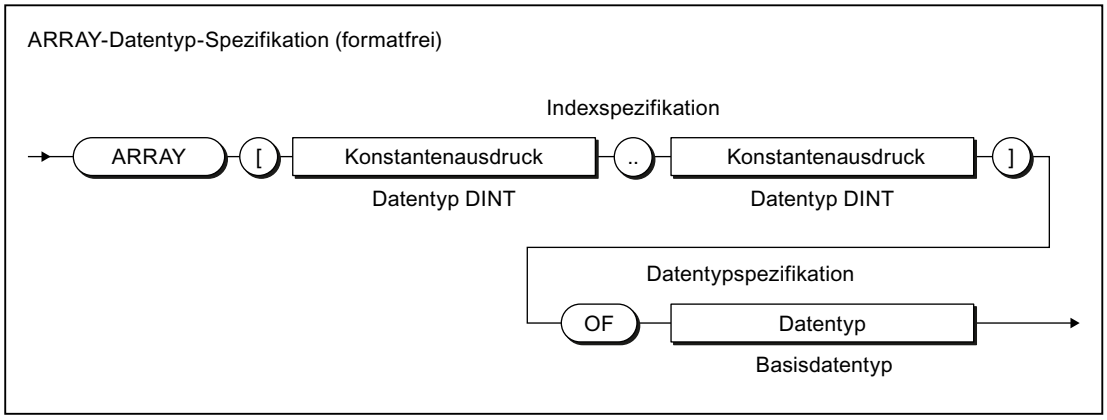


Bild 4-7 Syntax: ARRAY-Datentyp-Spezifikation

Die **Indexspezifikation** beschreibt die Grenzen des Feldes:

- Die Array-Grenzen geben den kleinst- und größtmöglichen Wert für den Index an. Sie können durch Konstanten oder Konstantenausdrücke vorgegeben werden; der Datentyp ist DINT (oder implizit nach DINT konvertierbar – siehe Konvertierung elementarer Datentypen (Seite 179)).
- Die Array-Grenzen müssen durch zwei Punkte getrennt angegeben werden.
- Die gesamte Indexspezifikation wird in eckigen Klammern eingeschlossen.
- Der Index selbst kann ein ganzzahliger Wert vom Datentyp DINT (oder implizit nach DINT konvertierbar – siehe Konvertierung elementarer Datentypen (Seite 179)) sein.

Hinweis

Wenn Array-Grenzen zur Laufzeit verletzt werden, führt dies zu einem Verarbeitungsfehler im Programm (siehe Funktionshandbuch "SIMOTION Basisfunktionen").

Mit der **Datentyp-Spezifikation** geben Sie den Datentyp der Feldelemente an (Basisdatentyp). Als Datentypen sind alle in diesem Kapitel aufgeführten Möglichkeiten zugelassen, z. B. auch ein selbstdefinierter Datentyp (UDT).

Die Größe des Speicherplatzes, den das ARRAY belegt, können Sie mit _sizeOf (in := *array-name*) bestimmen. Es gilt: _sizeOf (in := *array_of_type*) := _lengthIndexOf (in := *array_of_type*) * _sizeOf (in := *type*).

Die Syntax dieser Funktionen ist im Funktionshandbuch "SIMOTION Basisfunktionen" beschrieben.

Hinweis

Ab Version V4.2 des SIMOTION Kernels können Felder mit dynamischer Länge als Durchgangsparameter in Funktionen und Funktionsbausteinen deklariert werden. Siehe hierzu den entsprechenden Abschnitt (Seite 192) zum Deklarationsabschnitt von FB und FC (Seite 187).

Ein - und mehrdimensionale Felder

Man unterscheidet mehrere ARRAY-Typen:

- Der eindimensionale ARRAY-Typ ist eine Liste von Datenelementen, die in aufsteigender Reihenfolge angeordnet sind.
- Der zweidimensionale ARRAY-Typ ist eine Tabelle von Daten, die aus Zeilen und Spalten besteht. Die erste Dimension bezieht sich auf die Zeilennummer, die zweite auf die Spaltennummer.
- Der höherdimensionale ARRAY-Typ ist eine Erweiterung des zweidimensionalen ARRAY-Typs um weitere Dimensionen.

Tabelle 4-13 Beispiele für eindimensionale Felder

```
TYPE
    x : ARRAY[0..9] OF REAL;
    y : ARRAY[1..10] OF C1;
END_TYPE
```

Zweidimensionale Felder sind einer Tabelle mit Reihen und Spalten vergleichbar. Zwei- und mehrdimensionale Felder erstellen Sie über eine mehrstufige Typdeklaration, siehe Beispiel:

Tabelle 4-14 Beispiele für mehrdimensionale Felder

```
TYPE
    a      : ARRAY[1..3] OF INT;     // eindimensionales Feld
                                     // (3 Spalten)
    matrix1: ARRAY[1..4] OF a;       // zweidimensionales Feld
                                     // (4 Zeilen mit 3 Spalten)
    b      : ARRAY[4..8] OF INT;     // eindimensionales. Feld
                                     // (5 Spalten)
    matrix2: ARRAY[10..16] OF b;     // zweidimensionales Feld
                                     // (7 Zeilen mit 5 Spalten)
END_TYPE

VAR
    m : matrix1;     // Variable m vom Datentyp
                     // zweidimensionales Feld
    n : matrix2;     // Variable n vom Datentyp
                     // zweidimensionales Feld
END_VAR

m[4][3] := 9;      // Schreiben in matrix1 in Zeile 4, Spalte 3
n[16][8] := 10;    // Schreiben in matrix2 in Zeile 7, Spalte 5
```

Im Beispiel definieren Sie:

1. Die Spalten der Tabelle a[1] bis a[3] als eindimensionales Feld, das Ganzzahlen enthalten soll.
2. Die Zeilen der Tabelle matrix1[1] bis matrix1[4] ebenfalls als Feld, jedoch nehmen Sie als Datentyp-Spezifikation das soeben erzeugte Feld a mit den Spalten der Tabelle.
   Indem Sie als Datentyp-Spezifikation ein Feld angegeben haben, erzeugen Sie eine zweite Dimension. Weitere Dimensionen können Sie analog erzeugen.

Mit dem erzeugten Datentyp für die Tabelle deklarieren Sie eine Variable. Jede Dimension der Tabelle sprechen Sie über eckige Klammern an, in unserem Fall mit Angabe der Zeile und der Spalte.

Redeklaration

Datentyp-Definitionen von Felder (ARRAY) werden (auch in unterschiedlichen Quellen) als identisch erkannt, wenn nachstehende Bedingungen erfüllt sind:

1. Der Bezeichner des Datentyps ist identisch.
2. Die beiden Array-Grenzen sind identisch.
3. Der Datentyp der Feldelemente ist identisch.
4. Die optionale Feldinitialisierungsliste ist identisch (einschließlich Wiederholungsfaktoren und Konstantenausdrücken).

Initialisierung

Standardmäßig erhalten die Feldelemente den Initialisierungswert des Basisdatentyps. Optional kann der Initialisierungswert der Feldelemente durch Zuweisung einer Feldinitialisierungsliste, die in eckigen Klammern [ ] eingeschlossen ist, geändert werden, siehe "Initialisierung von Variablen oder Datentypen" (Seite 142).

4.4.2.4 Abgeleiteter Datentyp Aufzählung - Enumerator

Bei den Aufzählungsdatentypen wird im Konstrukt TYPE / END_TYPE dem zu definierenden Datentyp eine begrenzte Anzahl Bezeichner bzw. Namen zugeordnet:

TYPE Bezeichner: Aufzählungsdatentyp-Spezifikation { := Initialisierung } ; END_TYPE

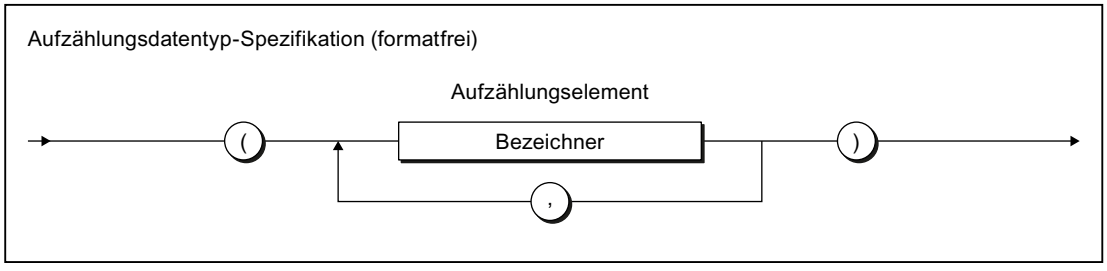


Bild 4-8 Syntax: Aufzählungsdatentyp-Spezifikation

Nach der Deklaration des Datentyps *Bezeichner* können Sie Variablen dieses Aufzählungsdatentyps definieren. Im Anweisungsteil dürfen Sie diesen Variablen nur Elemente aus der Liste der definierten Bezeichner (Aufzählungselemente) zuweisen.

Darüber hinaus können Sie den Datentyp direkt spezifizieren: Stellen Sie dem Aufzählungselement den Bezeichner des Aufzählungsdatentyps und das Zeichen "#" voran (siehe Tabelle *Beispiele für Aufzählungsdatentypen*).

Den ersten bzw. letzten Wert eines Aufzählungsdatentyps erhalten Sie mit *enum_type*#MIN bzw. *enum_type*#MAX, wobei *enum_type* der Bezeichner des Aufzählungsdatentyps ist.

Mit der Konvertierungsfunktion ENUM_TO_DINT erhalten Sie den numerischen Wert eines Aufzählungselements.

Beispiel

Tabelle 4-15 Beispiele für Aufzählungsdatentypen

```
TYPE
    C1 : (RED, GREEN, BLUE);
END_TYPE

VAR
    myC11, myC12, myC13 : C1;
END_VAR

myC11 := GREEN;
myC11 := C1#GREEN;
myC12 := C1#MIN;        // RED
myC13 := C1#MAX;        // BLUE
```

Hinweis

Aufzählungsdatentypen finden Sie auch als Systemdatentypen.

Aufzählungsdatentypen können Bestandteil von Strukturen sein, so dass Sie sie an beliebig tiefer Stelle in der anwenderdefinierten Datenstruktur finden können.

Redeklaration

Datentyp-Definitionen von Auszählungsdatentypen werden (auch in unterschiedlichen Quellen) als identisch erkannt, wenn nachstehende Bedingungen erfüllt sind:

1. Der Bezeichner des Datentyps ist identisch.
2. Alle Aufzählungselemente sind identisch
3. Die Reihenfolge der Aufzählungselemente ist identisch, so dass deren numerische Werte identisch sind.
4. Der optionale Initialisierungswert ist identisch.

Initialisierung

Standardmäßig erhält ein Aufzählungsdatentyp den 1. Wert der Aufzählung als Initialisierungswert. Optional kann der Initialisierungswert durch Zuweisung eines anderen Aufzählungselements geändert werden, siehe "Initialisierung von Variablen oder Datentypen" (Seite 142).

4.4.2.5 Abgeleiteter Datentyp STRUCT - Struktur

Der abgeleitete Datentyp STRUCT oder Struktur umfasst im Konstrukt TYPE / END_TYPE einen Bereich aus einer festen Anzahl von Komponenten, deren Datentypen verschieden sein dürfen:

TYPE Bezeichner : STRUCT-Datentyp-Spezifikation ; END_TYPE

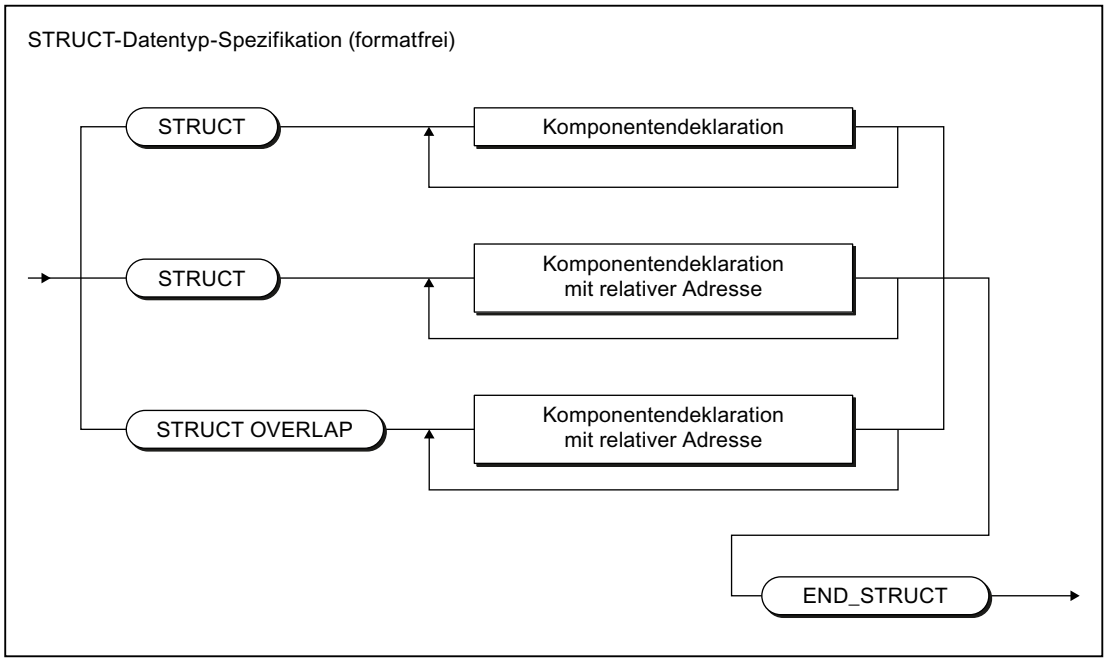


Bild 4-9 Syntax: STRUCT-Datentyp-Spezifikation

Komponentendeklaration

Gewöhnlich enthält eine Struktur zwischen den Schlüsselwörtern STRUCT und END_STRUCT die Definitionen der einzelnen Komponenten.

Die Syntax der Komponentendeklaration ist im nachfolgenden Bild dargestellt. Mit dieser Syntax werden die Komponenten in der Reihenfolge ihrer Deklaration angeordnet, Die relativen Adressen der Komponenten innerhalb der Struktur werden vom Compiler automatisch vergeben.

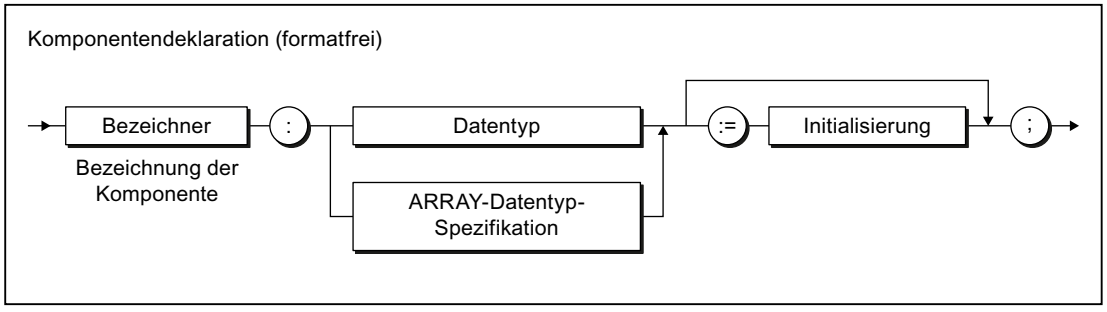


Bild 4-10 Syntax: Komponentendeklaration

Als Datentypen sind erlaubt:

- Elementare Datentypen
- Vorher vereinbarte UDT
- Systemdatentypen
- TO-Datentypen
- ARRAY-Datentyp-Spezifikation

Optional können Sie Initialisierungswerte der Komponenten vorgeben, siehe "Initialisierung" in diesem Abschnitt.

Hinweis

Innerhalb einer Komponentendeklaration ist die direkte Verwendung folgender Datentyp-Spezifikationen nicht möglich:

- STRUCT-Datentyp-Spezifikationen
- Aufzählungsdatentyp-Spezifikationen

Abhilfe:

Deklarieren Sie vorher einen UDT (anwenderdefinierten Datentyp) mit den obigen Spezifikationen und verwenden Sie diesen in der Komponentendeklaration.

Auf diese Weise können Sie STRUCT-Datentypen schachteln.

STRUCT-Datentypen finden Sie auch als Systemdatentypen.

Beispiel

Das Beispiel zeigt Ihnen eine Definition eines UDT und die Verwendung dieses Datentyps innerhalb einer Variablendeklaration.

Tabelle 4-16 Beispiele für den abgeleiteten Datentyp STRUCT

```
TYPE    // UDT Definition
    S1 : STRUCT
        var1 : INT;
        var2 : WORD := 16#AFA1;
        var3 : BYTE := 16#FF;
        var4 : TIME := T#1d_1h_10m_22s_2ms;
    END_STRUCT;
END_TYPE

VAR
    myS1 : S1;
END_VAR

myS1.var1 := -4;
myS1.var4 := T#2d_2h_20m_33s_2ms;
```

Redeklaration

Datentyp-Definitionen von Strukturen werden (auch in unterschiedlichen Quellen) als identisch erkannt, wenn nachstehende Bedingungen erfüllt sind:

1. Der Bezeichner des Datentyps ist identisch.
2. Alle Komponenten der Struktur sind identisch hinsichtlich:
   - Reihenfolge
   - Bezeichner
   - Datentypen bzw. ARRAY-Datentyp-Spezifikationen
   - Initialisierungswerte (einschließlich eventueller Feld- oder Strukturinitialisierungslisten)

Initialisierung

Standardmäßig erhält jede Komponente den Initialisierungswert ihres Datentyps. Optional kann der Initialisierungswert der Komponente durch Zuweisung einer entsprechenden Initialisierung geändert werden.

Bei Verwendung der Struktur in einer anderen Deklaration (z. B. Variablen- oder Datentypdeklaration) können die Initialisierungswerte einzelner Komponenten durch Zuweisung einer Strukturinitialisierungsliste, die in runden Klammern ( ) eingeschlossen ist, geändert werden.

Siehe "Initialisierung von Variablen oder Datentypen" (Seite 142).

Komponentendeklaration mit Angabe der relativen Adressen

Bei den Komponentendeklarationen können Sie die mit dem Schlüsselwort AT die relativen Adressen der Komponenten innerhalb dieser Struktur vorgeben, siehe nachfolgendes Syntaxdiagramm.

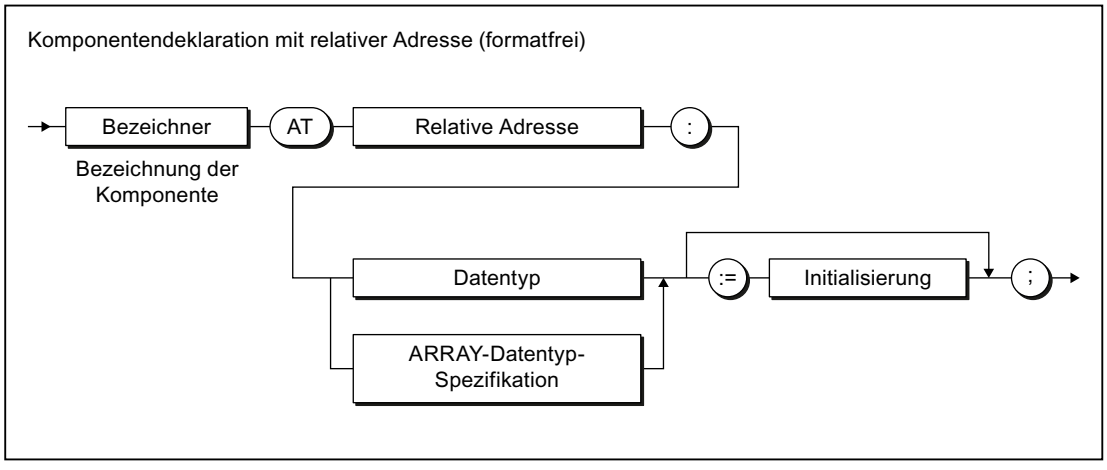


Bild 4-11 Syntax: Komponentendeklaration mit relativer Adresse

Die relative Adresse der Komponente (Byte-Offset) geben Sie wie folgt an:

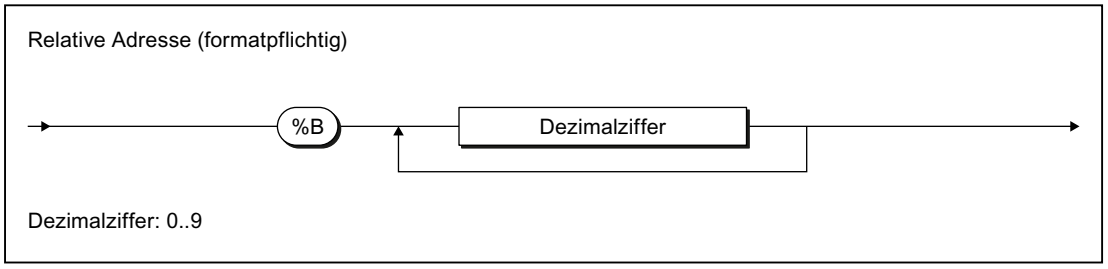


Bild 4-12 Syntax: Relative Adresse

Wenn Sie relative Adressen der Strukturkomponenten vorgeben, müssen Sie dies bei allen Komponenten dieser Struktur tun.

Die zulässigen relativen Adressen der Komponenten sind abhängig von deren Datentyp, siehe nachfolgende Tabelle:

Tabelle 4-17 Zulässige Adressen und Länge verschiedener Datentypen

Datentyp	Zulässige Adresse	Länge
Elementare Datentypen		
BOOL, BYTE, SINT, USINT	beliebig	1 Byte
WORD, INT, UINT	durch 2 teilbar	2 Byte
DWORD, DINT, UDINT, REAL TIME, DATE, TIME_OF_DAY (TOD)	durch 4 teilbar	4 Byte
LREAL DATE_AND_TIME (DT)	durch 8 teilbar	8 Byte
STRING	beliebige Adresse	deklarierte Länge + 2 Byte (Voreinstellung: 82 Byte)
Anwenderdefinierte Datentypen (UDT)		
Abgeleiteter Datentyp	entsprechend dem Basisdatentyp	
ARRAY (Feld)	entsprechend dem Basisdatentyp	entsprechend der Index-Spezifikation und dem Basisdatentyp
Aufzählung (Enumerator)	durch 4 teilbar	4 Byte
STRUCT (Struktur)	entsprechend dem größten Datentyp innerhalb der Struktur	entsprechend der Deklaration der Struktur
Weitere Datentypen		
StructAlarmId, StructTaskId	durch 4 teilbar	4 Byte
Systemdatentyp	entsprechend seiner Deklaration als STRUCT (Struktur) oder Aufzählung (Enumerator)	
TO-Datentyp, ANYOBJECT	durch 8 teilbar	8 Byte

Bei Verletzungen obiger Adressierungsregeln gibt der Compiler eine Fehlermeldung aus, in der auf den notwendiger Teiler hingewiesen wird.

Tabelle 4-18 Beispiel für Struktur mit Angabe der relativen Adresse

```
myLocatedStruct : STRUCT
    memb1 AT %B0 : INT;     // Relative Adresse 0
                            // Byte 0 und 1 belegt
    memb2 AT %B8 : REAL;    // Relative Adresse 8
                            // Byte 8 .. 11 belegt
                            // Byte 2 .. 7  frei (Lücke)
END_STRUCT
```

Bei Angabe der relativen Adressen ist die Deklarationsreihenfolge der Komponenten beliebig. Die Struktur im nachfolgenden Beispiel ist identisch mit derjenigen im vorhergehenden Beispiel.

Tabelle 4-19 Identische Struktur zum vorhergehenden Beispiel

```
myLocatedStruct : STRUCT
    memb2 AT %B8 : REAL;    // Relative Adresse 8
                            // Byte 8 .. 11 belegt
    memb1 AT %B0 : INT;     // Relative Adresse 0
                            // Byte 0 und 1 belegt
                            // Byte 2 .. 7  frei (Lücke)
END_STRUCT
```

Die Adressbereiche der Komponenten dürfen sich nicht überlappen, außer die Struktur wird mit dem Schlüsselwort STRUCT OVERLAP deklariert, siehe nachfolgenden Abschnitt.

Struktur mit überlappenden Adressbereichen (UNION)

Wenn Sie die Komponentendeklarationen mit Angabe der relativen Adresse (siehe oben) zwischen den Schlüsselwörtern STRUCT OVERLAP und END_STRUCT vornehmen, können sich Adressbereiche überlappen.

Für die Komponentendeklaration gilt das Syntaxdigramm für die Komponentendeklaration mit relativer Adresse (siehe oben). Für die zulässigen Adressen siehe Tabelle "Zulässige Adressen und Länge verschiedener Datentypen".

Tabelle 4-20 Bespiel für Struktur mit überlappenden Adressbereichen

```
myOverLapStruct : STRUCT OVERLAP
    memb1 AT %B0 : REAL;    // relative Adresse 0
                            // Byte 0 .. 3
    memb2 AT %B0 : INT;     // ebenfalls auf relativer Adresse 0
                            // Byte 0 und 1
    ar    AT %B0 : ARRAY [0..3] OF BYTE;
                            // nochmal relative Adresse 0
                            // Byte 0 .. 3
END_STRUCT
```

Wenn sich die Adressbereiche zweier Komponenten überlappen, sind in diesen Komponenten folgende Datentypen **nicht** zulässig:

- STRING
- ANYOBJECT
- TO-Datentyp

In Komponenten mit nicht überlappenden Adressbereichen sind sie zulässig. Folgende Deklaration ist z. B. erlaubt:

```
myOverLapStruct_2 : STRUCT OVERLAP
    dint_1  AT %B0 : DINT;
    udint_1 AT %B4 : UDINT;
    ar      AT %B0 : ARRAY [0..7] OF BYTE;
            // Byte 0 .. 7 überlappend
    pos_1   AT %B8 : PosAxis;
            // Byte 8 .. 15 nicht überlappend
END_STRUCT
```

Hinweis

Die Deklaration einer Struktur mit überlappendem ARRAY OF BYTE, wie in den Beispielen angegeben, ersetzt nicht die Marshalling-Funktionen. Die Anordnung der Bytes (Little Endian oder Big Endian) ist abhängig vom jeweiligen SIMOTION Gerät. Zur Byte-Anordnung Little Endian und Big Endian sowie die Marshalling-Funktionen siehe Funktionshandbuch SIMOTION Basisfunktionen.

Sichtbarkeit der Komponenten bei überlappenden Adressbereichen

Symbolinformationen (OPC-XML-Daten) werden nur für folgende Komponenten erzeugt:

- Komponenten ohne Überlappung von Adressbereichen
- Bei mehreren Komponenten, der Adressbereiche sich überlappen:
  Nur für die zuletzt deklarierte Komponente, die den jeweiligen Adressbereich nutzt. Der Compiler gibt eine Warnung für die verdeckten Komponenten aus.

Nur diese Komponenten sind sichtbar:

- Bei der Watchfunktion von IT DIAG
- Für die Funktionen *_exportUnitDataSet* und *_importUnitDataSet*, siehe Funktionshandbuch SIMOTION Basisfunktionen.
- Bei automatischen Anzeigen im SIMOTION SCOUT (z. B. Symbol-Browser (Seite 332), Status Programm (Seite 340)).

Hinweis

Verdeckte Komponenten können bei der Programmierung verwendet werden. Im ST-Editor werden Sie bei der Funktion "Automatisches Vervollständigen" (Seite 52) angezeigt.

Einzelne verdeckte Komponenten können Sie in einer Watchtabelle (Seite 335) beobachten, siehe die entsprechenden Ausführungen im Abschnitt "Watchtabelle einsetzen" (Seite 336).

Als Beispiel sind die sichtbaren Komponenten der folgenden Strukturdeklaration in der anschließenden Tabelle erläutert.

Die nachfolgenden Beispiele einer Strukturdeklaration erläutern das Verhalten für sichtbare und verdeckte Komponenten. Die sichtbaren Komponenten und die dazugehörigen relativen Adressen sind jeweils in der anschließenden Tabelle angegeben.

```
mystru : STRUCT OVERLAP
    a  AT %B0 : INT;    // Warnung: Für OPC-XML nicht sichtbar
    b  AT %B6 : INT;    // Warnung: für OPC-XML nicht sichtbar
    c  AT %B8 : INT;    // Sichtbar
    ar AT %B0 : ARRAY [0..7] OF BYTE; // Sichtbar
END_STRUCT
```

	Relative Adresse									
	0	1	2	3	4	5	6	7	8	9
Deklarierte Komponenten	a						b		c	
	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]			
Sichtbare Komponenten	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]	-	c	

Eine Änderung der Deklarationsreihenfolge wirkt sich auf die sichtbaren Komponenten aus:

```
mystru1 : STRUCT OVERLAP
    ar AT %B0 : ARRAY [0..7] OF BYTE; // Warnung: Für OPC-XML
                                      // nicht sichtbar
    a  AT %B0 : INT;    // Sichtbar
    b  AT %B6 : INT;    // Sichtbar
    c  AT %B8 : INT;    // Sichtbar
END_STRUCT
```

	Relative Adresse									
	0	1	2	3	4	5	6	7	8	9
Deklarierte Komponenten	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]			
	a						b		c	
Sichtbare Komponenten	a		-	-	-	-	b		c	

Initialisierung der Komponenten bei überlappenden Adressen

Bei nicht sichtbaren Komponenten werden Initialisierungswerte ignoriert. Der Compiler gibt eine entsprechende Warnung aus.

Sofern bei den sichtbaren Komponenten Initialisierungswerte angegeben sind, werden diese berücksichtigt. Diese Initialisierungswerte sind auch gültig für die gemeinsam genutzten Adressen verdeckter Komponenten. Für die Adressen verdeckter Komponenten, die nicht von anderen Komponenten überlappt werden, gilt: Sie werden mit dem Initialisierungswert, der bei der Deklaration des entsprechenden Datentyps festgelegt wurde, oder der Null vorbelegt.

Die nachfolgenden Beispiele einer Strukturdeklaration erläutern dieses Initialisierungsverhalten. Die Initialisierungswerte der jeweiligen relativen Adressen sind jeweils in der anschließenden Tabelle angegeben.

```
mystru_init : STRUCT OVERLAP
    a  AT %B0 : INT := 16#AAAA;     // Warnung Initialisierungswert
                                    // Warnung OPC-XML
    b  AT %B6 : INT := 16#BBBB;     // Warnung Initialisierungswert
                                    // Warnung OPC-XML
    c  AT %B8 : INT := 16#CCCC;
    ar AT %B0 : ARRAY [0..6] OF BYTE := [7(1)];
END_STRUCT
```

	Relative Adresse									
	0	1	2	3	4	5	6	7	8	9
Deklarierte Komponenten	a						b		c	
	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]			
Initialisierungswert	16#01	16#01	16#01	16#01	16#01	16#01	16#01	16#00	16#CCCC	

Eine Änderung der Deklarationsreihenfolge wirkt sich auf die Initialisierungwerte aus:

```
mystru_init1 : STRUCT OVERLAP
    ar AT %B0 : ARRAY [0..6] OF BYTE := [7(1)];
                                    // Warnung Initialisierungswert
                                    // Warnung OPC-XML
    a  AT %B0 : INT := 16#AAAA;
    b  AT %B6 : INT := 16#BBBB;
    c  AT %B8 : INT := 16#CCCC;
END_STRUCT
```

	Relative Adresse									
	0	1	2	3	4	5	6	7	8	9
Deklarierte Komponenten	ar[0]	ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]			
	a						b		c	
Initialisierungswert	16#AAAA		16#00	16#00	16#00	16#00	16#BBBB		16#CCCC	

4.4.3 Datentypen von Technologieobjekten

4.4.3.1 Beschreibung der Datentypen der Technologieobjekte

Sie können Variablen mit dem Datentyp eines Technologieobjekts (TO) deklarieren. Die nachfolgende Tabelle zeigt die Datentypen für die verfügbaren Technologieobjekte in den einzelnen Technologiepaketen.

Beispielsweise können Sie eine Variable mit dem Datentyp *posaxis* deklarieren und ihr eine entsprechende Instanz einer Positionierachse zuweisen. Eine solche Variable wird oft als Referenz bezeichnet.

Tabelle 4-21 Datentypen von Technologieobjekten (TO-Datentyp)

Technologieobjekt	Datentyp	enthalten im Technologiepaket
Drehzahlachse	DriveAxis	CAM, PATH[1], CAM_EXT
Externer Geber	ExternalEncoderType	CAM, PATH[1], CAM_EXT
Messtaster	MeasuringInputType	CAM, PATH[1], CAM_EXT
Nocken	OutputCamType	CAM, PATH[1], CAM_EXT
Nockenspur	_CamTrackType	CAM, PATH[1], CAM_EXT
Positionierachse	PosAxis	CAM, PATH[1], CAM_EXT
Gleichlaufachse	FollowingAxis	CAM, PATH[1], CAM_EXT
Gleichlaufobjekt	FollowingObjectType	CAM, PATH[1], CAM_EXT
Kurvenscheibe	CamType	CAM, PATH[1], CAM_EXT
Bahnachse[1]	_PathAxis	PATH[1], CAM_EXT
Bahnobjekt[1]	_PathObjectTpye	PATH[1], CAM_EXT
Festes Getriebe	_FixedGearType	CAM_EXT
Addierobjekt	_AdditionObjectType	CAM_EXT
Formelobjekt	_FormulaObjectType	CAM_EXT
Sensor	_SensorType	CAM_EXT
Reglerobjekt	_ControllerObjectType	CAM_EXT
Temperaturkanal	TemperatureControllerType	TControl
Allgemeiner Datentyp, dem jedes TO zuwiesen werden kann	ANYOBJECT	

[1] Ab Version V4.1 verfügbar.

Über Strukturen können Sie auf die Elemente von Technologieobjekten, Konfigurationsdaten und Systemvariablen zugreifen (siehe Funktionshandbuch SIMOTION Basisfunktionen).

Tabelle 4-22 Symbolische Konstanten für ungültige Werte der Datentypen von Technologieobjekten

Symbolische Konstante	Datentyp	Bedeutung
TO#NIL	ANYOBJECT	Ungültiges Technologieobjekt

Siehe auch

Vererbung der Eigenschaften bei Achsen (Seite 137)

Beispiele für die Verwendung von Datentypen der Technologieobjekte (Seite 137)

4.4.3.2 Vererbung der Eigenschaften bei Achsen

Vererbung bei den Achsen bedeutet, dass alle Datentypen, Systemvariablen und Funktionen des TO Drehzahlachse vollständig im TO Positionierachse enthalten sind. Die Positionierachse ist wiederum vollständig im TO Gleichlaufachse enthalten, die Gleichlaufachse im TO Bahnachse. Dies hat z. B. folgende Auswirkungen:

- Wenn eine Funktion oder ein Funktionsbaustein einen Eingangsparameter vom Datentyp driveAxis (Drehzahlachse) erwartet, können Sie beim Aufruf auch eine Positionierachse oder eine Gleichlaufachse oder eine Bahnachse verwenden.
- Wenn eine Funktion oder ein Funktionsbaustein einen Eingangsparameter vom Datentyp posAxis (Positionierachse) erwartet, können Sie beim Aufruf auch eine Gleichlaufachse oder eine Bahnachse verwenden.

4.4.3.3 Beispiele für die Verwendung von Datentypen der Technologieobjekte

Nachfolgend sehen Sie ein Beispiel für die optionale Verwendung einer Variablen vom Datentyp eines Technologieobjekts (ein Beispiel für die zwingende Verwendung einer Variablen vom TO-Datentyp finden Sie im Funktionshandbuch *SIMOTION Basisfunktionen*). Ein zweites Beispiel zeigt die Alternative, die auf die Verwendung einer Variablen vom TO-Datentyp verzichtet.

Es soll eine Achse im Hauptteil eines Programms mittels TO-Funktion frei geschaltet werden, um sie positionieren zu können. Nach der Positionierung soll die aktuelle Position der Achse mittels Strukturzugriff festgehalten werden.

Das erste Beispiel setzt eine Variable vom TO-Datentyp ein, um deren Verwendung zu demonstrieren.

Tabelle 4-23 Beispiel für die Verwendung eines Datentyps für Technologieobjekte

```
VAR
    myAxis : posAxis;   // Deklaration Variable für Achse
    myPos : LREAL;      // Variable für Position der Achse
    retVal : DINT;      // Variable für Rückgabewert der
                //TO-Funktion
END_VAR
myAxis := Axis1;       // Name Axis1 wurde bei der Konfiguration der
                       // Achse im Projektnavigator festgelegt.

// Aufruf der Funktion mit Variablen vom TO-Datentyp:
retVal := _enableAxis(axis := myAxis, commandId := _getCommandId());

// Achse wird positioniert.
retVal := _pos(axis := myAxis,
               position  := 100,
               commandId := _getCommandId());

// Abfrage der Position mittels Strukturzugriff
myPos := myAxis.positioningState.actualPosition;
```

Das zweite Beispiel verzichtet auf eine Variable vom TO-Datentyp.

Tabelle 4-24 Beispiel für die Verwendung eines Technologieobjekts

```
VAR
    myPos : LREAL; // Variable für Position der Achse
    retVal: DINT;  // Variable für Rückgabewert der TO-Funktion
END_VAR

// Aufruf der Funktion ohne Variable vom TO-Datentyp
// Name Axis1 wurde bei der Konfiguration der Achse
// im Projektnavigator festgelegt.
retVal := _enableAxis(axis := Axis1,
                      commandId := _getCommandId());

// Achse wird positioniert.
retVal := _pos(axis := Axis1
               position := 100,
               commandId := _getCommandId());

// Abfrage der Position mittels Strukturzugriff
myPos := Axis1.positioningState.actualPosition;
```

Einzelheiten zur Konfiguration von Technologieobjekten finden Sie in den Funktionsbeschreibungen SIMOTION Motion Control.

4.4.4 Systemdatentypen

Es stehen Ihnen eine Vielzahl von Systemdatentypen zur Verfügung, die Sie ohne vorherige Deklaration benutzen können. Auch jedes importierte Technologiepaket stellt eine Bibliothek von Systemdatentypen bereit.

Weitere Systemdatentypen (vorwiegend Aufzählungs- und STRUCT-Datentypen finden Sie

- bei Parametern zu den allgemeinen Standardfunktionen (siehe Funktionshandbuch *SIMOTION Basisfunktionen*)
- bei Parametern zu den allgemeinen Standardfunktionsbausteinen (siehe Funktionshandbuch *SIMOTION Basisfunktionen*)
- bei Systemvariablen der SIMOTION Geräte (siehe entsprechende Listenhandbücher)
- bei Parametern zu den Systemfunktionen der SIMOTION Geräte (siehe entsprechende Listenhandbücher)
- bei Systemvariablen und Konfigurationsdaten der Technologieobjekte (siehe entsprechende Listenhandbücher)
- bei Parametern zu den Systemfunktionen der Technologieobjekte (siehe entsprechende Listenhandbücher)

4.5 Variablendeklaration

Eine Variable definiert ein Datum mit variablem Inhalt, das in der ST-Quelle verwendet werden kann. Eine Variable besteht aus einem frei wählbaren Bezeichner (z. B. *myVar1*) und einem Datentyp (z. B. BOOL). Reservierte Bezeichner (siehe Reservierte Bezeichner (Seite 102)) dürfen nicht als Bezeichner verwendet werden.

4.5.1 Syntax der Variablendeklaration

Sie legen Variablen im Deklarationsabschnitt eines Quelldatei-Abschnitts immer nach dem gleichen Muster an:

1. Beginnen Sie einen Deklarationsblock mit dem entsprechenden Schlüsselwort (z. B. VAR, VAR_GLOBAL – siehe Übersicht aller Variablendeklarationen (Seite 141)).
2. Danach folgen die eigentlichen Variablendeklarationen (siehe Bild), von denen Sie beliebig viele erstellen können. Die Reihenfolge ist unbedeutend.
3. Beenden Sie den Deklarationsblock mit END_VAR.
4. Sie können weitere Deklarationsblöcke (auch mit demselben Schlüsselwort) erstellen.

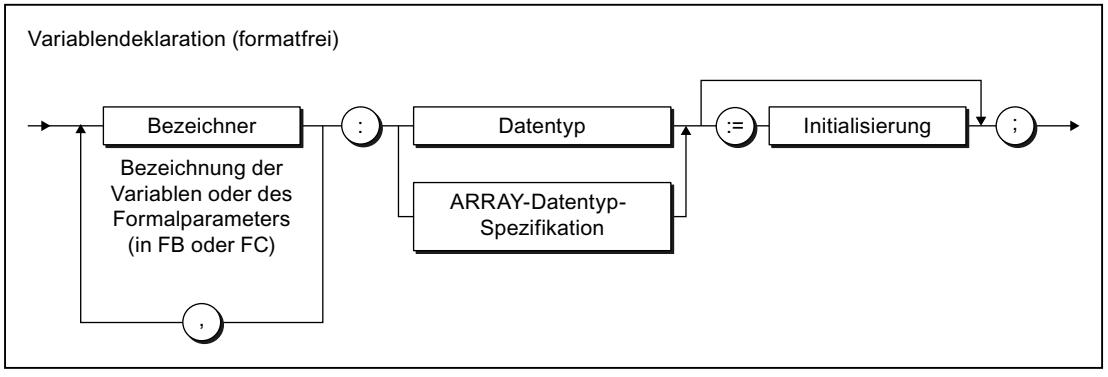


Bild 4-13 Syntax: Variablendeklaration

Beachten Sie auch:

- Der Variablenname muss ein Bezeichner sein, d. h. er darf nur Buchstaben, Ziffern oder den Unterstrich enthalten, aber keine Sonderzeichen.
- Als Datentyp sind zulässig:
  - Elementare Datentypen
  - UDT (anwenderdefinierte Datentypen)
  - Systemdatentypen
  - TO-Datentypen
  - ARRAY-Datentyp-Spezifikationen (Felder)
  - Bezeichnung eines Funktionsbausteins (Instanzdeklaration – siehe Funktionen und Funktionsbausteine aufrufen (Seite 193)).
- Die Zuweisung von Anfangswerten zu den Variablen ist bereits bei der Deklaration möglich. Dies nennt man Initialisierung (siehe Initialisierung von Variablen oder Datentypen (Seite 142)).

Abweichungen vom beschriebenen Muster finden Sie:

- bei der Deklaration von Konstanten (eine Konstante **muss** mit einem Wert initialisiert werden, siehe Konstanten (Seite 146)),
- bei Zugriffen auf das Prozessabbild (siehe Übersicht aller Variablendeklarationen (Seite 141)):
  - keine Variablendeklaration nötig beim absoluten PA-Zugriff,
  - keine Initialisierung erlaubt beim symbolischen PA-Zugriff.

Tabelle 4-25 Beispiele für Variablendeklarationen

```
VAR CONSTANT
    PI : REAL := 3.1415;
END_VAR

VAR
    // Deklaration einer Variablen ...
    var1 : REAL;
    // ... oder falls es mehrere Variablen desselben Typs gibt:
    var2, var3, var4 : INT;
    // Deklaration eines eindimensionalen Feldes:
    a1 : ARRAY[1..100] OF REAL;
    // Deklaration einer Zeichenkette (String):
    str1 : STRING[40];
END_VAR
```

4.5.2 Übersicht aller Variablendeklarationen

In den Variablen- und Parameterdeklarationen vereinbaren Sie Namen, Datentypen und Anfangswerte für Variablen. Diese Deklarationen führen Sie immer in folgenden Deklarationsabschnitten nachstehender Quelldatei-Abschnitte aus:

- Interfaceabschnitt
- Implementationsabschnitt
- POE (Programm, Funktion, Funktionsbaustein, Expression)

Der Quelldatei-Abschnitt bestimmt auch, welche Variablen Sie deklarieren können (siehe Tabelle), sowie deren Reichweite.

Mehr zu den Quelldatei-Abschnitten finden Sie in Gliederung der ST-Quelle (Seite 113) und Quelldatei-Abschnitte (Seite 212).

Tabelle 4-26 Schlüsselwörter für Deklarationsblöcke

Schlüsselwort	Bedeutung	Deklaration in folgenden Deklarationsabschnitten
VAR	Deklaration temporärer oder statischer Variablen siehe Variablenmodell (Seite 230)	beliebige POE
VAR_GLOBAL	Deklaration von Unit-Variablen siehe Variablenmodell (Seite 230)	Interfaceabschnitt Implementationsabschnitt
VAR_IN_OUT	Variablendeklaration Durchgangsparameter; die POE greift auf diese Variable direkt (über Referenz) zu und kann diese unmittelbar ändern. siehe Funktionen definieren (Seite 186) , Funktionsbausteine definieren (Seite 186)	Funktion Funktionsbaustein Expression
VAR_INPUT	Variablendeklaration Eingangsparameter; Wert wird von außerhalb geliefert, nicht innerhalb der POE änderbar. siehe Funktionen definieren (Seite 186) , Funktionsbausteine definieren (Seite 186)	Funktion Funktionsbaustein Expression
VAR_OUTPUT	Variablendeklaration Ausgangsparameter; Wert wird vom Funktionsbaustein nach außen geliefert siehe Funktionen definieren (Seite 186) , Funktionsbausteine definieren (Seite 186)	Funktionsbaustein

Schlüsselwort	Bedeutung	Deklaration in folgenden Deklarationsabschnitten
VAR_TEMP	Deklaration von temporären Variablen siehe Variablenmodell (Seite 230)	Programm Funktionsbaustein
RETAIN	Deklaration remanenter Variablen siehe Variablenmodell (Seite 230)	Nur als Zusatz zu VAR_GLOBAL im Interface- und Implementationsabschnitt
CONSTANT	Deklaration von Konstanten siehe Konstanten (Seite 146)	Nur als Zusatz: • zu VAR im FB, FC oder Programm • zu VAR_GLOBAL im Interface- oder Implementationsabschnitt

4.5.3 Initialisierung von Variablen oder Datentypen

Die Zuweisung von Anfangswerten zu den Variablen oder Datentypen innerhalb der Deklaration ist optional, siehe Syntax der Variablendeklaration (Seite 139) bzw. Syntax anwenderdefinierter Datentypen (Seite 121):

- Falls bei der Variablendeklaration keine Initialisierung angeben ist, weist der Compiler den Variablen automatisch den bei der Datentypdeklaration vorgegebenen Initialisierungswert zu.
  Bei Feldern (ARRAY) gilt: Die Feldelemente erhalten den Initialisierungswert des Basisdatentyps.
- Falls auch bei der Datentypdeklaration keine Initialisierung angeben ist, weist der Compiler den Variablen bzw. Datentypen den Wert Null zu.
  Ausnahme:
  - Bei Datentypen der Zeit: den Initialisierungswert des jeweiligen Datentyps,
  - Bei Aufzählungsdatentypen: 1. Wert der Aufzählung.
  - Bei Feldern (ARRAY): Die Feldelemente erhalten den Initialisierungswert des Basisdatentyps.
  - Bei Strukturen (STRUCT): Die Strukturkomponenten erhalten den Initialisierungswert des jeweiligen Datentyps.

Die Vorlegung einer Variablen oder eines anwenderdefinierten Datentyps mit Anfangswerten erfolgt mit einer Wertzuweisung (**:=**) nach der Datentypangabe:

- Weisen Sie elementaren Datentypen (bzw. von elementaren Datentypen abgeleiteten Datentypen) einen Konstantenausdruck gemäß Bild *Syntax: Konstantenausdruck* zu.
- Weisen Sie einem Feld (ARRAY) eine Feldinitialisierungsliste gemäß Bild *Syntax: Feldinitialisierungsliste* zu.

- Weisen Sie einer Struktur (STRUCT) eine Strukturinitialisierungsliste gemäß Bild *Syntax: Strukturinitialisierungsliste* zu (bei Verwendung der Struktur in anderen Deklarationen).
- Weisen Sie einem Aufzählungsdatentyp ein Aufzählungselement zu.

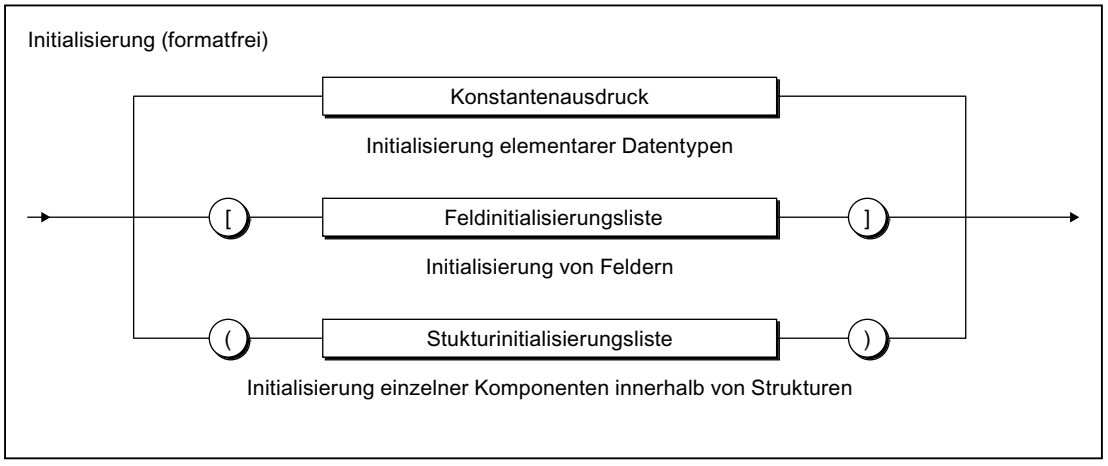


Bild 4-14 Syntax: Variableninitialisierung

Der einer Variablen zugewiesene Initialisierungswert wird zum Zeitpunkt der Übersetzung aus dem Konstantenausdruck berechnet. Zur Syntax des Konstantenausdrucks siehe Bild *Syntax: Konstantenausdruck*.

Beachten Sie, dass die Initialisierung einer Variablenliste (a1, a2, a3,.. : INT := ..) mit einem gemeinsamen Wert möglich ist. Sie müssen die Variablen in diesem Fall nicht einzeln initialisieren (a1 : INT := .. ; a2 : INT := .. ; usw.).

Hinweis

Die zur Initialisierung verwendeten Konstantenausdrücke werden im Datentyp der deklarierten Variablen bzw. im deklarierten Datentyp berechnet.

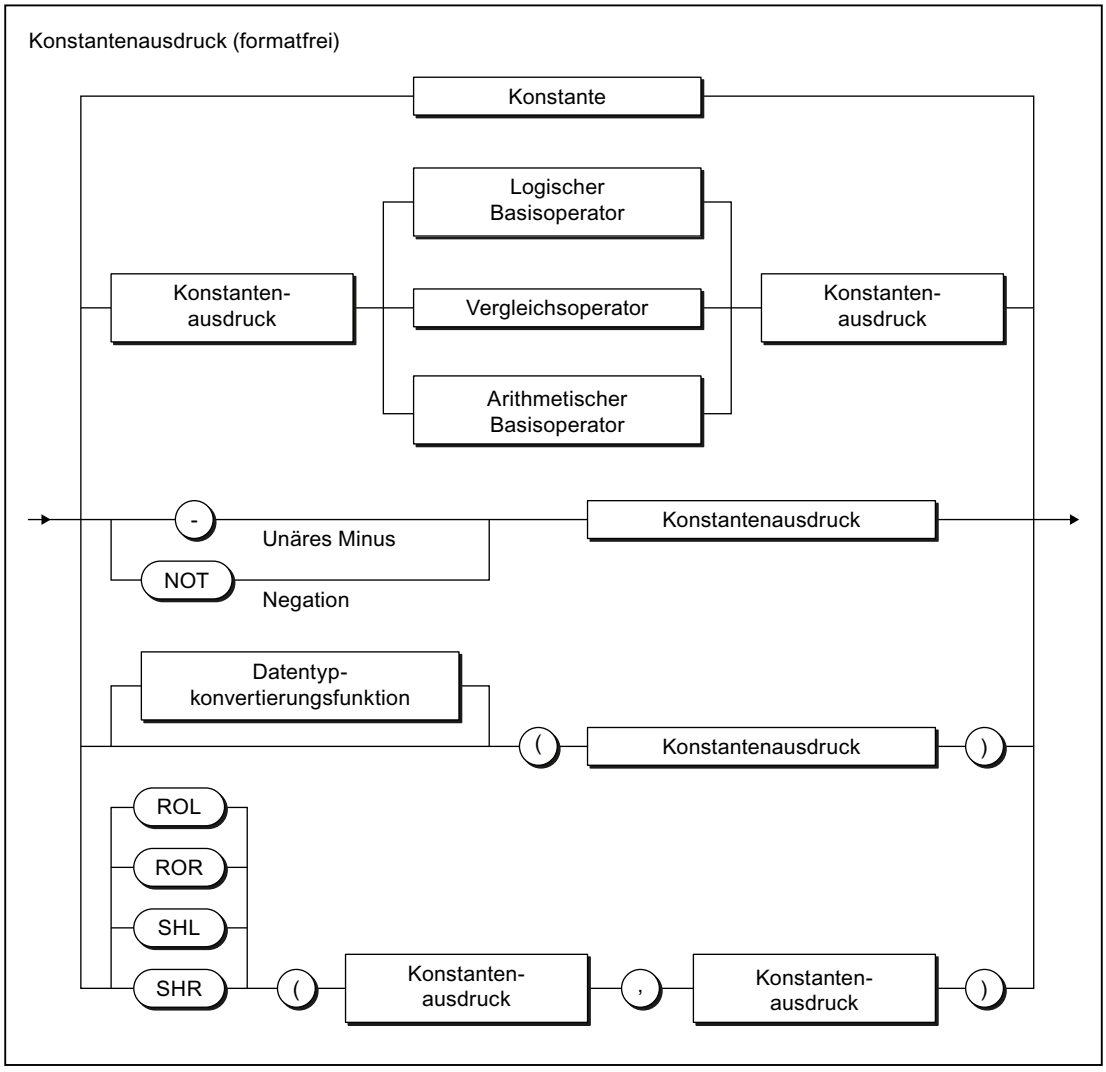


Bild 4-15 Syntax: Konstantenausdruck

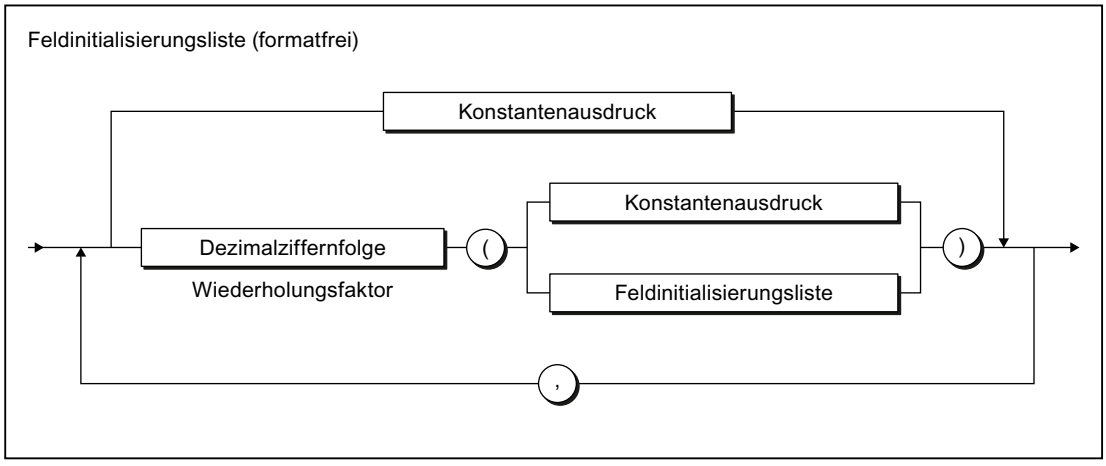


Bild 4-16 Syntax: Feldinitialisierungsliste

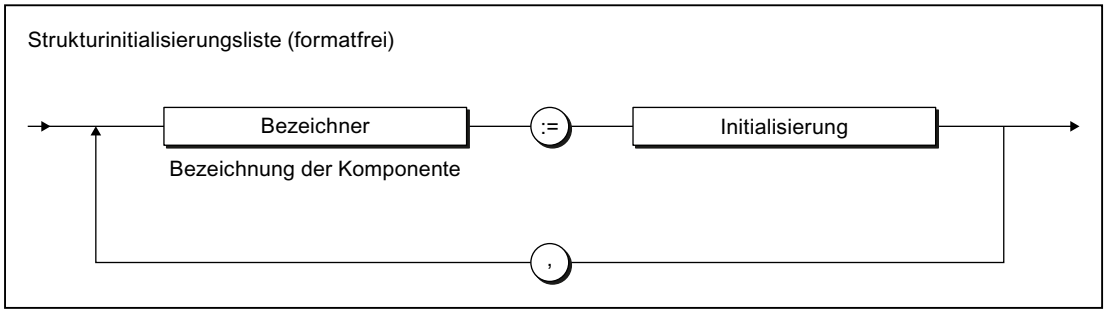


Bild 4-17 Syntax: Strukturinitialisierungsliste

Tabelle 4-27 Beispiele für Variableninitialisierungen

```
VAR
    // Deklaration einer Variablen ...
    var1 : REAL := 100.0;
    // ... oder falls es mehrere Variablen desselben Typs gibt:
    var2, var3, var4 : INT := 1;
    var5 : REAL := 3 / 2;
    var6 : INT := 5 * SHL(1, 4);
    myC1 : C1 := GREEN;
    array1 : ARRAY [0..4] OF INT := [1, 3, 8, 4, 0];
    array2 : ARRAY [0..5] OF DINT := [6 (7)];
    array3 : ARRAY [0..10] OF INT := [2 (2(3),3(1)),0];
                // entspricht [2(3),3(1),2(3),3(1)),0]
                // Initialisierung wie folgt:
                // Feldelemente 0, 1        mit 3;
                // Feldelemente 2, 3, 4     mit 1;
                // Feldelemente 5, 6        mit 3;
                // Feldelemente 7, 8, 9     mit 1;
                // Feldelement 10           mit 0
    myAxis : PosAxis := TO#NIL;
END_VAR
```

Tabelle 4-28 Beispiele für Datentypinitialisierungen

```
TYPE
    // Initialisierung eines abgeleiteten Datentyps
    type1 : REAL := 10.0;
    // Initialisierung eines Aufzählungsdatentyps
    cmyk_colour : (cyan, magenta, yellow, black) := yellow;
    // Initialisierung von Strukturen
    var_rgb_colour : STRUCT
            red, green, blue : USINT := 255;// weiß
        END_STRUCT;
    new_colour : var_rgb_colour := (red := 0, blue := 0);// grün
    myInt : INT := 9;
    myArray : ARRAY [0..5] OF myInt := [1, 2, 3];
                // Initialisierung wie folgt:
                // Feldelement  0          mit 1;
                // Feldelement  1          mit 2;
                // Feldelement  2          mit 3;
                // Feldelemente 3, 4, 5    mit 9
END_TYPE
```

Variablen vom Datentyp eines Technologieobjekts kann kein Initialisierungswert zugewiesen werden. Sie werden vom Compiler immer mit TO#NIL initialisiert.

Informationen, unter welchen Umständen Variablen initialisiert werden, finden Sie im Kapitel "Zeitpunkt der Variableninitialisierung" (Seite 246).

4.5.4 Konstanten

Konstanten sind Daten, die einen festen Wert besitzen, der sich zur Programmlaufzeit nicht ändern kann. Die Vereinbarung von Konstanten erfolgt analog der Variablendeklaration:

- für lokale Konstanten im Deklarationsabschnitt einer POE (siehe Bild *Syntax: Konstantenblock in einer POE* und *Syntax: Konstantendeklaration*).
- für Unit-Konstanten im Interface- oder Implementationsabschnitt der ST-Quelle (siehe Bild *Syntax: Unit-Konstanten im Interface- oder Implementationsabschnitt* und *Syntax: Konstantendeklaration*). Im Interfaceabschnitt deklarierte Unit-Konstanten können Sie in andere ST-Quellen importieren (siehe Variablenmodell (Seite 230)).

Der Quelldatei-Abschnitt bestimmt auch die Reichweite der Konstantendeklaration.

Bild 4-18 Syntax: Konstantenblock in einer POE

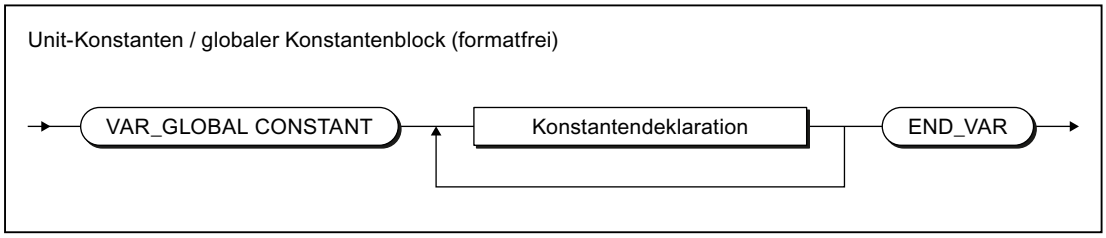


Bild 4-19 Syntax: Unit-Konstanten im Interface- oder Implementationsabschnitt

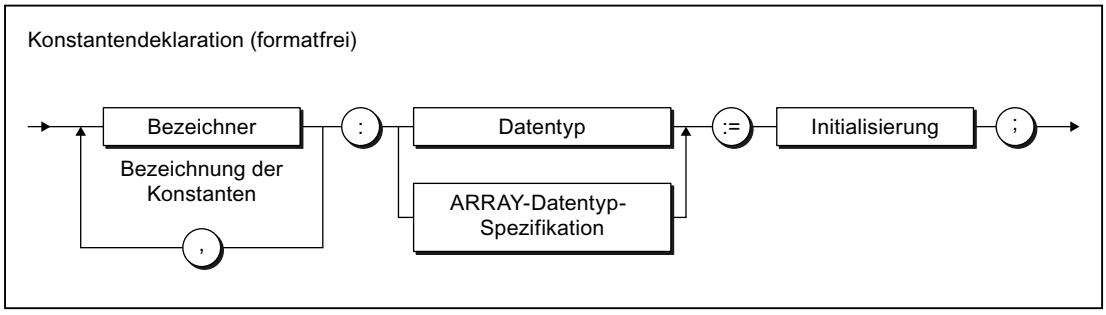


Bild 4-20 Syntax: Konstantendeklaration

Zur Initialisierung einer Konstanten siehe "Initialisierung von Variablen oder Datentypen (Seite 142)". Der einer Konstanten zugewiesene Wert wird zum Zeitpunkt der Übersetzung aus dem Konstantenausdruck berechnet.

Tabelle 4-29 Beispiele für Konstanten

```
VAR CONSTANT
    PI            : REAL := 3.1415;
    intConst      : INT  := 10;
    sintConst     : SINT := 0;
    dintConst     : DINT := 10_000;
    timeConst     : TIME := TIME#1h;
    strConst      : STRING[40] := 'Example of a string';
    Two_PI        : REAL := 2 * PI;
END_VAR
```

4.6 Wertzuweisungen und Ausdrücke

Sie haben bisher ganz selbstverständlich Wertzuweisungen mit der Zeichenfolge := erstellt, etwa beim Beispiel für eine Anweisung (siehe Tabelle *Beispiele für Anweisungen* in Anweisungen (Seite 114)) oder bei der Initialisierung von Variablen im Deklarationsabschnitt eines Quelldatei-Abschnitts.

Dies ist jedoch nur ein kleiner Ausschnitt der Möglichkeiten, die Sie bei der Bildung von Wertzuweisungen haben. Eine umfassendere Behandlung dieses Themas mit vielen Beispielen erfahren Sie an dieser Stelle.

Siehe auch

Hinweise zur Fehlervermeidung und zum effizienten Programmieren (Seite 324)

4.6.1 Wertzuweisungen

4.6.1.1 Syntax der Wertzuweisung

Eine Wertzuweisung dient dazu, einer Variablen den Wert eines Ausdrucks zuzuweisen. Der bisherige Wert wird überschrieben. Voraussetzung für die korrekte Bildung einer Wertzuweisung ist die Deklaration einer Variablen im Deklarationsabschnitt (siehe Syntax der Variablendeklaration (Seite 139)).

Wie das folgende Syntaxdiagramm zeigt, wird der Ausdruck auf der rechten Seite des Zuweisungszeichens := ausgewertet. Das Ergebnis wird in der Variablen abgespeichert, deren Namen auf der linken Seite des Zuweisungszeichens steht (Zielvariable). Das Bild zeigt auch die Gesamtheit der dabei zugelassenen Zielvariablen nach formellen Gesichtspunkten.

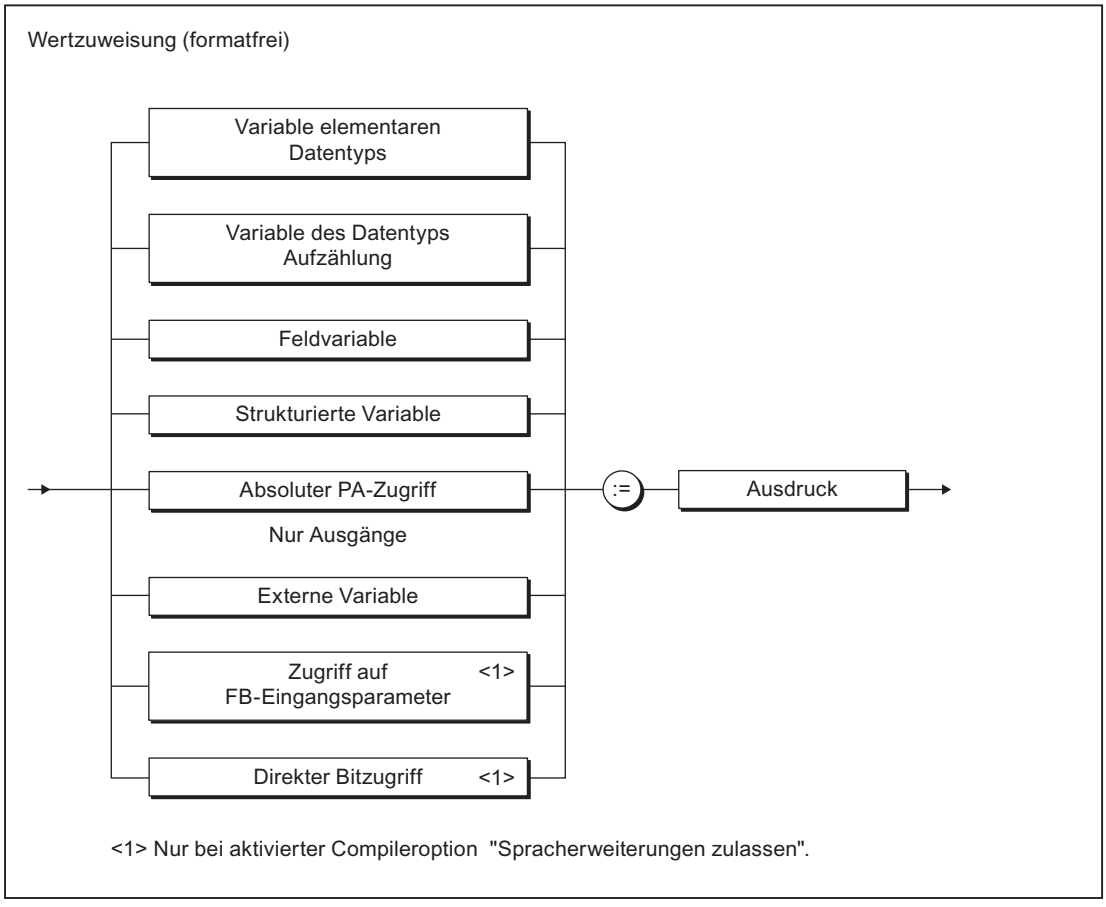


Bild 4-21 Syntax: Wertzuweisung

In den folgenden Ausführungen finden Sie Erläuterungen und Beispiele zu der linken Seite der Wertzuweisung:

- Wertzuweisungen mit Variablen eines elementaren Datentyps (Seite 150) ,
- Wertzuweisungen mit Variablen des abgeleiteten Datentyps Aufzählung (Enumerator) (Seite 154)
- Wertzuweisungen mit Variablen des abgeleiteten Datentyps ARRAY (Feld) (Seite 154)
- Wertzuweisungen mit Variablen des abgeleiteten Datentyps STRUCT (Struktur) (Seite 154)
- Wertzuweisungen mit absolutem PA-Zugriff (auf Adressen des Prozessabbilds), siehe: Absoluter Zugriff auf das feste Prozessabbild der BackgroundTask (Absoluter PA-Zugriff) (Seite 280).

Wie die rechte Seite einer Wertzuweisung, d. h. ein Ausdruck, gebildet wird, erfahren Sie in Ausdrücke (Seite 155).

4.6.1.2 Wertzuweisungen mit Variablen eines elementaren Datentyps

Ein Ausdruck mit einem elementaren Datentyp (Seite 116) kann einer Variablen zugewiesen werden, wenn eine der folgenden Bedingungen erfüllt ist:

- Ausdruck und Zielvariable haben den gleichen Datentyp.
  Beachten Sie den nachstehenden Hinweis zum Datentyp STRING (Seite 150).
- Der Datentyp des Ausdrucks kann durch implizite Konvertierung in den Datentyp der Zielvariablen gewandelt werden, siehe Konvertierung elementarer Datentypen (Seite 179) und *Funktionen zur Konvertierung von numerischen Datentypen und Bit-Datentypen* im Funktionshandbuch *SIMOTION Basisfunktionen*.

Beispiele

```
elemVar         := 3*3;
elemVar         := elemVar1;
```

Siehe auch

Wertzuweisungen mit Variablen eines Bitdatentyps (Seite 152)

4.6.1.3 Wertzuweisungen mit Variablen des elementaren Datentyps STRING

Zuweisungen zwischen Variablen vom Datentyp STRING

Zuweisungen zwischen Variablen vom Datentyp STRING (Zeichenketten), die mit verschiedenen Längen deklariert wurden, sind uneingeschränkt möglich. Wenn die deklarierte Länge der Zielvariablen kleiner ist als die aktuelle Länge der zugewiesenen Zeichenkette, wird die Zeichenkette auf die Länge der Zielvariablen abgeschnitten.

Ausnahme: Bei einer Durchgangszuweisung (Parameterübergabe zu einem Durchgangsparameter) gilt: Die deklarierte Länge der zugewiesenen Variablen (Aktualparameter) muss größer oder gleich sein der deklarierten Länge der Zielvariablen (formaler Durchgangsparameter). Siehe Parameterübergabe zu Durchgangsparametern (Seite 195).

Siehe auch Syntaxdiagramm Datentyp STRING (Seite 116):

Beispiele:

```
string20 := 'ABCDEFG';
string20 := string30;
```

Zugriff auf Elemente eines Strings

Die einzelnen Elemente eines Strings können wie die Elemente eines Feldes [1..n] angesprochen werden. Diese Elemente werden implizit in den elementaren Datentyp BYTE konvertiert. Dadurch sind Zuweisungen zwischen Stringelementen und Variablen des Datentyps BYTE möglich.

Beispiele:

```
byteVar      := string20[5];
string20[10] := byteVar;
```

Folgende Sonderfälle sind zu betrachten:

1. Bei der Zuweisung einer Variablen vom Datentyp BYTE an ein Stringelement (z. B. `stringVar[n] := byteVar`):
   - Das Stringelement, dem der Wert zugewiesen werden soll, liegt außerhalb der deklarierten Länge des Strings:
     Der String bleibt unverändert, TSI#ERRNO wird auf den Wert 1 gesetzt
   - Das Stringelement, dem der Wert zugewiesen werden soll, liegt außerhalb der belegten Länge des Strings (n > LEN(stringVar)), aber innerhalb der deklarierten Länge:
     Die Länge des Strings wird angepasst, die Stringelemente zwischen LEN(stringvar) und n werden auf $00 gesetzt,
2. Bei der Zuweisung eines Stringelements an eine Variable vom Datentyp BYTE (`byteVar := stringVar[n]`):
   - Das Stringelement, das der Variablen zugewiesen werden soll, liegt außerhalb der belegten Länge des Strings (n > LEN(stringVar)):
     Die Variable wird auf den Wert 16#00 gesetzt, TSI#ERRNO auf den Wert 2.

Bearbeiten von Strings

Zur Stringbearbeitung, wie Zusammenfügen von Strings, Ersetzen oder Extrahieren von Zeichen, stehen verschiedene Systemfunktionen zu Verfügung, siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

Konvertieren zwischen Zahlen und Strings

Zur Konvertierung zwischen Variablen numerischer Datentypen und Strings stehen verschiedene Systemfunktionen zur Verfügung, siehe Konvertierung elementarer Datentypen (Seite 179) und Funktionshandbuch *SIMOTION Basisfunktionen*.

4.6.1.4 Wertzuweisungen mit Variablen eines Bitdatentyps

Zugriff auf einzelne Bits einer Variablen eines Bitdatentyps

Sie können auf die einzelnen Bits einer Variablen vom Datentyp BYTE, WORD oder DWORD zugreifen:

- Mit Standardfunktionen, siehe Funktionshandbuch SIMOTION Basisfunktionen:
  Mit den Funktionen _getBit, _setBit und _toggleBit können Sie ein beliebiges Bit eines Bitstrings lesen, schreiben bzw. invertieren.
  Die Nummer des Bits können Sie über eine Variable vorgeben.
- Mit direktem Bitzugriff:
  Das Bit der Variablen, auf das sie zugreifen wollen, können Sie durch einen Punkt getrennt hinter der Variablen als Konstante angeben.
  Die Nummer des Bits können Sie nur über eine Konstante vorgeben.
  Um diese Möglichkeit nutzen zu können. müssen Sie die Compileroption "Spracherweiterungen zulassen" aktivieren, siehe Globale Einstellungen des Compilers (Seite 64) und Lokale Einstellungen des Compilers (Seite 67).

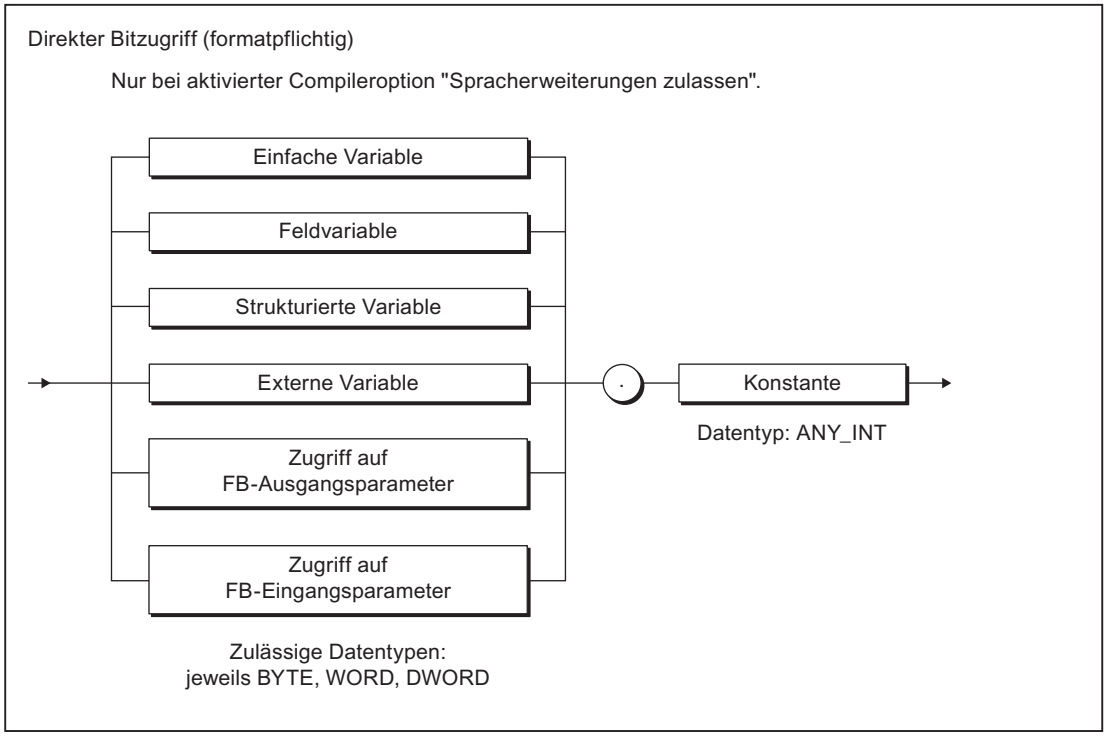


Bild 4-22 Syntax: Direkter Bitzugriff

Tabelle 4-30 Beispiel für direkten Bitzugriff

```
// Nur mit Compileroption "Spracherweiterungen zulassen"
FUNCTION f : VOID
    VAR CONSTANT
        BIT_7 : INT := 7;
    END_VAR
    VAR
        dw : DWORD;
        b  : BOOL;
    END_VAR
    b := dw.BIT_7;  // Zugriff auf Bit 7
    b := dw.3;      // Zugriff auf Bit 3
//  b := dw.33;     // Übersetzungsfehler:
                    // Bit 33 nicht erlaubt.
    dw.BIT_7 := b   // Zugriff auf Bit 7
    dw.3     := NOT dw.3  // Zugriff auf Bit 3
END_FUNCTION
```

Hinweis

Der Zugriff auf die Bits einer I/O-Variablen oder Systemvariablen kann durch andere Tasks unterbrochen werden. Die Konsistenz ist dadurch nicht gewährleistet.

Bearbeiten von Variablen der Bitdatentypen

Sie können:

1. Mehrere Variablen des gleichen Datentyps zu einer Variablen eines übergeordneten Datentyps zusammenfassen (z. B. 2 Variablen vom Datentyp BYTE in 1 vom Datentyp WORD). Hierzu stehen verschiedene Systemfunktionen zur Verfügung, z. B. WORD_FROM_2BYTE.
2. Eine Variable in mehrere Variablen eines untergeordneten Datentyps zerlegen (z. B. 1 Variable vom Datentyp DWORD in 4 vom Datentyp BYTE). Hierzu stehen Ihnen verschiedene Systemfunktionsbausteine zur Verfügung, z. B. DWORD_TO_4BYTE.
3. Die Bits innerhalb einer Variablen rotieren oder schieben. Hier stehen Ihnen die Bitsting-Standardfunktionen ROL, ROR, SHL und SHR zur Verfügung.

Diese Systemfunktionen und Systemfunktionsbausteine sind im Funktionshandbuch *SIMOTION Basisfunktionen* beschrieben.

Logische Operatoren

Variablen der Bittdatentypen können Sie mit logischen Operatoren verknüpfen, siehe Logische Ausdrücke und Bitfolgeausdrücke (Seite 164).

4.6.1.5 Wertzuweisungen mit Variablen des abgeleiteten Datentyps Aufzählung (Enumerator)

Jeder Ausdruck und jede Variable des abgeleiteten Datentyps Aufzählung (siehe auch: Abgeleiteter Datentyp Aufzählung - Enumerator (Seite 126)) kann einer anderen typgleichen Variablen zugewiesen werden.

```
type1        := BLUE;
```

4.6.1.6 Wertzuweisungen mit Variablen des abgeleiteten Datentyps ARRAY (Feld)

Ein Feld besteht aus mehreren Dimensionen und Feldelementen, die alle vom selben Typ sind (siehe auch: Abgeleiteter Datentyp ARRAY - Feld (Seite 123)).

Für die Zuweisung von Feldern an eine Variable gibt es mehrere Varianten, Sie können komplette Felder, einzelne Elemente oder Teilfelder zuweisen:

- Ein komplettes Feld ist einem anderen Feld zuweisbar, wenn sowohl die Datentypen der Komponenten als auch die Feldgrenzen (kleinst- und größtmögliche Feldindizes) übereinstimmen. Gültige Zuweisungen sind:

```
array_1     := array_2;
```

- Ein einzelnes Feldelement wird mit dem Feldnamen, gefolgt vom Indexwert in eckigen Klammern, angesprochen. Ein Index muss ein arithmetischer Ausdruck vom Datentyp SINT, USINT, INT, UINT oder DINT sein.

```
elem1        := array [i];
array_1 [2]  := array_2 [5];
array [j]    := 14;
```

- Eine Wertzuweisung für ein zulässiges Teilfeld erhalten Sie, indem Sie für jede Dimension des Feldes ein Paar eckige Klammern von rechts beginnend weg lassen. Damit sprechen Sie einen Teilbereich an, dessen Dimensionsanzahl gleich der Anzahl der übrig gebliebenen Indizes ist (siehe Beispiel unten).
  Daraus ergibt sich, dass Sie in einer Matrix zwar Zeilen und einzelne Komponenten, aber keine Spalten geschlossen referenzieren können (geschlossen heißt von...bis). Gültige Zuweisungen sind:

```
matrix1 [i]   := matrix2 [k];
array1        := matrix2 [k];
```

4.6.1.7 Wertzuweisungen mit Variablen des abgeleiteten Datentyps STRUCT (Struktur)

Variablen eines anwenderdefinierten Datentyps, die STRUCT-Datentyp-Spezifikationen enthalten, werden als strukturierte Variablen bezeichnet (siehe auch Abgeleiteter Datentyp STRUCT - Struktur (Seite 128). Sie können entweder für eine komplette Struktur oder eine Komponente dieser Struktur stehen.

Gültige Angaben für eine Strukturvariable sind:

```
struct1             // Bezeichner für eine Struktur
struct1.elem1       // Bezeichner für Strukturkomponente
struct1.array1      // Bezeichner eines einfachen Feldes
                    // innerhalb einer Struktur
struct1.array1[5]   // Bezeichner einer Feldkomponente
                    // innerhalb einer Struktur
```

Für die Zuweisung von Strukturen an eine Variable gibt es zwei Varianten, Sie können komplette Strukturen oder Strukturkomponenten referenzieren:

- Eine gesamte Struktur ist einer anderen Struktur nur dann zuweisbar, wenn die Strukturkomponenten sowohl in ihren Datentypen als auch in ihren Namen übereinstimmen. Eine gültige Zuweisung ist:

```
struct1 := struct2;
```

- Sie können jeder Strukturkomponente eine typverträgliche Variable, einen typverträglichen Ausdruck oder eine andere Strukturkomponente zuweisen.
  Gültige Zuweisungen sind:

```
struct1.elem1       := Var1;
struct1.elem1       := 20;
struct1.elem1       := struct2.elem1;
struct1.array1      := struct2.array1;
struct1.array1[10]  := 100;
```

Hinweis

Auch beim Zugriff auf Ausgangsvariablen eines Funktionsbausteins, d. h. auf seine Ergebnisse, verwenden Sie strukturierte Variablen im Format *FB-Instanzname.Ausgangsparameter*, z. B. *meinKreis.Umfang*. Näheres zu Funktionsbausteinen siehe Erläuterungen in Funktionen definieren (Seite 186) und Funktionsbausteine definieren (Seite 186).

Eine weitere Anwendung von strukturierten Variablen finden Sie bei Zugriffen auf TO-Variablen und auf Variablen des Grundsystems.

4.6.2 Ausdrücke

Ein Ausdruck steht für einen Wert, der bei der Übersetzung oder zur Laufzeit des Programms berechnet wird. Er besteht aus Operanden (z. B. Konstanten, Variablen oder Funktionswerten) und Operatoren (z. B. *, /, +, -).

Die Datentypen der Operanden und die beteiligten Operatoren bestimmen den Typ des Ausdrucks.

ST unterscheidet:

- Arithmetische Ausdrücke (Seite 159)
- Vergleichsausdrücke (Seite 162)
- Logische Ausdrücke und Bitfolgeausdrücke (Seite 164)

4.6.2.1 Ergebnis eines Ausdrucks

Das Ergebnis eines Ausdrucks können Sie

- einer Variablen zuweisen,
- als Bedingung für eine Kontrollanweisung verwenden,
- als Parameter für den Aufruf einer Funktion oder eines Funktionsbausteins verwenden.

Der Datentyp des Ergebnisses eines arithmetischen Ausdrucks oder eines Bitfolgeausdrucks wird von den Datentypen der Operanden bestimmt. Es wird der kleinste gemeinsame Datentyp herangezogen, in den beide Operanden implizit konvertiert werden können.

Eine Wertzuweisung des Ausdrucks an eine Variable (bzw. Parameter einer Funktion oder eines Funktionsbausteins) ist nur in folgenden Fällen möglich:

- Der berechnete Ausdruck und die zuzuweisende Variable haben den gleichen Datentyp.
- Der Datentyp des berechneten Ausdrucks kann implizit in den Datentyp der zuzuweisenden Variablen konvertiert werden.

Zu dieser möglichen Fehlerquelle und deren Abhilfe siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

Hinweis

Ausdrücke, die ausschließlich folgende Elemente enthalten, können zur Initialisierung von Variablen und zur Indexspezifikation bei ARRAY-Deklarationen verwendet werden (Initialisierungsausdrücke – siehe Bild *Syntax: Konstantenausdruck* in Initialisierung von Variablen oder Datentypen (Seite 142)):

- Konstanten
- Arithmetische Basisoperationen
- Logische und Vergleichsoperationen
- Bitstring-Standardfunktionen

Die zur Initialisierung verwendeten Konstantenausdrücke werden im Datentyp der deklarierten Variablen bzw. im deklarierten Datentyp berechnet.

4.6.2.2 Auswertungsreihenfolge eines Ausdrucks

Die Auswertungsreihenfolge eines Ausdrucks ist abhängig von:

- der Priorität der beteiligten Operatoren,
- der Links-Rechts-Reihenfolge,
- der vorgenommenen Klammerung (bei Operatoren gleicher Priorität).

Die Verarbeitung der Ausdrücke erfolgt nach bestimmten **Regeln**:

- Die Operatoren werden entsprechend ihrer Priorität bearbeitet (siehe Tabelle in Rangfolge der Operatoren (Seite 166)).
- Operatoren gleicher Priorität werden von links nach rechts bearbeitet.
- Das Voranstellen eines Minuszeichens vor einen Bezeichner ist gleichbedeutend mit der Multiplikation mit -1.
- Arithmetische Operatoren dürfen nicht direkt aufeinander folgen. Deshalb ist der Ausdruck *a * -b* ungültig, aber *a * (-b)* erlaubt.
- Das Setzen von Klammerpaaren setzt den Operatorenvorrang außer Kraft, d. h. die Klammerung hat die höchste Priorität.
- Ausdrücke in Klammern werden als einzelne Operanden betrachtet und immer als Erstes ausgewertet.
- Die Anzahl von öffnenden Klammern muss mit der Anzahl von schließenden Klammern übereinstimmen.
- Arithmetische Operationen können nicht auf Zeichen oder logische Daten angewendet werden. Deshalb sind Ausdrücke wie *(n<=0) + (n<0)* falsch.

Tabelle 4-31 Beispiele für Ausdrücke

```
testVar           // Operand
A AND (B)         // logischer Ausdruck
A AND (NOT B)     // logischer Ausdruck mit Verneinung
(C) < (D)         // Vergleichsausdruck
3+3*4/2           // arithmetischer Ausdruck
```

4.6.3 Operanden

Definition

Operanden sind Objekte, mit denen Ausdrücke gebildet werden können. Operanden lassen sich durch das Syntaxdiagramm darstellen:

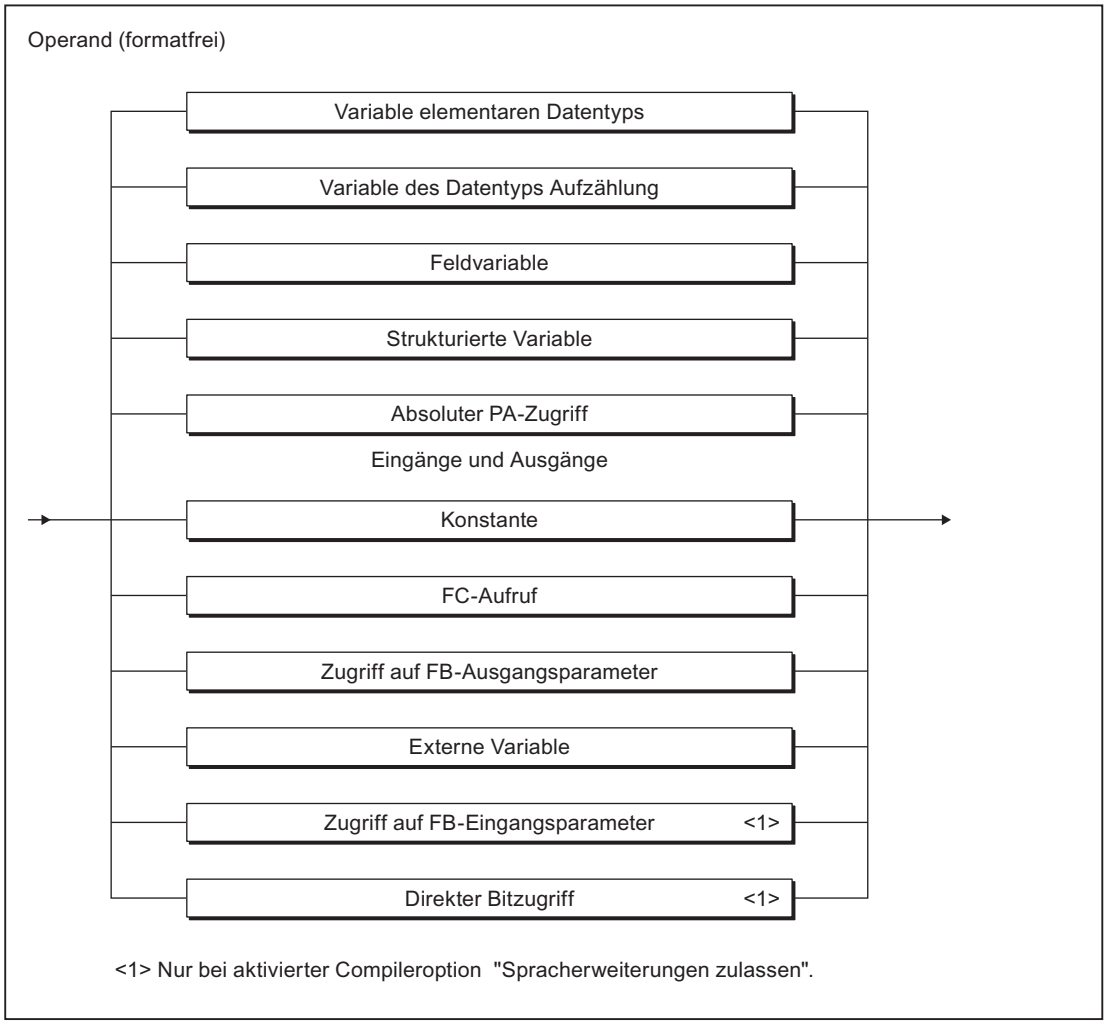


Bild 4-23 Syntax: Operand

Tabelle 4-32 Beispiele für Operanden

```
intVar
5
%I4.0
PI
NOT TRUE
axis1.motionStateData.actualVelocity
```

4.6.4 Arithmetische Ausdrücke

Ein arithmetischer Ausdruck ist ein mit arithmetischen Operatoren gebildeter Ausdruck. Solche Ausdrücke ermöglichen die Verarbeitung von numerischen Datentypen.

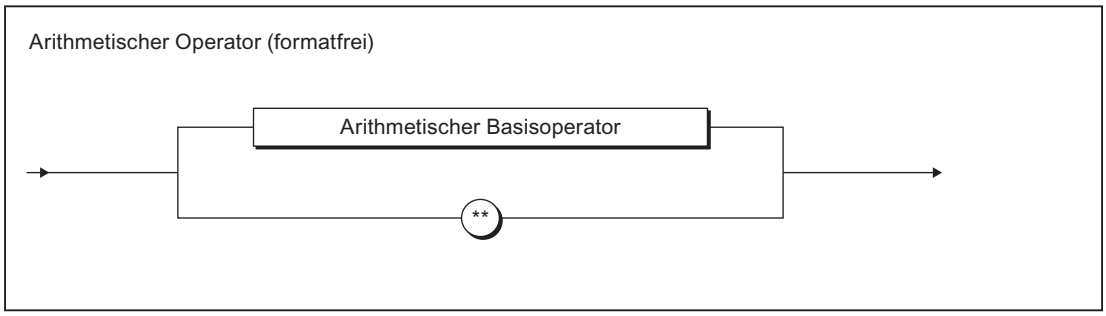


Bild 4-24 Syntax: arithmetischer Operator

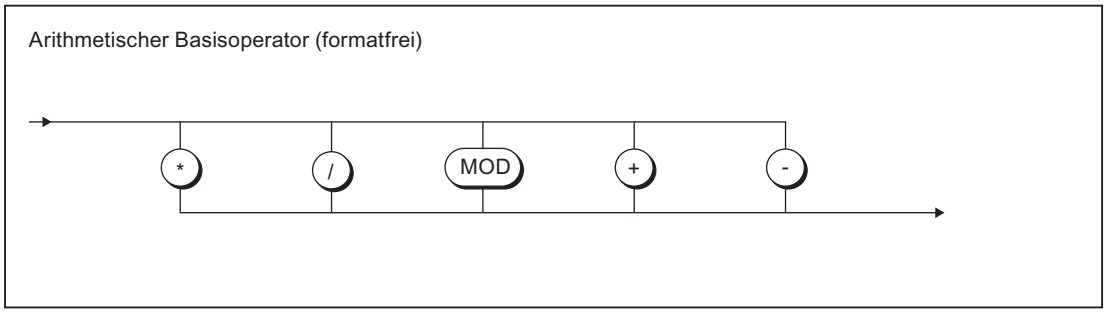


Bild 4-25 Syntax: arithmetischer Basisoperator

Die nachfolgende Tabelle zeigt für jede arithmetische Operation:

- den arithmetischen Operator
- die zulässigen Datentypen der Operanden
- den Datentyp des Ergebnisses.

Dabei werden z. T die allgemeinen Datentypen (Seite 119) verwendet.

Hinweis

Weitere Operationen sind mit numerischen Standardfunktionen möglich, siehe *Numerische Standardfunktionen* im Funktionshandbuch *SIMOTION Basisfunktionen*.

Es empfiehlt sich, negative Zahlen wegen der Übersichtlichkeit in Klammern zu setzen, auch dort, wo es nicht notwendig ist.

Die arithmetischen Operatoren werden gemäß ihrer Rangfolge (Seite 166) behandelt.

Tabelle 4-33 Arithmetische Operatoren

Operation	Operator	Datentyp		
		1. Operand	2. Operand	Ergebnis[1]
Potenzierung (siehe auch Funktion EXPT)	**	ANY_REAL[2]	ANY_REAL	ANY_REAL[3]
unäres Minus	-	ANY_NUM	(kein)	ANY_NUM
Multiplikation	*	ANY_NUM	ANY_NUM	ANY_NUM
		ANY_BIT[4]	ANY_BIT[4]	ANY_BIT
		TIME	ANY_NUM	TIME
Division	/	ANY_NUM	ANY_NUM[5]	ANY_NUM
		ANY_BIT[4]	ANY_BIT[4] [5]	ANY_BIT
		TIME	ANY_NUM[5]	TIME
		TIME	TIME[5]	UDINT
Modulo- Division	MOD	ANY_INT	ANY_INT[5]	ANY_INT
		ANY_BIT[4]	ANY_BIT[4] [5]	ANY_BIT
Addition	+	ANY_NUM	ANY_NUM	ANY_NUM
		ANY_BIT[4]	ANY_BIT[4]	ANY_BIT
		TIME	TIME	TIME[6]
		TOD	TIME	TOD[6]
		DT	TIME	DT[7]
Subtraktion	-	ANY_NUM	ANY_NUM	ANY_NUM
		ANY_BIT[4]	ANY_BIT[4]	ANY_BIT
		TIME	TIME	TIME
		TOD	TIME[8]	TOD
		DATE	DATE	TIME[9]
		TOD	TOD	TIME[9]
		DT	TIME	DT
		DT	DT	TIME[9]

1) Der Datentyp des Ergebnisses ist (soweit nicht explizit angegeben) der kleinste gemeinsame Datentyp, in den beide Operanden implizit konvertiert werden können.

2) Der 1. Operand muss größer Null sein.
Ausnahmen ab Version V4.1 des SIMOTION Kernels:
– Wenn der 2. Operand eine Ganzzahl ist, kann der 1. Operand auch kleiner Null sein.
– Wenn der 2. Operand positiv ist, kann der 1. Operand auch gleich Null sein.
Bis Version V4.0 des SIMOTION Kernels gilt: Wenn der 1. Operand gleich Null ist, kann eine eventuelle Fehlermeldung mit der ExecutionFaultTask abgefangen werden.

3) Datentyp des 1. Operanden.

4) Außer Datentyp BOOL. Berechnung erfolgt mit der vorzeichenlosen Ganzzahl gleicher Bitbreite.

5) Der 2. Operand muss ungleich Null sein.

6) Addition ggf. mit Überlauf.

7) Addition mit Datumskorrektur.

8) Einschränkung von TIME auf TOD vor Berechnung.

9) Diese Operationen arbeiten modulo zum Maximalwert des Datentyps TIME.

Hinweis

Wird bei Operationen mit Variablen vom allgemeinen Datentyp ANY_REAL die Grenzen des Wertebereichs überschritten, enthält das Ergebnis das entsprechende Bitmuster nach IEEE 754.

Um festzustellen, ob bei der Operation der Wertebereich überschritten wurde, können Sie mit der Funktion *_finite* (siehe Funktionshandbuch *SIMOTION Basisfunktionen*) das Ergebnis überprüfen.

4.6.4.1 Beispiele für arithmetische Ausdrücke

Beispiele für arithmetische Ausdrücke mit Zahlen

Unter der Annahme, dass *i* und *j* ganzzahlige Variablen (z. B. vom Datentyp INT) mit den Werten 11 bzw. -3 sind, sehen Sie nachfolgend als Beispiel einige ganzzahlige Ausdrücke und ihre entsprechenden Werte:

Ausdruck	Wert
`i + j`	`8`
`i - j`	`14`
`i * j`	`-33`
`i MOD j`	`-2`
`i / j`	`-3`

Beispiele für gültige arithmetische Ausdrücke mit Zeitangaben

Folgende Variablen seien gegeben:

Variablen	Inhalt	Datentyp
`t1`	`T#1D_1H_1M_1S_1MS`	`TIME`
`t2`	`T#2D_2H_2M_2S_2MS`	`TIME`
`d1`	`D#2004-01-11`	`DATE`
`d2`	`D#2004-02-12`	`DATE`
`tod1`	`TOD#11:11:11.11`	`TIME_OF_DAY`
`tod2`	`TOD#12:12:12.12`	`TIME_OF_DAY`
`dt1`	`DT#2004-01-11-11:11:11.11`	`DATE_AND_TIME`
`dt2`	`DT#2004-02-12-12:12:12.12`	`DATE_AND_TIME`

Einige Ausdrücke mit diesen Variablen und ihre entsprechenden Werte sehen Sie im Beispiel.

Ausdruck	Wert
`t1 + t2`	`T#3D_3H_3M_3S_3MS`
`dt1 + t1`	`DT#2004-01-12-12:12:12.111`
`t1 - t2`	`T#48D_16H_1M_46S_295MS`
`t1 * 2`	`T#2D_2H_2M_2S_2MS`

```
t1 / 2                                T#12H_30M_30S_500MS
DATE_AND_TIME_TO_TIME_OF_DAY(dt1)     TOD#11:11:11.110
DATE_AND_TIME_TO_DATE(dt1)            D#2004-01-11
```

4.6.5 Vergleichsausdrücke

Definition

Ein Vergleichsausdruck ist ein mit Vergleichsoperatoren (siehe Bild) gebildeter Ausdruck vom Datentyp BOOL.

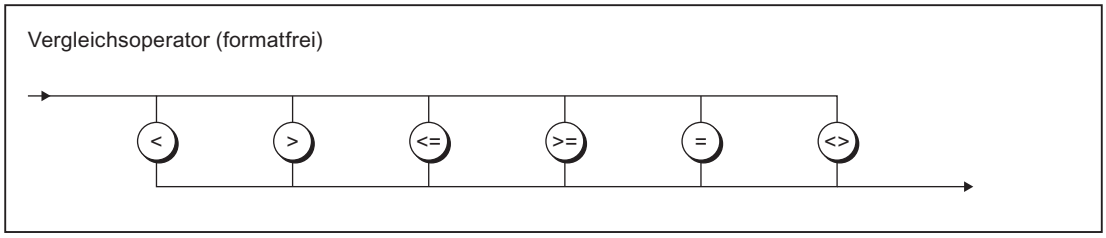


Bild 4-26 Syntax: Vergleichsoperatoren

Vergleichsoperatoren vergleichen 2 Operanden bezüglich ihres Wertes (siehe Tabelle) und liefern als Ergebnis einen Booleschen Wert.

1. Operand Operator 2. Operand -> Boolescher Wert

Tabelle 4-34 Bedeutung der Vergleichsoperatoren

Operator	Bedeutung
>	1. Operand ist **größer** als der 2. Operand
<	1. Operand ist **kleiner** als der 2. Operand
>=	1. Operand ist **größer oder gleich** dem 2. Operanden
<=	1. Operand ist **kleiner oder gleich** dem 2. Operanden
=	1. Operand ist **gleich** dem 2. Operanden
<>	1. Operand ist **ungleich** dem 2. Operanden

Das Ergebnis des Vergleichsausdrucks ist:

- 1 (TRUE), wenn der Vergleich erfüllt ist,
- 0 (FALSE), wenn der Vergleich nicht erfüllt ist.

Die nachstehende Tabelle zeigt die für beide Operanden zulässigen Kombinationen der Datentypen und Vergleichsoperatoren.

Tabelle 4-35 Vergleichsausdrücke: Zulässige Kombinationen der Datentypen und Vergleichsoperatoren

Datentyp		Zulässige Vergleichsoperatoren
1. Operand	2. Operand	
ANY_NUM	ANY_NUM[1)]	<, >, <=, >=, =, <>
ANY_BIT	ANY_BIT	<, >, <=, >=, =, <>
DATE	DATE	<, >, <=, >=, =, <>
TIME_OF_DAY (TOD)	TIME_OF_DAY (TOD)	<, >, <=, >=, =, <>
DATE_AND_TIME (DT)	DATE_AND_TIME (DT)	<, >, <=, >=, =, <>
TIME	TIME	<, >, <=, >=, =, <>
STRING	STRING[2)]	<, >, <=, >=, =, <>
Aufzählungsdatentyp	Aufzählungsdatentyp[3)]	=, <>
Feld (ARRAY)	Feld (ARRAY)[3)]	=, <>
Struktur (STRUCT)	Struktur (STRUCT)[3)]	=, <>

1) Der Vergleich erfolgt im kleinsten gemeinsamen Datentyp, in den beide Operanden implizit konvertiert werden können.

2) Variablen vom Datentyp STRING sind unabhängig von der deklarierten Länge des Strings vergleichbar.
Zum Vergleich zweier Variablen vom Datentyp STRING mit unterschiedlicher Länge wird die kürzere Zeichenkette durch rechtsseitiges Einfügen von $00 auf die Länge der längeren Zeichenkette expandiert. Der Vergleich beginnt von links nach rechts und erfolgt auf Basis des ASCII-Code der jeweiligen Zeichen. Beispiel: 'ABC' < 'AZ' < 'Z' < 'abc' < 'az' < 'z'.

3) Datentyp des 1. Operanden.

Vergleichsausdrücke sowie Variablen oder Konstanten vom Datentyp BOOL können mit logischen Operatoren zu logischen Ausdrücken verknüpft werden (siehe Logische Ausdrücke und Bitfolgeausdrücke (Seite 164)). Damit können Abfragen wie *Wenn a < b und b < c, dann …* realisiert werden.

Hinweis

In einem Ausdruck sind Vergleichsoperatoren höherrangig als logische Operatoren (siehe Rangfolge der Operatoren (Seite 166)). Deshalb müssen die Operanden eines Vergleichsausdrucks in Klammern stehen, wenn sie selbst logische Ausdrücke oder Bitfolgeausdrücke sind.

Beachten Sie eine mögliche Fehlerquelle beim Vergleich von REAL- oder LREAL-Größen (auch entsprechende Systemvariablen, z. B. Achsposition).

Tabelle 4-36 Beispiele für Vergleichsausdrücke

```
IF A = 2 THEN
    ; //...
END_IF;

var_1 := B < C;         // var_1 vom Datentyp BOOL

IF D < E OR var_2 THEN  // var_2 vom Datentyp BOOL
    ; // ...
END_IF;

var_3 := 0 < F < 10     // var_3 vom Datentyp BOOL, Wert TRUE
    // Die Berechnung erfolgt von links nach rechts
    // in folgender Form: (0 < F) < 10
    // Zuerst wird 0 < F berechnet.
    // Das Ergebnis ist vom Datentyp BOOL (FALSE oder TRUE)
    // und wird mit BYTE#10 verglichen.
    // Äquivalenter Ausdruck:
var_3 : = BOOL_TO_BYTE (0 < F) < BYTE#10
```

4.6.6 Logische Ausdrücke und Bitfolgeausdrücke

Definition

Mit den logischen Operatoren AND, &, XOR und OR können Sie Operanden und Ausdrücke vom allgemeinen Datentyp ANY_BIT (BOOL, BYTE, WORD oder DWORD) verknüpfen.

Mit dem logischen Operator NOT können Sie Operanden und Ausdrücke vom Datentyp ANY_BIT verneinen.

Die Tabelle gibt Auskunft über die verfügbaren Operatoren:

Tabelle 4-37 Logische Operatoren

Operation	Operator	1. Operand	2. Operand	Ergebnis[1]
Negation	NOT	ANY_BIT	-	ANY_BIT
Konjunktion	AND bzw. &	ANY_BIT	ANY_BIT	ANY_BIT
Exklusive Disjunktion	XOR	ANY_BIT	ANY_BIT	ANY_BIT
Disjunktion	OR	ANY_BIT	ANY_BIT	ANY_BIT
[1] Der Datentyp des Ergebnisses wird vom mächtigsten Datentyp der Operanden bestimmt.				

Der Ausdruck wird bezeichnet

- als **logischer Ausdruck**, wenn ausschließlich Operanden vom Datentyp BOOL verwendet werden.
  Die Operatoren wirken gemäß der nachfolgenden Wahrheitstabelle auf die Operanden.
  Das Ergebnis eines logischen Ausdrucks ist 1 (TRUE) oder 0 (FALSE).
- als **Bitfolgeausdruck**, wenn Operanden der Datentypen BYTE, WORD oder DWORD verwendet werden.
  Die Operatoren wirken gemäß der nachfolgenden Wahrheitstabelle auf die einzelnen Bits der Operanden.

Tabelle 4-38 Wahrheitstabelle der logischen Operatoren

Operanden (Datentyp BOOL)		Ergebnis (Datentyp BOOL)				
a	b	NOT a	NOT b	a AND b a & b	a XOR b	a OR b
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	1

Beispiele

Ausdruck (n sei 10)	Wert
`(n>0) AND (n<20)`	`TRUE`
`(n>0) AND (n<5)`	`FALSE`
`(n>0) OR (n<5)`	`TRUE`
`(n>0) XOR (n<20)`	`FALSE`
`NOT ((n>0) AND n<20))`	`FALSE`

Ausdruck	Wert
`2#01010101 AND 2#11110000`	`2#01010000`
`2#01010101 OR 2#11110000`	`2#11110101`
`2#01010101 XOR 2#11110000`	`2#10100101`
`NOT 2#01010101`	`2#10101010`

Ausdruck in Abfrage (value1 sei 2#01, value2 sei 2#11)

```
IF (value1 AND value2) = 2#01 THEN...
```

Bedingung liefert TRUE, da Bitfolgeausdruck 2#01 liefert.

4.6.7 Rangfolge der Operatoren

In Ausdrücke (Seite 155) wurden einige allgemeine Regeln zur Bildung von Ausdrücken beschrieben. In der Tabelle sehen Sie die Rangfolge der einzelnen Operatoren innerhalb eines Ausdrucks.

Operation	Symbol	Rangfolge
Klammerung	**(**Ausdruck**)**	Höchste
Funktions-Auswertung	**Bezeichner (**Argument-Liste**)** z. B. LN(a), EXPT (a,b) usw.	
Negation Komplement	– **NOT**	
Potenzierung	**	
Multiplikation Division Modulo	* / **MOD**	
Addition Subtraktion	**+** **–**	
Vergleich	**<, >, <=, >=**	
Gleichheit Ungleichheit	= <>	
Boolesches UND	**&, AND**	
Boolesches EXKLUSIV ODER	**XOR**	
Boolesches ODER	**OR**	Niederste

4.7 Kontrollanweisungen

Nur wenige Quelldatei-Abschnitte können Sie so programmieren, dass alle Anweisungen bis zum Abschnittsende in Folge hintereinander ablaufen. Im Regelfall werden, in Abhängigkeit von Bedingungen, nur bestimmte Anweisungen ausgeführt (Alternativen) oder auch mehrfach wiederholt (Schleifen). Die programmtechnischen Mittel hierzu sind die Kontrollanweisungen innerhalb eines Quelldatei-Abschnitts.

4.7.1 IF-Anweisung

Beschreibung

Die IF-Anweisung ist eine bedingte Anweisung. Sie bietet eine oder mehrere Optionen und wählt eine (gegebenenfalls auch keine) ihrer Anweisungsteile zur Ausführung an.

Die Ausführung der bedingten Anweisung bewirkt die Auswertung der angegebenen logischen Ausdrücke. Ist der Wert eines Ausdrucks TRUE, so gilt die Bedingung als erfüllt, bei FALSE als nicht erfüllt.

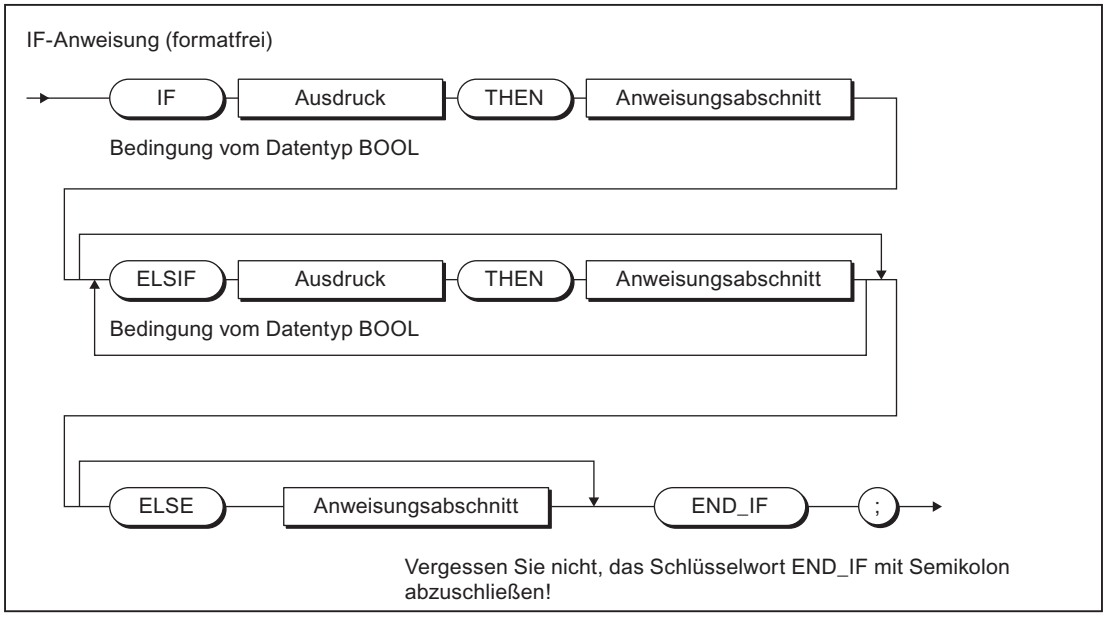


Bild 4-27 Syntax: IF-Anweisung

Bearbeitungsreihenfolge

Die IF-Anweisung wird nach den folgenden Regeln bearbeitet:

1. Ist der Wert des ersten Ausdrucks TRUE, so wird der nach THEN folgende Anweisungsteil ausgeführt.
   Danach wird die Programmabarbeitung nach END_IF fortgesetzt.
2. Ist der Wert des ersten Ausdrucks FALSE, so werden die Ausdrücke in den ELSIF-Zweigen ausgewertet. Falls in den ELSIF-Zweigen ein boolescher Ausdruck TRUE ist, wird der nach THEN folgende Anweisungsteil ausgeführt.
   Danach wird die Programmabarbeitung nach END_IF fortgesetzt.
3. Falls in den ELSIF-Zweigen kein boolescher Ausdruck TRUE ist, wird die Anweisungsfolge bei ELSE ausgeführt (oder keine Anweisungsfolge, falls der ELSE-Zweig nicht vorhanden ist).
   Danach wird die Programmabarbeitung nach END_IF fortgesetzt.

Es dürfen beliebig viele ELSIF-Anweisungen vorhanden sein.

Beachten Sie, dass die ELSIF-Zweige und/oder der ELSE-Zweig fehlen können. Diese Fälle werden behandelt, als wären diese Zweige mit leeren Anweisungen vorhanden.

Hinweis

Die Verwendung eines oder mehrerer ELSIF-Zweige bietet gegenüber einer Sequenz von IF-Anweisungen den Vorteil, dass die einem gültigen Ausdruck folgenden logischen Ausdrücke nicht mehr ausgewertet werden. Die Laufzeit eines Programms lässt sich so verkürzen bzw. die Ausführung von nicht zulässigen Programmteilen verhindern.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der IF-Anweisung:

Beispiel für die IF-Anweisung

```
IF A = B THEN
    n:= 0;
END_IF;

IF temperature < 5.0 THEN
    %Q0.0 := TRUE;
ELSIF temperature > 10.0 THEN
    %Q0.2 := TRUE;
ELSE
    %Q0.1 := TRUE;
END_IF;
```

4.7.2 CASE-Anweisung

Beschreibung

Die CASE-Anweisung dient der 1 aus n Auswahl eines Programmteils.

Die Auswahl bestimmt über einen Auswahlausdruck (Selektor),

- Ausdruck vom allgemeinen Datentyp ANY_INT
- Variable eines Aufzählungsdatentyps (Enumerator)

Ausgewählt wird aus einer Liste von Werten (Wertliste), wobei jedem Wert oder jeder Gruppe von Werten ein Programmabschnitt zugeordnet ist.

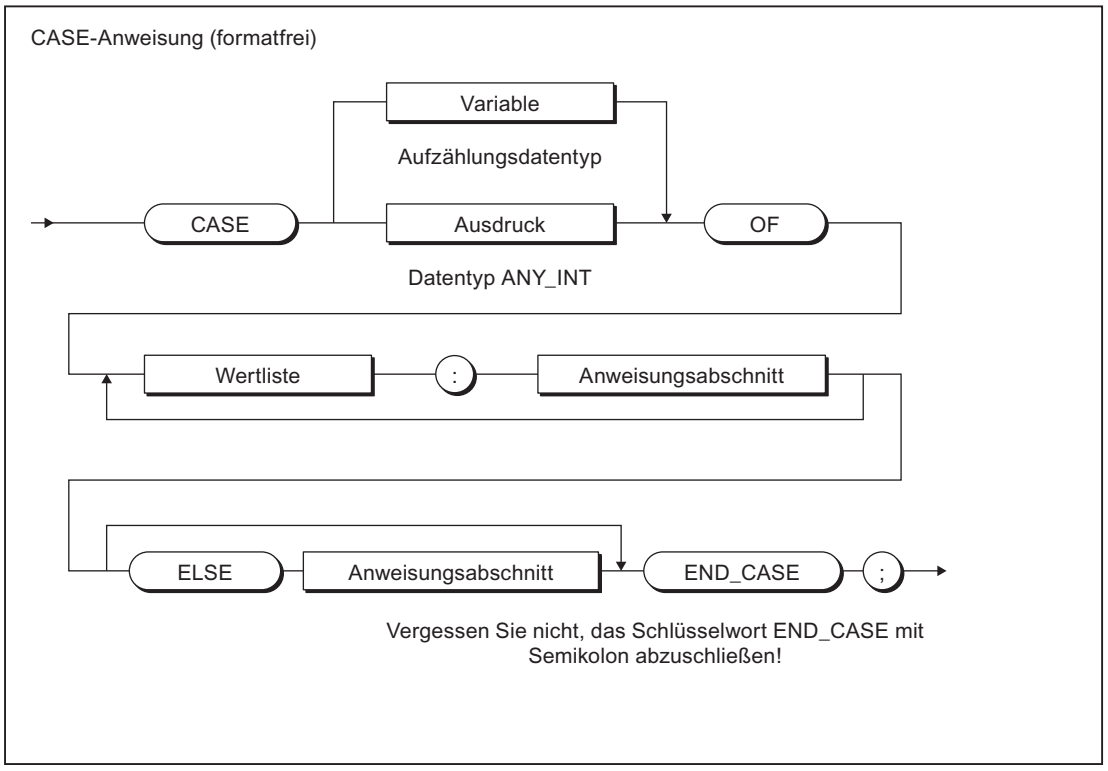


Bild 4-28 Syntax: CASE-Anweisung

Bearbeitungsreihenfolge

Die CASE-Anweisung wird nach folgenden Regeln bearbeitet:

1. Der Auswahlausdruck (Selektor) wird berechnet. Er muss einen Wert vom allgemeinen Datentyp ANY_INT (Ganzzahl) oder von einem Aufzählungsdatentyp liefern.
2. Anschließend wird überprüft, ob der Wert des Selektors in der Wertliste enthalten ist. Jeder Wert in dieser Liste stellt einen der erlaubten Werte für den Auswahlausdruck dar.
3. Bei Übereinstimmung wird der, der Liste zugeordnete Programmteil, ausgeführt.
4. Der ELSE-Zweig ist optional. Er wird ausgeführt, wenn der Vergleichsvorgang keine Übereinstimmung ergibt.
5. Wenn der ELSE-Zweig fehlt und der Vergleichsvorgang keine Übereinstimmung ergibt, wird mit der Programmausführung nach END_CASE fortgefahren.

Wertliste

Die Wertliste enthält die erlaubten Werte für den Auswahlausdruck.

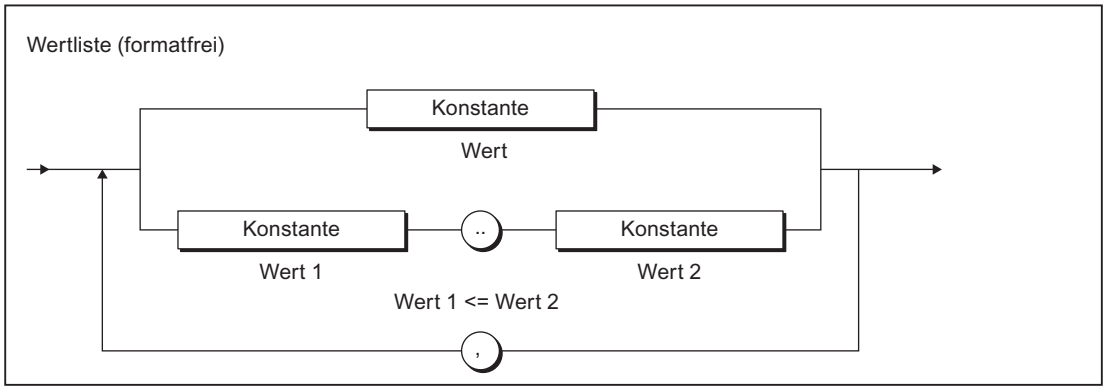


Bild 4-29 Syntax: Wertliste

Beachten Sie bei der Formulierung der Wertliste:

- Jede Wertliste kann mit einer Konstanten (*Wert*), einer Konstantenliste (*Wert1, Wert2, Wert3, …*) oder einem Konstantenbereich (*Wert1 .. Wert2*) beginnen.
- Die Werte innerhalb der Werteliste ganzzahlige Konstanten bzw. Elemente des Aufzählungsdatentyps des Selektors sein.

Hinweis

Ein Wert soll in den Wertlisten einer CASE-Anweisung nur einmal auftreten.

Bei mehrfachem Auftreten eines Wertes gibt der Compiler eine Warnmeldung aus; es wird der Anweisungsteil zu der Wertliste ausgeführt, in welcher der Wert erstmals auftritt.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der CASE-Anweisung:

Beispiel für die CASE-Anweisung

```
CASE intVar OF
    1    : a := 1;
    2,3  : b := 1;
    4..9 : c := 1; d := 2;
ELSE
    e := 5;
END_CASE;
```

4.7.3 FOR-Anweisung

Beschreibung

Eine FOR-Anweisung oder auch Wiederholungsanweisung führt eine Anweisungsfolge in einer Schleife aus, wobei einer Variablen (der Laufvariablen) fortlaufend Werte zugewiesen werden. Die Laufvariable muss der Bezeichner einer lokalen Variablen vom Typ SINT, INT oder DINT sein.

Die Definition einer Schleife mit FOR schließt die Festlegung eines Start- und Endwertes mit ein. Beide Werte müssen typgleich mit der Laufvariablen sein.

Hinweis

Verwenden Sie die FOR-Anweisung, wenn die Anzahl der Durchläufe bei der Programmierung bekannt ist.

Ist die Anzahl der Durchläufe nicht bekannt, ist die WHILE- oder REPEAT-Anweisung geeigneter, siehe WHILE-Anweisung (Seite 173) und REPEAT-Anweisung (Seite 174).

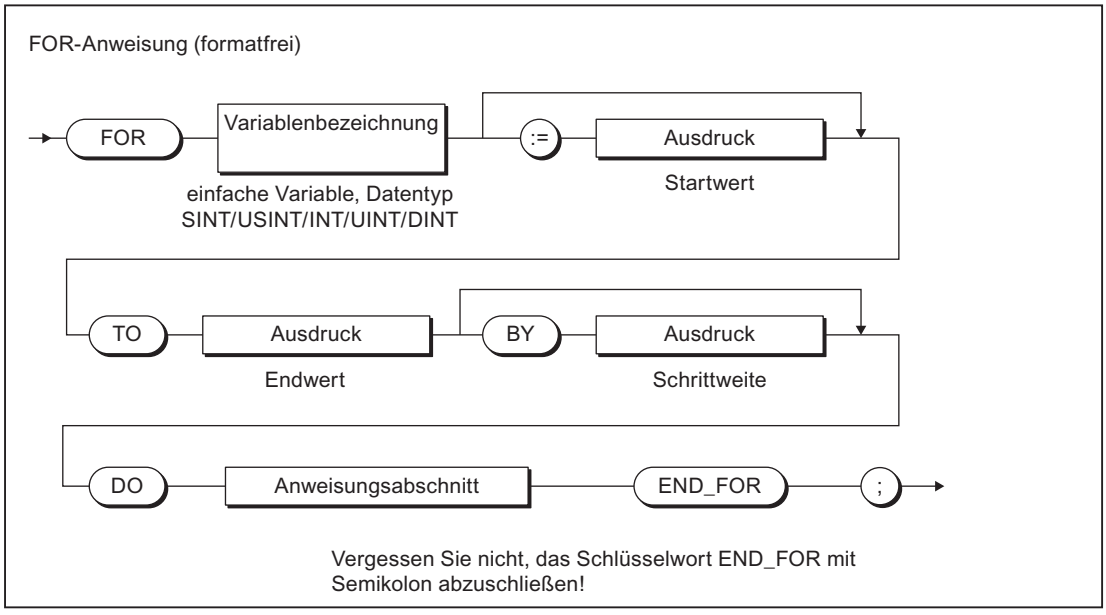


Bild 4-30 Syntax: FOR-Anweisung

Bearbeitungsreihenfolge

Die FOR-Anweisung wird nach folgenden Regeln bearbeitet:

1. Beim Start der Schleife wird die Laufvariable auf den **Startwert** gesetzt und nach jedem Schleifendurchlauf um die angegebene Schrittweite erhöht (positive Schrittweite) oder erniedrigt (negative Schrittweite), solange bis der **Endwert** erreicht ist. Nach dem ersten Schleifendurchlauf wird der Inhalt der Laufvariable als **Aktualwert** bezeichnet.
2. Bei jedem Durchlauf wird überprüft, ob folgende Bedingungen erfüllt sind:
   - **Startwert bzw. Aktualwert <= Endwert (**bei **positiver Schrittweite)** oder
   - **Startwert bzw. Aktualwert >= Endwert (**bei **negativer Schrittweite)**

   Ist die Bedingung erfüllt, wird die Anweisungsfolge ausgeführt.
   Ist die Bedingung nicht erfüllt, wird die Schleife und damit die Anweisungsfolge übersprungen und die Programmbearbeitung nach END_FOR fortgesetzt.
3. Wird die FOR-Schleife wegen Schritt 2 nicht ausgeführt, behält die Laufvariable den Aktualwert.

Regeln

Für die FOR-Anweisung gelten folgende Regeln:

- Die Angabe von *BY [Schrittweite]* kann entfallen. Ist keine Schrittweite spezifiziert, dann beträgt sie +1.
- Startwert, Endwert und Schrittweite sind Ausdrücke, siehe Ausdrücke (Seite 155). Die Auswertung erfolgt einmalig am Beginn der Ausführung der FOR-Anweisung.
- Wenn Startwert und Endwert vom Datentyp DINT sind, muss der Betrag von (Endwert - Startwert) kleiner DINT#MAX (2**31 - 1) sein, siehe auch Wertebereichsgrenzen elementarer Datentypen (Seite 118).
- Bei Abbruch der Schleife enthält die Laufvariable den Wert, der zum Abbruch führt, d. h. er wird vor Abbruch der Schleife entsprechend der Schrittweite aktualisiert.
- Während der Ausführung der Schleife dürfen die Laufvariable (Aktualwert) sowie der Startwert, der Endwert und die Schrittweite nicht geändert werden.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der FOR-Anweisung:

Beispiel für die FOR-Anweisung

```
FOR k := 1 TO 10 BY 2 DO
    l := l + 1;
    // ...
END_FOR;
```

4.7.4 WHILE-Anweisung

Beschreibung

Die WHILE-Anweisung erlaubt die wiederholte Ausführung einer Anweisungsfolge unter der Kontrolle einer Durchführungsbedingung. Die Durchführungsbedingung wird nach den Regeln eines logischen Ausdrucks gebildet.

Hinweis

Verwenden Sie die WHILE-Anweisung, wenn die Anzahl der Durchläufe bei der Programmierung nicht bekannt ist.

Wenn die Anzahl der Durchläufe bekannt ist, ist die FOR-Anweisung (Seite 171) geeigneter.

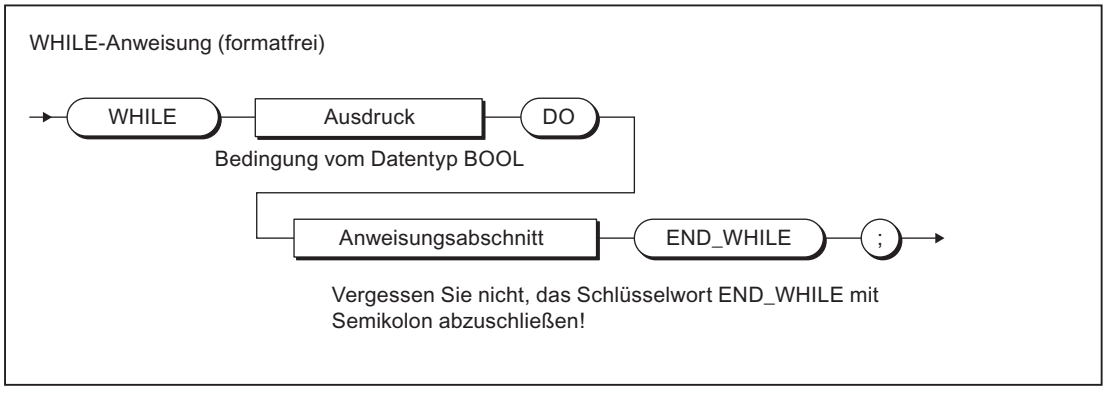


Bild 4-31 Syntax: WHILE-Anweisung

Der auf DO folgende Anweisungsteil wird solange wiederholt, wie die Durchführungsbedingung den Wert TRUE besitzt.

Bearbeitungsreihenfolge

Die WHILE-Anweisung wird nach folgenden Regeln bearbeitet:

1. **Vor** jeder Ausführung des Anweisungsteils wird die Durchführungsbedingung ausgewertet.
2. Tritt der Wert TRUE auf, wird der Anweisungsteil ausgeführt.
3. Tritt der Wert FALSE auf, ist die Ausführung der WHILE-Anweisung beendet (dies kann schon bei der ersten Auswertung der Fall sein), die Programmabarbeitung wird nach END_WHILE fortgesetzt.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der WHILE-Anweisung:

Beispiel für die WHILE-Anweisung

```
WHILE Index <= 50 DO
      Index:= Index + 2;
END_WHILE;
```

4.7.5 REPEAT-Anweisung

Beschreibung

Eine REPEAT-Anweisung bewirkt die wiederholte Ausführung einer zwischen REPEAT und UNTIL stehenden Anweisungsfolge bis zum Eintreten einer Abbruchbedingung. Die Abbruchbedingung wird nach den Regeln eines logischen Ausdrucks gebildet.

Hinweis

Verwenden Sie die REPEAT-Anweisung, wenn die Anzahl der Durchläufe bei der Programmierung nicht bekannt ist.

Wenn die Anzahl der Durchläufe bekannt ist, ist die FOR-Anweisung (Seite 171) geeigneter.

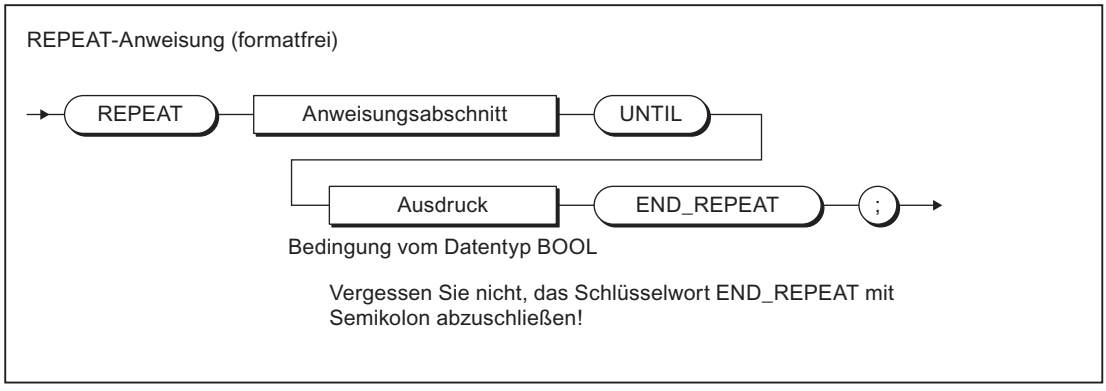


Bild 4-32 Syntax: REPEAT-Anweisung

Die Bedingung wird jeweils **nach** der Ausführung der Anweisungsteils überprüft. Dies bedeutet, dass der Anweisungsteil mindestens einmal ausgeführt wird, auch wenn die Abbruchbedingung von Anfang an erfüllt ist.

Bearbeitungsreihenfolge

Die REPEAT-Anweisung wird nach folgenden Regeln bearbeitet:

1. Nach jeder Ausführung des Anweisungsteils wird die Durchführungsbedingung ausgewertet.
2. Tritt der Wert FALSE auf, wird der Anweisungsteil erneut ausgeführt.
3. Tritt der Wert TRUE auf, wird die Ausführung der REPEAT-Anweisung beendet; die Programmabarbeitung wird nach END_REPEAT fortgesetzt.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der REPEAT-Anweisung:

Beispiel für die REPEAT-Anweisung

```
Index := 1;
REPEAT
    Index := Index + 2;
UNTIL Index > 50
END_REPEAT;
```

4.7.6 EXIT-Anweisung

Beschreibung

Eine EXIT-Anweisung dient zum Verlassen einer Schleife (FOR, WHILE oder REPEAT-Schleife) an beliebiger Stelle und unabhängig vom Erfülltsein der Abbruchbedingung.

Diese Anweisung bewirkt das sofortige Verlassen derjenigen Wiederholungsanweisung, welche die EXIT-Anweisung unmittelbar umgibt.

Die Ausführung des Programms wird nach dem Ende der Wiederholungsschleife (z. B. nach END_FOR) fortgesetzt.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der EXIT-Anweisung:

Beispiel für die EXIT-Anweisung

```
Index := 1;
FOR Index := 1 to 51 BY 2 DO
    IF %I0.0 THEN
        EXIT;
    END_IF;
END_FOR;
(*
Die folgende Wertzuweisung kommt nach der Ausführung von EXIT oder nach dem
regulären Ende der FOR-Schleife zur Ausführung:
*)
Index_find := Index_2;
```

4.7.7 RETURN-Anweisung

Beschreibung

Eine RETURN-Anweisung bewirkt das Beenden der aktuell bearbeiteten POE (Programm, Funktion, Funktionsbaustein).

Nach Beenden einer Funktion oder eines Funktionsbausteins wird die Programmbearbeitung in der übergeordneten POE nach der Stelle fortgesetzt, wo die Funktion bzw. der Funktionsbaustein aufgerufen wurde.

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der RETURN-Anweisung:

Beispiel für die RETURN-Anweisung

```
Index := 1;
FOR Index := 1 to 51 BY 2 DO
    IF %I0.0 THEN
        RETURN;
    END_IF;
END_FOR;
(*
Die folgende Wertzuweisung kommt nach dem regulären Ende der FOR-Schleife
zur Ausführung, nicht jedoch nach Ausführung von RETURN.
*)
Index_find:= Index_2;
```

4.7.8 WAITFORCONDITION-Anweisung

Beschreibung

Mit der WAITFORCONDITION-Anweisung können Sie in einer MotionTask auf ein programmierbares Ereignis bzw. eine Bedingung warten. Der Befehl setzt die Ausführung der MotionTask, aus der heraus er aufgerufen wird, so lange aus, bis die Bedingung wahr wird. Diese Bedingung programmieren Sie in einer Expression (Seite 208).

Mehr zu WAITFORCONDITION und Expressions im Zusammenhang erfahren Sie im Funktionshandbuch *SIMOTION Basisfunktionen*.

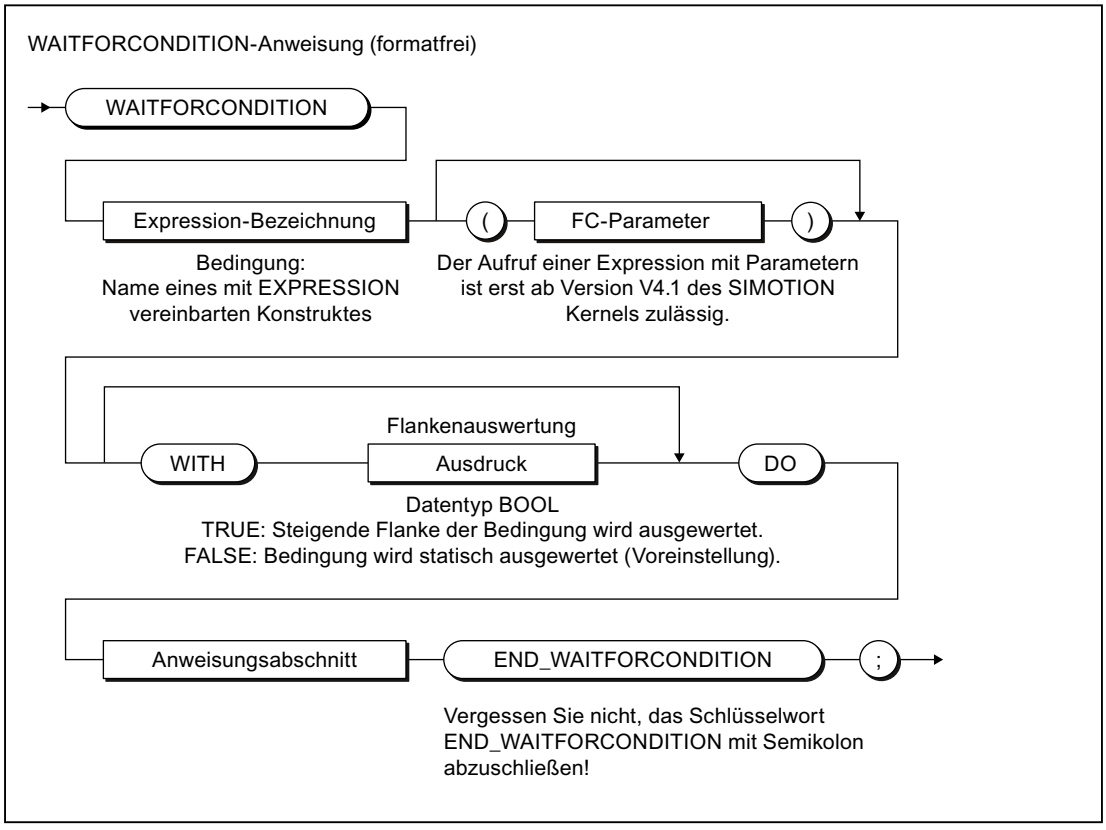


Bild 4-33 Syntax: WAITFORCONDITION-Anweisung

Expression-Bezeichnung ist ein mit EXPRESSION vereinbartes Konstrukt; ihr Wert bestimmt (ggf. zusammen mit *WITH Flankenauswertung*), ob die Bedingung als erfüllt gilt.

Die Folge *WITH Flankenauswertung* ist optional. Dabei ist *Flankenauswertung* ein Ausdruck vom Datentyp BOOL; er bestimmt, wie der Wert von *Expression-Bezeichnung* ausgewertet wird:

- *Flankenauswertung* = TRUE:
  Die steigende Flanke von *Expression-Bezeichnung* wird ausgewertet; d. h. die Bedingung ist erfüllt, wenn der Wert von *Expression-Bezeichnung* von FALSE nach TRUE **wechselt**.
- *Flankenauswertung* = FALSE:
  Der statische Wert von *Expression-Bezeichnung* wird ausgewertet; d. h. die Bedingung ist erfüllt, wenn der Wert von *Expression-Bezeichnung* TRUE **ist**.

Bei Nichtangabe von *WITH Flankenauswertung* ist die Voreinstellung FALSE, d. h. der statische Wert von *Expression-Bezeichnung* wird ausgewertet.

Der Anweisungsabschnitt muss mindestens eine Anweisung enthalten (auch leere Anweisung möglich).

Beispiel

Das folgende Beispiel veranschaulicht den Gebrauch der WAITFORCONDITION-Anweisung:

Beispiel für die WAITFORCONDITION-Anweisung

```
// ...
// Aufruf des Befehls mit Namen der Expression
WAITFORCONDITION myExpression WITH TRUE DO
// hier mind. eine Anweisung, wird hochprior ausgeführt, z. B.
    %Q0.0 := TRUE;
END_WAITFORCONDITION;
// ...
```

Vollständiges Beispiel siehe Beschreibung zur Expression (Seite 208).

4.7.9 GOTO-Anweisung

Die GOTO-Anweisung bewirkt einen Sprung zu der im Befehl angegebenen Sprungmarke (siehe Sprunganweisung und -markierung (Seite 322)).

Sprunganweisungen programmieren Sie mit dem Befehl GOTO unter Angabe der Sprungmarke, zu der Sie springen wollen. Sprünge sind nur innerhalb einer POE zulässig.

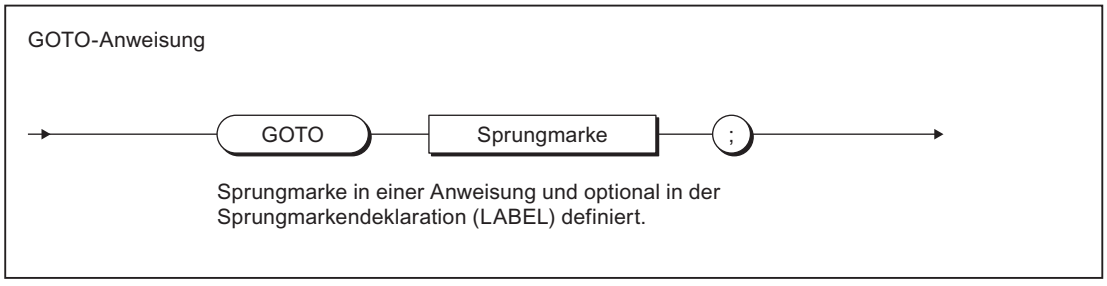


Bild 4-34 Syntax: GOTO-Anweisung

Hinweis

Die GOTO-Anweisung sollte nur in Sonderfällen (z. B. Fehlerbearbeitung) verwendet werden. Nach den Regeln der strukturierten Programmierung sollte sie nicht verwendet werden.

Sprünge sind nur innerhalb einer POE zulässig.

Folgende Sprünge sind nicht zulässig:

- Sprünge in untergeordnete Kontrollstrukturen (WHILE, FOR usw.)
- Sprünge aus einer WAITFORCONDITION-Struktur
- Sprünge innerhalb von CASE-Anweisungen

Sprungmarken können nur in der POE deklariert werden, in der sie verwendet werden. Wenn Sprungmarken deklariert werden, dürfen nur diese deklarierten Sprungmarken verwendet werden.

4.8 Datentyp-Konvertierungen

In diesem Kapitel erfahren Sie, wie Sie implizit und explizit zwischen elementaren Datentypen konvertieren können. Außerdem ist ein Überblick über ergänzende Konvertierungsmöglichkeiten enthalten.

4.8.1 Konvertierung elementarer Datentypen

Die Tabelle gibt einen Überblick über die Konvertierungsmöglichkeiten zwischen numerischen Datentypen und Bit-Datentypen. Dabei wird unterschieden zwischen:

- Implizite Konvertierung (Seite 180): Die Konvertierung wird bei Verwendung unterschiedlicher Datentypen in einem Ausdruck oder einer Wertzuweisung vom Compiler automatisch durchgeführt.
- Explizite Konvertierung (Seite 182): Die Konvertierung wird vom Anwender durch Aufruf einer Konvertierungsfunktion (siehe Funktionshandbuch *SIMOTION Basisfunktionen*) durchgeführt.

Tabelle 4-39 Typkonvertierung numerischer Datentypen und Bitdatentypen

Quelldatentyp	Zieldatentyp												
	BOOL	BYTE	WORD	DWORD	USINT	UINT	UDINT	SINT	INT	DINT	REAL	LREAL	STRING
BOOL	–	Im/Ex	Im/Ex	Im/Ex	Val	Val	Val	Val	Val	Val	Val	Val	–
BYTE	Ex	–	Im/Ex	Im/Ex	Ex	Ex	Ex	Ex	Ex	Ex	Val	Val	Elem
WORD	Ex	Ex	–	Im/Ex	Ex	Ex	Ex	Ex	Ex	Ex	Val	Val	–
DWORD	Ex	Ex	Ex	–	Ex	Ex	Ex	Ex	Ex	Ex	Ex/Val	Val	–
USINT	Val	Ex	Ex	Ex	–	Im/Ex	Im/Ex	Ex	Im/Ex	Im/Ex	Im/Ex	Im/Ex	–
UINT	Val	Ex	Ex	Ex	Ex	–	Im/Ex	Ex	Ex	Im/Ex	Im/Ex	Im/Ex	–
UDINT	Val	Ex	Ex	Ex	Ex	Ex	–	Ex	Ex	Ex	Ex	Ex	Ex
SINT	Val	Ex	Ex	Ex	Ex	Ex	Ex	–	Im/Ex	Im/Ex	Im/Ex	Im/Ex	–
INT	Val	Ex	Ex	Ex	Ex	Ex	Ex	Ex	–	Im/Ex	Im/Ex	Im/Ex	–
DINT	Val	Ex	Ex	Ex	Ex	Ex	Ex	Ex	Ex	–	Ex	Im/Ex	Ex
REAL	Val	Val	Val	Ex/Val	Ex	Ex	Ex	Ex	Ex	Ex	–	Im/Ex	Ex
LREAL	Val	Val	Val	Val	Ex	Ex	Ex	Ex	Ex	Ex	Ex	–	Ex
STRING	–	Elem	–	–	–	–	Ex	–	–	Ex	Ex	Ex	–

Im: Implizite Datentypkonvertierung möglich

Ex: Explizite Datentypkonvertierung mittels Typumwandlungsfunktion *Quelldatentyp*_TO_*Zieldatentyp* möglich

Val: Explizite Datentypkonvertierung mittels Typumwandlungsfunktion *Quelldatentyp*_VALUE_TO_*Zieldatentyp* möglich

Elem: Implizite Datentypkonvertierung mit Element des Datentyps STRING

Zu den Konvertierungsfunktionen für Datentypen des Datums und der Zeit: siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

4.8.1.1 Implizite Datentyp-Konvertierungen

Eine implizite Konvertierung ist immer dann möglich, wenn durch Vergrößerung des Wertebereichs kein Wertverlust gegeben ist, z. B. von REAL nach LREAL oder von INT nach REAL. Das Ergebnis ist immer definiert.

Das folgende Bild zeigt alle impliziten Typkonvertierungsketten grafisch. Dabei stellt eine Stufe in einer Typkonvertierungskette – von links nach rechts oder von oben nach unten gelesen – immer eine Vergrößerung des Wertebereichs dar.

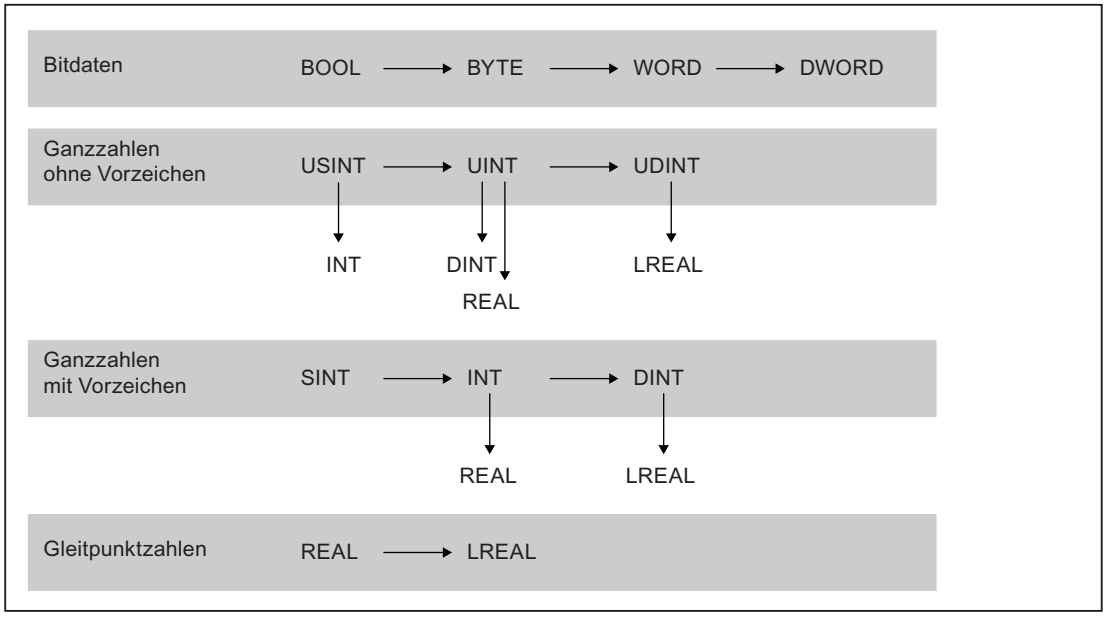


Bild 4-35 Implizite Typkonvertierungsketten (von links nach rechts eine oder mehrere Stufen oder von oben nach unten eine Stufe)

Folgende impliziten Typkonvertierungen sind möglich:

1. horizontal (von links nach rechts) über eine oder mehrere Stufen (z. B. USINT nach UDINT)
2. vertikal (von oben nach unten) über eine Stufe (z. B. UINT nach REAL)

Die impliziten Typkonvertierungen können in der angegebenen Reihenfolge kombiniert werden. (z. B. INT nach LREAL).

Alle anderen Typkonvertierungen sind nicht implizit durchführbar (z. B. UDINT nach REAL), d. h. Sie müssen eine explizite Funktion (Seite 182) benutzen, siehe Funktionshandbuch SIMOTION Basisfunktionen.

Hinweis

In Ausdrücken wird das Ergebnis immer im größten Zahlenformat des Ausdrucks berechnet.

Eine Wertzuweisung des Ausdrucks an eine Variable ist nur möglich:

- wenn der berechnete Ausdruck und die zuzuweisende Variable den gleichen Datentyp haben
- wenn der Datentyp des berechneten Ausdrucks implizit in den Datentyp der zuzuweisenden Variablen konvertiert werden kann.

Zu dieser Fehlerquelle und deren Abhilfe: siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

Tabelle 4-40 Beispiel für Datentypen in Ausdrücken und Wertzuweisungen

```
VAR
    usint_var  : USINT;
    real_var   : REAL;
    byte_var   : BYTE;
    string_var : STRING[80] := 'example for string';
END_VAR

usint_var := 234 / 10;       // Datentyp des Ausdrucks: USINT
                             // Ergebnis = 23

real_var  := 234 / 10;       // Datentyp des Ausdrucks: USINT
                             // Implizite Konvertierung möglich
                             // Ergebnis = 23.0

usint_var := 234 / SINT#10;  // Datentyp des Ausdrucks: INT
                             // Implizite Konvertierung und
                             // Wertzuweisung nicht möglich

real_var  := 234 / 10.0;     // Datentyp des Ausdrucks: REAL
                             // Ergebnis = 23.4

usint_var := 234 / 10.0;     // Datentyp des Ausdrucks: REAL
                             // Implizite Konvertierung und
                             // Wertzuweisung nicht möglich

byte_var  := string_var[5];  // Implizite Konvertierung möglich
                             // Ergebnis = 16#70 ('p')

string_var[10] := byte_var;  // Implizite Konvertierung möglich
                             // Ergebnis = 'example fpr string'
```

Hinweis

Geben Sie ggf. bei Zahlen den Datentyp explizit an (z. B. UINT#127, wenn die Zahl 127 vom Datentyp UINT statt USINT sein soll).

4.8.1.2 Explizite Datentyp-Konvertierungen

Eine explizite Konvertierung ist immer dann notwendig, wenn Informationsverlust möglich ist, z. B. durch Verkleinerung des Wertebereichs oder durch Verringerung der Genauigkeit, wie bei der Konvertierung von LREAL nach REAL.

Die Konvertierungsfunktionen für numerische Datentypen und Bit-Datentypen sind im Funktionshandbuch *SIMOTION Basisfunktionen* aufgelistet.

Der Compiler gibt bei verlustbehafteten Konvertierungen Warnungen aus.

Hinweis

Das Ergebnis der Typkonvertierung kann bei Laufzeit des Programms zu Fehlern führen, es wird dann die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst (siehe "Verabeitungsfehler in Programmen" im Funktionshandbuch *SIMOTION Basisfunktionen*).

Besondere Vorsicht ist bei der Konvertierung von DWORD zu REAL geboten. Der Bitstring aus DWORD wird ungeprüft als REAL-Wert übernommen. Achten Sie selbst darauf, dass der Bitstring in DWORD dem Bitmuster einer normalisierten Gleitpunktzahl nach IEEE entspricht. Sie können hierzu die Funktionen _finite und _isNaN verwenden.

Andernfalls kann ein Fehler (FPU-Exception) ausgelöst werden, sobald der REAL-Wert erstmals bei einer Rechenoperation verwendet wird (z. B. im Programm oder beim Beobachten im Symbol-Browser).

Hinweis

Werden bei der Konvertierung von LREAL nach REAL die Grenzen des Wertebereichs überschritten, gilt:

- Underflow (Absolutwert der LREAL-Zahl ist kleiner als die kleinste positive REAL-Zahl): Ergebnis ist 0.0.
- Overflow (Absolutwert der LREAL-Zahl ist größer als die größte positive REAL-Zahl): Es wird die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst.

4.8.2 Ergänzende Konvertierungen

Über ST-Systemfunktionen und ST-Systemfunktionsbausteine sind außerdem folgende Konvertierungen möglich:

- **Zusammenfassen von Bitstring-Datentypen**
  Diese Funktionen fassen mehrere Variablen eines Bitstring-Datentyps zu einer Variablen eines übergeordneten Datentyps zusammen.
- **Zerlegen von Bitstring-Datentypen**
  Diese Funktionsbausteine zerlegen eine Variablen eines Bitstring-Datentyps in mehrere Variablen eines untergeordneten Datentyps.
- **Konvertierung zwischen beliebigen Datentypen und Byte-Feldern**
  Sie werden häufig verwendet, um definierte Übertragungsformate für den Datenaustausch zwischen verschiedenen Geräten zu schaffen.
  Weitere Informationen (z. B. zur Anordnung der Bytefelder, Anwendungsbeispiel): siehe Funktionshandbuch *SIMOTION Basisfunktionen*.
- **Wandlung von Datentypen technologischer Objekte**
  Sie wandelt Variablen eines hierarchischen TO-Datentyps (driveAxis, posAxis, followingAxis) oder des allgemeinen Typs ANYOBJECT in einen kompatiblen TO-Datentyp.

Anwendungsbeispiele und weitere Informationen: siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

Funktionen, Funktionsbausteine, Programme 5

In diesem Kapitel erfahren Sie, wie Sie selbstdefinierte Funktionen und Funktionsbausteine erstellen und aufrufen. Standardfunktionen sind bereits im System vorhanden und stellen Ihnen Lösungen im Bereich Typumwandlung, Trigonometrie und Bitstrings zur Verfügung. Wie Sie Systemfunktionen und Funktionen der Technologieobjekte (TO-Funktionen) einsetzen, erfahren Sie im Funktionshandbuch *SIMOTION Basisfunktionen*.

Eine **Funktion** (FC) ist ein Codebaustein ohne statische Daten. Alle lokalen Variablen verlieren nach Verlassen der Funktion ihren Wert; beim nächsten Aufruf werden sie neu initialisiert.

Ein **Funktionsbaustein** (FB) ist ein Codebaustein mit statischen Daten. Da ein FB über ein Gedächtnis verfügt, kann auf seine Ausgangsparameter zu jeder Zeit an jeder beliebigen Stelle im Anwenderprogramm zugegriffen werden. Lokale Variablen behalten ihren Wert zwischen mehreren Aufrufen.

Programme sind den FB ähnlich, nur besitzen sie keine Parametrierbarkeit. Sie können jedoch Ablaufebenen bzw. Tasks zugeordnet werden (siehe Funktionshandbuch *SIMOTION Basisfunktionen).*

FC und FB bieten den Vorteil der Wiederverwendbarkeit, da sie gekapselte und parametrierbare Quelldatei-Abschnitte sind.

Funktionen, Funktionsbausteine und Programme gehören zu den Programmorganisationseinheiten (POE), d. h. zu den ausführbaren Quelldatei-Abschnitten. Einen Überblick zu allen Quelldatei-Abschnitten erhalten Sie in Verwendung der Quelldatei-Abschnitte (Seite 212).

5.1 Erstellung und Aufruf von Funktionen und Funktionsbausteinen

Nachfolgend erfahren Sie Grundsätzliches zur Erstellung und Aufruf von Funktionen (FC) und Funktionsbausteinen (FB). Ein vollständiges Beispiel, das auch die Unterschiede zwischen FC und FB aufzeigt, siehe Funktionen und Funktionsbausteine im Vergleich (Seite 202).

Die Reihenfolge, die Sie bei Definition und Aufruf der genannten Quelldatei-Abschnitte einhalten müssen, siehe Verwendung der Quelldatei-Abschnitte (Seite 212).

Wie Sie FC und FB exportieren und importieren, siehe Import und Export zwischen ST-Quellen (Seite 225).

5.1.1 Funktionen definieren

Eine Funktion definieren Sie im Vereinbarungsteil des Implementationsabschnitts vor dem Abschnitt der Quelldatei (Programm, FB oder FC), in dem sie aufgerufen wird.

Dabei verwenden Sie folgende Syntax:

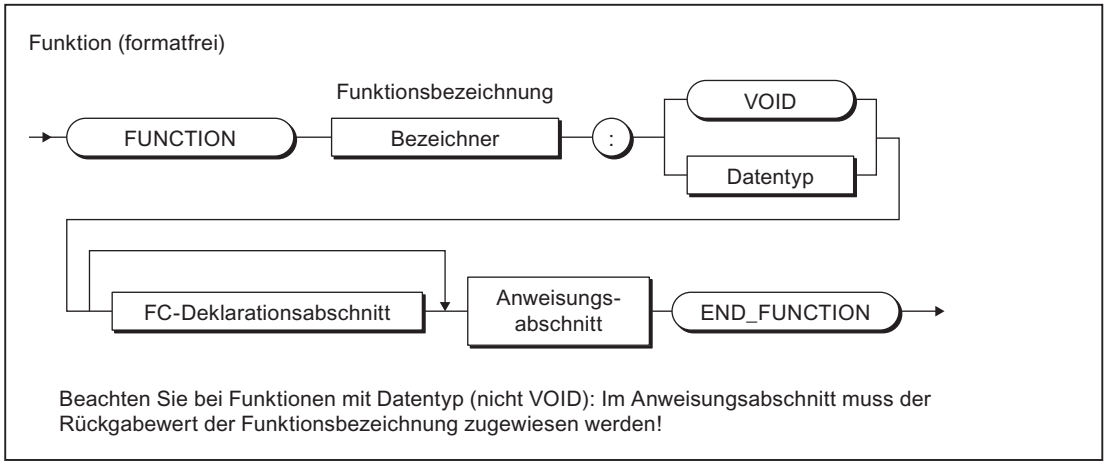


Bild 5-1 Syntax: Funktion (FC)

Geben Sie nach dem Schlüsselwort FUNCTION einen Bezeichner als FC-Namen und den Datentyp des Rückgabewerts an. Tragen Sie als Datentyp VOID ein, wenn die FC keinen Rückgabewert hat.

Danach folgen (siehe Beispiel in Quelldatei mit Kommentaren (Seite 203)):

- der optionale Deklarationsabschnitt
- der Anweisungsabschnitt
- das Schlüsselwort END_FUNCTION

5.1.2 Funktionsbausteine definieren

Einen Funktionsbaustein definieren Sie im Vereinbarungsteil des Implementationsabschnitts vor dem Abschnitt der Quelldatei (Programm, FB oder FC), in dem er aufgerufen wird.

Dabei verwenden Sie folgende Syntax:

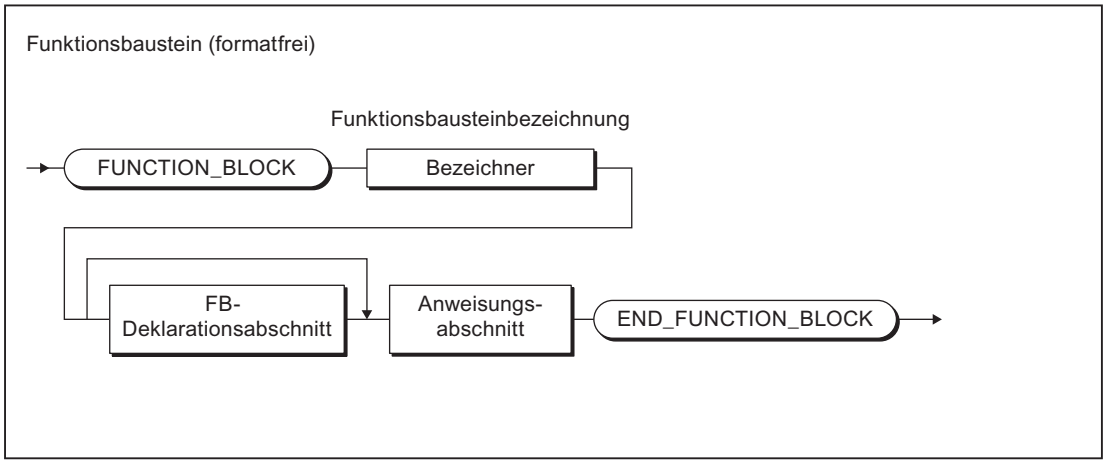


Bild 5-2 Syntax: Funktionsbaustein (FB)

Geben Sie nach dem Schlüsselwort FUNCTION_BLOCK einen Bezeichner als FB-Namen an.

Danach folgen (siehe Beispiel in Quelldatei mit Kommentaren (Seite 203)):

- der optionale Deklarationsabschnitt
- der Anweisungsabschnitt
- das Schlüsselwort END_FUNCTION_BLOCK

5.1.3 Deklarationsabschnitt von FB und FC

Ein Deklarationsabschnitt gliedert sich in unterschiedliche Deklarationsblöcke, die jeweils durch ein eigenes Schlüsselwortpaar gekennzeichnet sind. Jeder Block enthält eine Vereinbarungsliste für gleichartige Daten, wie z. B. Konstanten, lokale Variablen, Parameter. Ein Blocktyp darf nur einmal vorkommen, die Reihenfolge der Blöcke ist beliebig.

Für den Deklarationsabschnitt einer FC und eines FB haben Sie folgende Möglichkeiten (siehe auch Beispiel in Quelldatei mit Kommentaren (Seite 203)):

Erlaubte Deklarationsböcke

Tabelle 5-1 Deklarationsblöcke für FC und FB: Möglichkeiten

Daten	Syntax	FB	FC
Konstante	VAR CONSTANT *Vereinbarungsliste* END_VAR	X	X
Eingangsparameter	VAR_INPUT *Vereinbarungsliste* END_VAR	X	X

Daten	Syntax	FB	FC
Durchgangsparameter	VAR_IN_OUT *Vereinbarungsliste* END_VAR	X	X
Ausgangsparameter	VAR_OUTPUT *Vereinbarungsliste* END_VAR	X	–
Lokale Variable (für FC und FB)	VAR *Vereinbarungsliste* END_VAR	X (statisch)	X (temporär)
Lokale Variable (für FB)	VAR_TEMP *Vereinbarungsliste* END_VAR	X (temporär)	-
Vereinbarungsliste: die Liste der Bezeichner des Typs, der deklariert werden soll.			

Parameterblöcke

Parameter gehören zu den Lokaldaten und sind Formalparameter eines Funktionsbausteins oder einer Funktion. Wenn der FB oder die FC aufgerufen wird, ersetzen die Aktualparameter die Formalparameter und bilden somit einen Mechanismus zum Informationsaustausch zwischen den aufgerufenen und aufrufenden Quelldatei-Abschnitten.

- Formale Eingangsparameter nehmen die aktuellen Eingangswerte auf (Datenfluss von außen nach innen).
- Formale Ausgangsparameter (nur bei FB) dienen der Übergabe von Ausgangswerten (Datenfluss von innen nach außen).
- Formale Durchgangsparameter haben sowohl die Funktion eines Eingangs- als auch eines Ausgangsparameters.

Die folgenden Bilder zeigen die Syntax für die Parameterdeklaration eines FB bzw. einer FC.

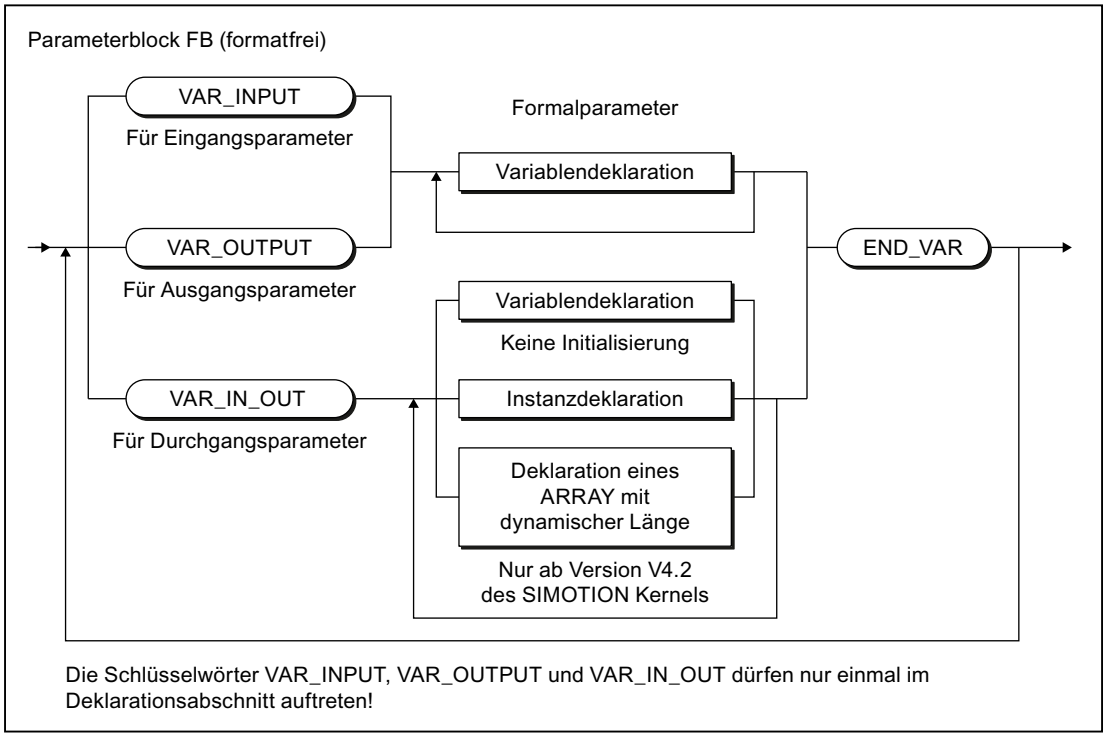


Bild 5-3 Syntax: Parameterblock FB

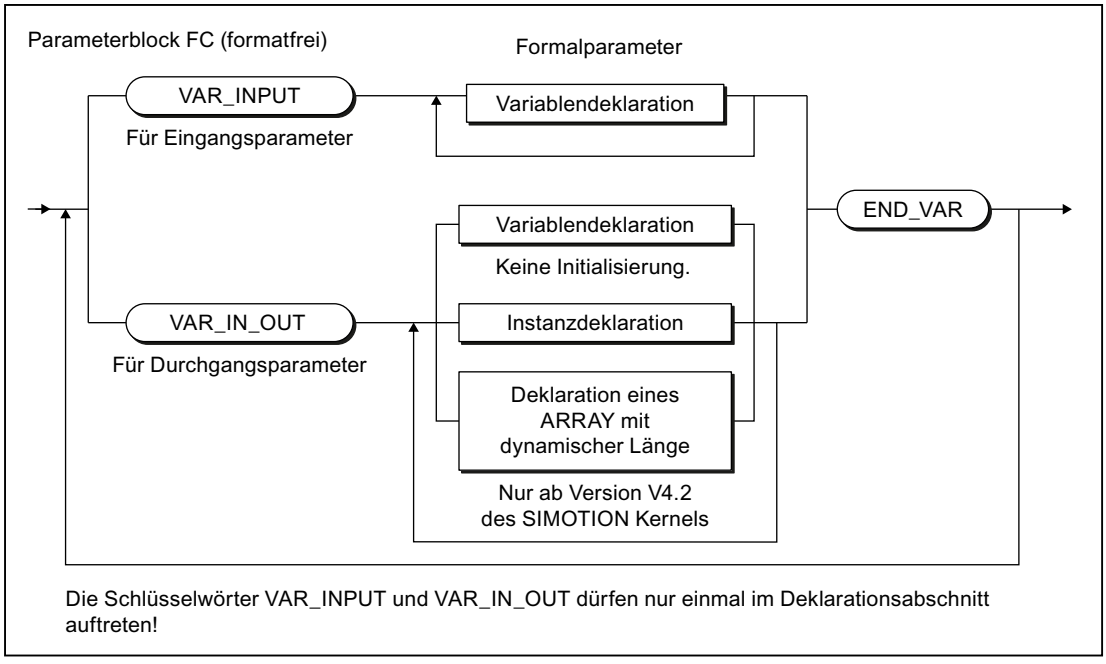


Bild 5-4 Syntax: Parameterblock FC

Im Parameterblock für Durchgangsparameter können Sie außer Variablen auch deklarieren:

- Instanzen von Funktionsbausteinen (Seite 198) ,
- Felder (ARRAY) mit dynamischer Länge (Seite 192) , ab Version V4.2 der SIMOTION Kernels.

Innerhalb eines FB oder FC können Sie die deklarierten Parameter wie andere Variablen verwenden, mit folgender Ausnahme: Sie können Eingangsparametern keine Werte zuweisen.

Von außerhalb eines FB oder einer FC können Sie zugreifen:

- Auf die Eingangs- und Ausgangsparameter eines FB mittels strukturierter Variablen (Seite 128).
  Der Zugriff auf die Eingangsparameter ist nur möglich, wenn die Compileroption (Seite 64) "Spracherweiterungen zulassen" aktiviert ist.
  Der Zugriff auf die Ausgangsparameter ist standardmäßig möglich.
- Auf den Rückgabewert einer FC, indem Sie die Funktion in einem Ausdruck verwenden und diesen z. B. einer Variablen zuweisen. Die Angabe des Funktionsnamens ruft die Funktion auf und liefert gleichzeitig ein Ergebnis.

5.1.4 Anweisungsabschnitt von FB und FC

Der Anweisungsabschnitt der FC oder des FB beinhaltet Anweisungen, die mit dem Aufruf zur Ausführung kommen. Es gibt keine Unterschiede zu den formalen Regeln zur Bildung eines Anweisungsabschnitts, inhaltlich sollten Sie jedoch die Hinweise in der folgenden Tabelle beachten.

Hinweis

Tipps zur effizienten Verwendung von Parametern siehe *Laufzeitoptimierende Programmierung* im Funktionshandbuch *SIMOTION Basisfunktionen*.

Tabelle 5-2 Verwendung von Parametern und Variablen in FC und FB

Parameter/Variable	Verwendung
Eingangsparameter	Beim Aufruf einer FC oder eines FB weisen Sie den Eingangsparametern aktuelle Werte zu. Diese Werte werden innerhalb der FC bzw. des FB für die Bearbeitung von Daten verwendet, beispielsweise für Berechnungen, können jedoch selbst nicht verändert werden. Nur bei aktivierter Compileroption "Spracherweiterungen zulassen" (siehe Globale Einstellungen des Compilers (Seite 64) oder Lokale Einstellungen des Compilers (Seite 67)): Auf die Eingangsparameter eines FB kann über strukturierte Variablen außerhalb auch des FB (z. B. im aufrufenden Quelldatei-Abschnitt) lesend und schreibend zugegriffen werden.
Durchgangsparameter	Beim Aufruf der FC oder des FB weisen Sie einem Durchgangsparameter eine Variable zu. Die FC bzw. der FB greift auf diese Variable direkt zu und kann diese unmittelbar verändern. Typkonvertierungen sind nicht möglich. Die einem Durchgangsparameter zugeordnete Variable muss direkt schreib- und lesbar sein. Systemvariablen (des SIMOTION Geräts oder eines Technologieobjekts), I/O-Variablen oder Prozessabbildzugriffe können deshalb einem Durchgangsparameter nicht zugeordnet werden.
Ausgangsparameter (nur bei FB)	Beim Aufruf eines FB weisen Sie mit dem Operator **=>** einen Ausgangsparameter einer Variablen zu. Der Wert des Ausgangsparameters (Ergebnis) wird nach Beenden des FB der Variablen übergeben. Außerdem kann auf die Ausgangsparameter eines FB über strukturierte Variablen außerhalb auch des FB (z. B. im aufrufenden Quelldatei-Abschnitt) lesend zugegriffen werden. Eine FC enthält formell keine Ausgangsparameter, da dem Funktionsnamen der Rückgabewert zugewiesen wird. Der Funktionsname selbst ist gewissermaßen der Ausgangsparameter.
Lokale Variablen	Lokale Variablen sind Variablen, die nur innerhalb des Bausteins deklariert und verwendet werden. In FC sind alle lokalen Variablen (VAR ... END_VAR) temporär, d. h. sie verlieren ihren Wert beim Beenden der FC. Beim nächsten Aufruf der FC werden sie neu initialisiert. In FB wird zwischen statischen und temporären lokalen Variablen unterschieden: • Statische Variablen (VAR ... END_VAR) behalten beim Beenden des FB ihren Wert. • Temporäre Variablen (VAR_TEMP ... END_VAR) verlieren beim Beenden des FB ihren Wert. Beim nächsten Aufruf des FB werden sie neu initialisiert. Der Wert der lokalen Variablen kann nicht direkt vom aufrufenden Baustein abgefragt werden. Dies ist nur über einen Ausgangsparameter möglich.

5.1.5 ARRAY mit dynamischer Länge (ab Kernel V4.2)

Deklaration als Durchgangsparameter

Ab Version 4.2 des SIMOTION Kernels können Sie Felder mit dynamischer Länge in Funktionen und Funktionsbausteinen deklarieren. Dies ist ausschließlich im jeweiligen Parameterblock für Durchgangsparameter (Seite 187) (VAR_IN_OUT / END_VAR) möglich.

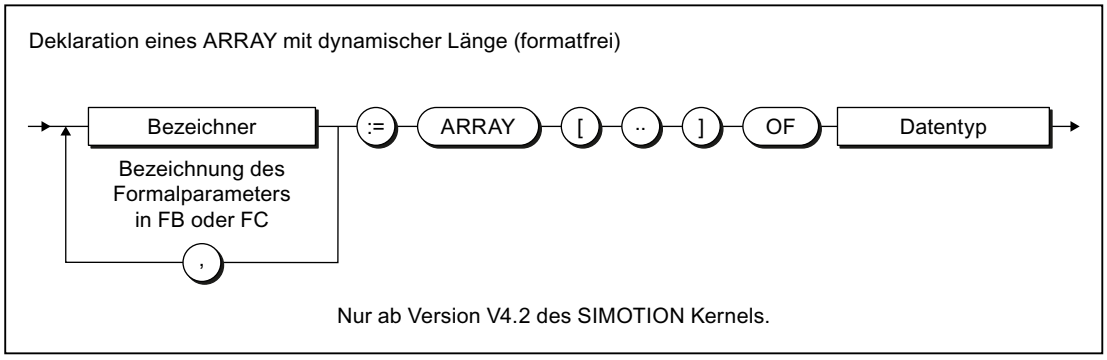


Bild 5-5 Syntax: Deklaration eines ARRAY mit dynamischer Länge

Verwendung eines ARRAY mit dynamischer Länge

Beim Aufruf der Funktion bzw. einer Instanz des Funktionsbausteins können Felder beliebiger Länge des deklarierten Datentyps übergeben werden. Auf dem Lokaldatenstack (Seite 245) werden außer der Referenz zum übergebenen Feld auch dessen Indexgrenzen gespeichert.

Innerhalb der Funktion bzw. Funktionsbausteins können Sie mit der Funktionen _firstIndexOf (in := *array-name*) bzw. _lastIndexOf (in := *array-name*) die untere bzw. obere Indexgrenze bestimmen. Die Funktion _lengthIndexOf (in := *array-name*) liefert die Anzahl der Elemente des ARRAYs. Es gilt: _lengthIndexOf (x) := _lastIndexOf (x) - _firstIndexOf (x) + 1.

Die Größe des Speicherplatzes, den das ARRAY belegt, können Sie mit _sizeOf (in := *array-name*) bestimmen. Es gilt: _sizeOf (in := *array_of_type*) := _lengthIndexOf (in := *array_of_type*) * _sizeOf (in := *type*).

Die Syntax dieser Funktionen ist im Funktionshandbuch "SIMOTION Basisfunktionen" beschrieben.

Hinweis

Bei einem ARRAY mit dynamischer Länge werden die obigen Funktionen zur Laufzeit ausgeführt.

Beispiel

Tabelle 5-3 Beispiel für Verwendung eines ARRAY mit dynamischer Länge

```
FUNCTION_BLOCK example_dyn_array
    VAR_IN_OUT
        flexarray : ARRAY [..] OF DINT;
    END_VAR

    VAR_TEMP
        i : DINT := 0;
    END_VAR

    i := _firstIndexOf (in := flexarray);

    WHILE i <= _lastIndexOf (in := flexArray) DO
        flexArray[i] := i;
        i := i + 1;
    END_WHILE;
END_FUNCTION_BLOCK

PROGRAM test_dyn_array
    VAR
        array_1 : ARRAY [0 .. 29] OF DINT;
        fb_example : example_dyn_array;
    END_VAR
    // ...
    fb_example (flexarray := array_1);
    // ...
END_PROGRAM
```

5.1.6 Aufruf von Funktionen und Funktionsbausteinen

Hier erhalten Sie einen Überblick über den Aufruf der Funktionen und Funktionsbausteine.

5.1.6.1 Prinzip der Parameterübergabe

Beim Aufruf von FC und FB findet ein Datenaustausch zwischen dem aufrufenden und dem aufgerufenen Baustein statt. Die Parameter, die übergeben werden sollen, müssen im Aufruf als Parameterliste angegeben werden. Die Parameter werden in Klammern geschrieben. Mehrere Parameter werden durch Kommata getrennt.

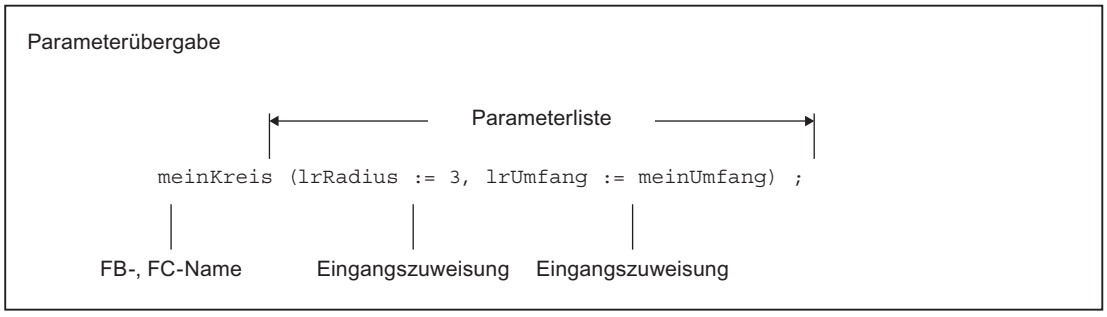


Bild 5-6 Prinzip der Parameterübergabe beim Aufruf

Eingangs- und Durchgangsparameter werden üblicherweise in Form einer Wertzuweisung angegeben. Hierdurch weisen Sie den Parametern, die Sie im Deklarationsabschnitt des aufgerufenen Bausteins definiert haben (Formalparameter), einen Wert (Aktualparameter) zu.

Die Zuweisung von Ausgangsparametern erfolgt mit dem Operator **=>**. Hierdurch weisen Sie die Ausgangsparameter, die Sie im Deklarationsabschnitt des aufgerufenen Bausteins definiert haben (Formalparameter), einer Variablen (Aktualparameter) zu.

5.1.6.2 Parameterübergabe zu Eingangsparametern

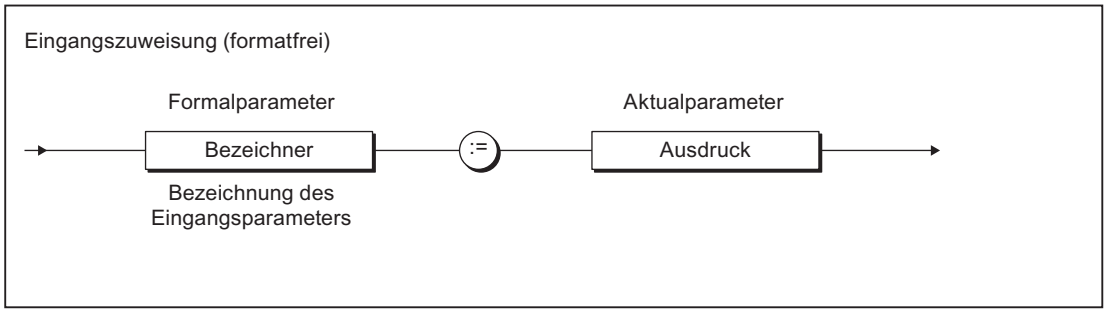


Bild 5-7 Syntax: Eingangszuweisung

Den formalen Eingangsparametern eines FB oder einer FC übergeben Sie die Daten (Aktualparameter) mittels Eingangszuweisungen. Die Aktualparameter können Sie in Form von Ausdrücken angeben. Die formalen Eingangsparameter können Sie in Anweisungen innerhalb des FB oder der FC verwenden, aber nicht verändern.

Es ist auch eine Kurzform der Parameterübergabe möglich, die Sie jedoch nicht bei selbstdefinierten FB verwenden sollten. Nötig ist diese Kurzform nur bei einigen FC, siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

Bei einem FB ist die Zuweisung von Aktualparametern optional. Wenn keine Eingangszuweisung angegeben wird, dann bleiben die Werte des letzten Aufrufs erhalten, da der FB ein Quelldatei-Abschnitt mit Gedächtnis ist.

Bei einer FC ist die Zuweisung eines Aktualparameters optional, wenn bei der Deklaration des Formalparameters ein Initialisierungsausdruck angegeben wurde.

Siehe auch Beispiele in Funktionen aufrufen (Seite 197) bzw. Funktionsbausteine aufrufen (Instanzen aufrufen) (Seite 198).

Sie können auch außerhalb eines FB jederzeit auf dessen Eingangsparameter lesend und schreibend zugreifen. Siehe hierzu: Außerhalb des FB auf dessen Eingangsparameter zugreifen (Seite 200).

5.1.6.3 Parameterübergabe zu Durchgangsparametern

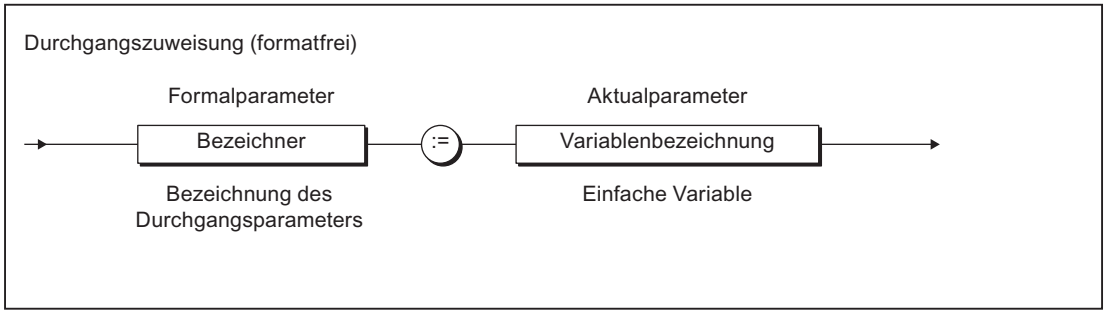


Bild 5-8 Syntax: Durchgangszuweisung

Den formalen Durchgangsparametern einer FC oder eines FB übergeben Sie die Daten (Aktualparameter) mittels Durchgangszuweisungen. Sie können dem formalen Durchgangsparameter nur eine typgleiche Variable zuweisen, Datentypkonvertierungen sind nicht möglich.

Die formalen Durchgangsparameter können Sie in Anweisungen innerhalb der FC bzw. des FB verwenden und verändern. Die FC bzw. der FB greift direkt auf die Variable des Aktualparameters zu und kann diese unmittelbar verändern.

Siehe auch Beispiele in Funktionen aufrufen (Seite 197) bzw. Funktionsbausteine aufrufen (Instanzen aufrufen) (Seite 198).

Bei Verwendung des Datentyps STRING in Durchgangszuweisungen muss die deklarierte Länge des Aktualparameters größer oder gleich der Länge des formalen Durchgangsparameters sein (siehe nachfolgendes Beispiel).

Tabelle 5-4 Beispiel für die Verwendung des Datentyps STRING in Durchgangszuweisungen

```
FUNCTION_BLOCK REF_STRING
    VAR_IN_OUT
        io : STRING[80];
    END_VAR
;    // Anweisungen
END_FUNCTION_BLOCK

FUNCTION_BLOCK test
    VAR
        my_fb : REF_STRING;
        str1 : STRING [100];
        str2 : STRING [50] ;
    END_VAR
    my_fb(io := str1);    // Zulässiger Aufruf
    my_fb(io := str2);    // Nicht zulässiger Aufruf,
                          // Compilerfehlermeldung

END_FUNCTION_BLOCK
```

Die einem Durchgangsparameter zugeordnete Variable muss direkt schreib- und lesbar sein. Systemvariablen (des SIMOTION Geräts oder eines Technologieobjekts), I/O-Variablen oder Prozessabbildzugriffe können deshalb einem Durchgangsparameter nicht zugeordnet werden.

Beachten Sie die unterschiedlichen Zugriffszeiten auf Parameter!

5.1.6.4 Parameterübergabe zu Ausgangsparametern (nur bei FB)

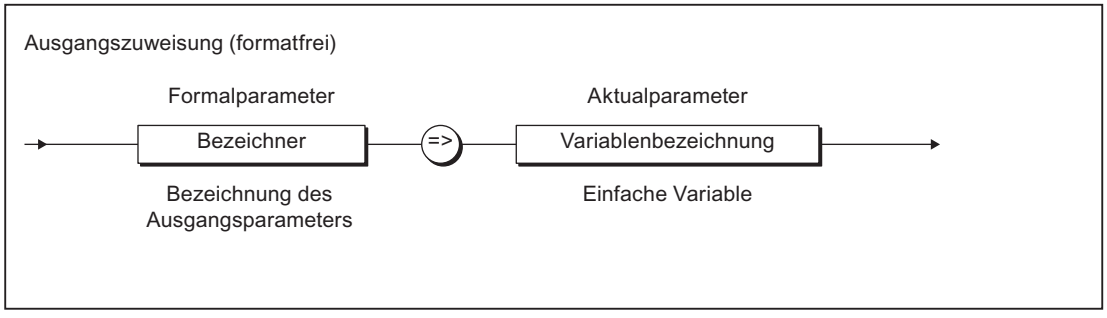


Bild 5-9 Syntax: Ausgangzuweisung

Die formalen Ausgangsparametern eines FB weisen Sie mit einer Ausgangszuweisung den Variablen (Aktualparameter) zu, die nach Beendigung des FB den Wert des formalen Ausgangsparameters aufnehmen.

Die formalen Ausgangsparameter können Sie in Anweisungen innerhalb des FB verwenden und verändern.

Siehe auch Beispiele in Funktionsbausteine aufrufen (Instanzen aufrufen) (Seite 198).

Ausgangszuweisungen sind bei der Parameterübergabe optional. Sie können auch außerhalb eines FB jederzeit auf dessen Ausgangsparameter lesend zugreifen. Siehe hierzu: Außerhalb des FB auf dessen Ausgangsparameter zugreifen (Seite 200)).

5.1.6.5 Zugriffszeiten auf Parameter

Die Zugriffsarten und damit Zugriffszeiten auf Parameter sind unterschiedlich:

- Bei Eingangszuweisungen werden die Werte der Aktualparameter in die Formalparameter kopiert. Wenn größere Strukturen, wie z. B. Arrays, kopiert werden und die FC oder der FB oft aufgerufen werden, kann dies die Performance mindern.
- Durchgangszuweisungen kopieren keine Werte, sondern verknüpfen nur Speicheradressen der Formalparameter mit denen der Aktualparameter. Die Übergabe der Variablen ist dadurch schneller als bei Eingangszuweisungen (besonders bei größeren Datenmengen). Der Zugriff auf die Variable vom FB aus ist unter Umständen jedoch langsamer.
- Wenn Sie Unit-Variablen verwenden, werden keine Kopiervorgänge in die Funktion oder den Funktionsbaustein durchgeführt, da diese Variablen in der gesamten ST-Quelle gültig sind (siehe Variablenmodell (Seite 230)).

Hinweis

Das Verwenden von Durchgangsparametern anstelle von Eingangsparametern bringt nur dann Geschwindigkeitsvorteile, wenn ein großes Datenvolumen an den Funktionsbaustein zu übergeben ist.

Die überwiegende Verwendung von Unit-Variablen anstelle von Parametern macht die Programmstruktur unübersichtlich: Objektorientierung, Datenkapselung, mehrfaches Verwenden von Variablennamen (Kapselung von Gültigkeitsbereichen) usw. sind nicht mehr möglich.

5.1.6.6 Funktionen aufrufen

Eine Funktion wird folgendermaßen aufgerufen:

- Funktion mit Rückgabewert (Datentyp verschieden von **VOID**):
  Die Funktion steht auf der rechten Seite einer Wertzuweisung. Sie kann auch als Operand innerhalb eines Ausdrucks auftreten. Nach Aufruf der Funktion wird deren Rückgabewert an der entsprechenden Stelle zur Berechnung des Ausdrucks verwendet.
  Beispiele:

```
y := sin(x);
y := sin(in := x);
y := sqrt (1 - cos(x) * cos(x));
```

- Funktion ohne Rückgabewert (Datentyp **VOID**)
  Die Anweisung besteht nur aus dem Funktionsaufruf.
  Das nachstehende Beispiel ist gültig unter der Voraussetzung, dass eine Funktion funct1 mit dem Eingangsparametern in1 und in2 und dem Durchgangsparametern inout bereits definiert wurde.
  Beispiel:

```
funct1 (in1 := var11, in2 := var12, inout1 := var13);
```

Hinweis

In der Funktion selbst wird das Ergebnis (Rückgabewert) dem Funktionsnamen zugewiesen (außer bei Datentyp VOID).

5.1.6.7 Funktionsbausteine aufrufen (Instanzen deklarieren und aufrufen)

Instanz eines Funktionsbausteins deklarieren

Bevor Sie einen Funktionsbaustein (FB) aufrufen, müssen Sie eine Instanz deklarieren. Sie deklarieren eine Variable und geben als Datentyp den Namen des Funktionsbausteins ein. Diese Instanzdeklaration führen Sie durch:

- lokal (innerhalb VAR / END_VAR im Deklarationsabschnitt eines Programms oder Funktionsbausteins),
- global (innerhalb VAR_GLOBAL / END_VAR im Interface- oder Implementationsabschnitt),
- als Durchgangsparameter (innerhalb VAR_IN_OUT/ END_VAR im Deklarationsabschnitt eines Funktionsbausteins oder einer Funktion).

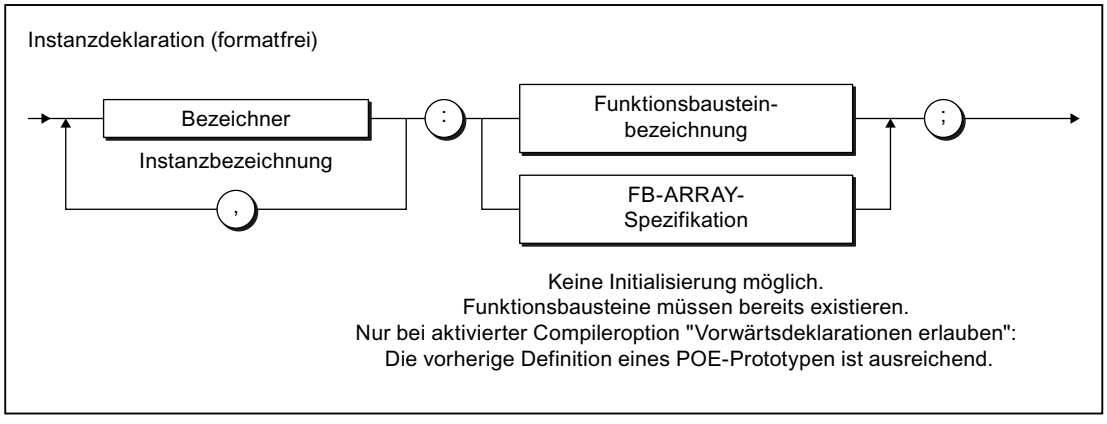


Bild 5-10 Syntax: Instanzdeklaration

Die Instanzdeklaration ist auch als Feld möglich, z. B.:

```
FB_inst : ARRAY [1..2] OF FB_name.
```

Hinweis

Achten Sie auf die unterschiedlichen Initialisierungs-Zeitpunkte verschiedener Variablentypen.

Instanz eines Funktionsbausteins aufrufen

Im Anweisungsabschnitt einer POE rufen Sie die Instanz des Funktionsbausteins auf (Syntax siehe Bild). FB-Parameter sind die durch Kommata getrennten Eingangs- und Durchgangszuweisungen.

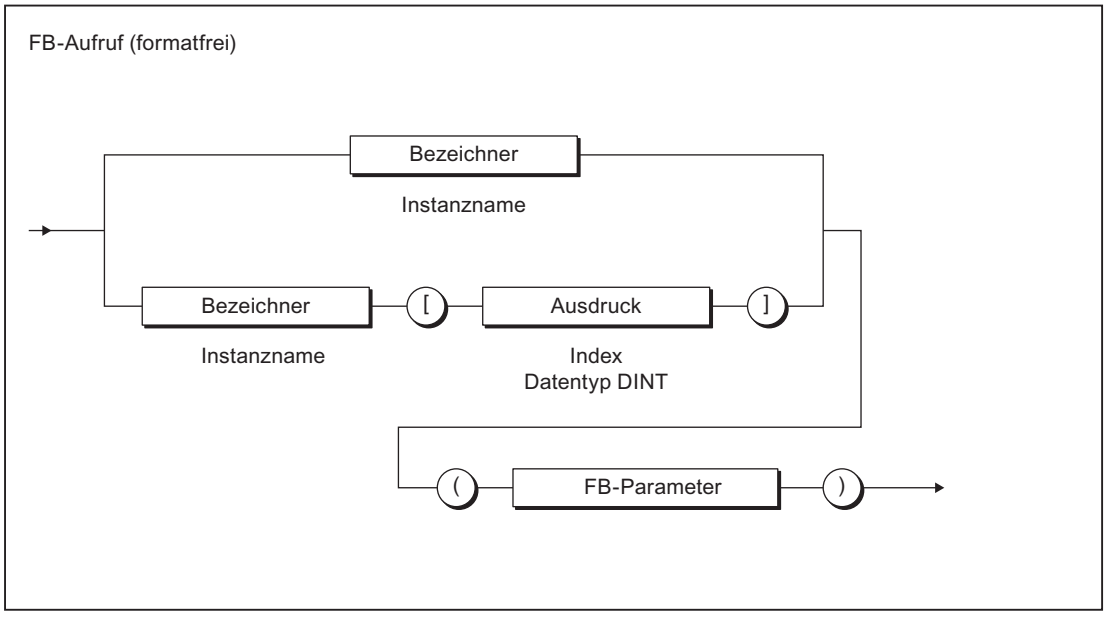


Bild 5-11 Syntax FB-Aufruf

Das Beispiel der folgenden Tabelle ist gültig unter der Voraussetzung, dass die Funktionsbausteine *Supply und Motor* bereits definiert wurden:

- FB Supply:
  Eingangsparameter in1, in2; Durchgangsparameter inout; Ausgangsparameter out
- FB Motor:
  Durchgangsparameter inout1, inout2; Ausgangsparameter out1, out2

Tabelle 5-5 Beispiel für Instanzdeklaration, FB-Aufruf und Zugriff auf Ausgangsparameter

```
VAR
    Supply1, Supply2: Supply;
    Motor1 : Motor;
END_VAR

// Parameterübergabe (Ausgangszuweisung) beim Aufruf der Instanz eines FB
Supply1 (in1 := var11, in2 := expr12, inout := var13, out => var14) ;
Supply2 (in1 := var21, in2 := expr22, inout := var23, out => var24) ;
Motor1 (inout1 := var31, inout2 := var32, out1 => var33, out2 => var34);
// ...
// Außerhalb des FB auf dessen Ausgangsparameter zugreifen
var15 := Supply1.out;
var25 := Supply2.out;
var35 := Motor1.out1;
var36 := Motor1.out2;
var41 := Motor1.out1 * Motor1.out2 * (Supply1.out + Supply2.out);
```

5.1.6.8 Außerhalb des FB auf dessen Ausgangsparameter zugreifen

Neben der Ausgangszuweisung (Seite 196) beim Aufruf eines FB ist es jederzeit möglich, außerhalb eines FB auf dessen Ausgangsparameter lesend zuzugreifen.

Verwenden Sie hierzu strukturierte Variablen (Seite 128) im Format *FB-Instanzname.Ausgangsparameter*, z. B. *Supply1.out*.

Siehe auch Beispiele in Funktionsbausteine aufrufen (Instanzen aufrufen) (Seite 198).

Der Instanzname des FB selbst darf in einer Wertzuweisung nicht verwendet werden!

5.1.6.9 Außerhalb des FB auf dessen Eingangsparameter zugreifen

Neben der Eingangzuweisung (Seite 194) beim Aufruf eines FB ist es möglich, außerhalb eines FB auf dessen Eingangsparameter lesend und schreibend zuzugreifen.

Verwenden Sie hierzu strukturierte Variablen (Seite 128) im Format *FB-Instanzname.Eingangsparameter*, z. B. *Supply1.in1*.

Hinweis

Um diese Möglichkeit nutzen zu können, muss die Compileroption "Spracherweiterungen zulassen" aktiviert sein, siehe Globale Einstellungen des Compilers (Seite 64) und Lokale Einstellungen des Compilers (Seite 67).

Der Instanzname des FB selbst darf in einer Wertzuweisung nicht verwendet werden!

Tabelle 5-6 Beispiel für Zuweisung an Eingangsparameter

```
// Nur mit aktivierter Compileroption "Spracherweiterungen zulassen"
VAR
    var_fb   : _WORD_TO_2BYTE;
    var_word : WORD;
END_VAR
var_fb.wordin := var_word;
// ..
var_fb();
```

5.1.6.10 Fehlerquellen beim Aufruf eines FB

Beachten Sie beim Aufruf einer Instanz eines FB:

- **Weisen Sie Durchgangsparametern nur Variablen zu, die direkt im Speicher hinterlegt sind!**
  Erlaubt sind als Aktualparameter nur:
  - globale Variablen (Unit-Variablen und geräteglobale Anwendervariablen),
  - lokale Variablen
  - Variablen vom Datentyp der TO (TO-Instanzen).

  Nicht möglich sind insbesondere:
  - Systemvariablen (TO-Variablen),
  - Namen technologischer Objekte aus ES,
  - I/O-Variablen
  - absolute und symbolische Prozessabbildzugriffe
- **Verwenden Sie keine Funktion (FC) als Durchgangsparameter!**
  Der FC-Rückgabewert, d. h. der Aufruf der FC, kann nicht als Aktualparameter in einer Durchgangszuweisung stehen. Sie müssen deshalb das Ergebnis der FC vorher in einer lokalen Variablen speichern und diese Variable als Aktualparameter in der Durchgangszuweisung verwenden.
- **Verwenden Sie keine Konstanten als Durchgangsparameter!**
  Sie dürfen nur Variablen als Aktualparameter einer Durchgangszuweisung verwenden, da der Wert zurückgeschrieben wird.
- **Durchgangsparameter können Sie nicht initialisieren.**

5.2 Funktionen und Funktionsbausteine im Vergleich

Die nachfolgenden Ausführungen zeigen Ihnen in knapper Form und anhand eines vollständigen Beispiels die Unterschiede zwischen selbst definierten Funktionsbausteinen (FB) und Funktionen (FC).

5.2.1 Beschreibung des Beispiels

Das folgende Beispiel verdeutlicht die Unterschiede zwischen FB und FC. Der Einfachheit halber wird jede Art Parameter nur einmal eingesetzt, tatsächlich können Sie beliebig viele Parameter definieren. Erklärungen zu den verwendeten Begriffen finden Sie bei den detaillierten Ausführungen in Funktionen definieren (Seite 186) und Funktionsbausteine definieren (Seite 186).

Im Vereinbarungsteil im Implementationsabschnitt soll ein Baustein als FB und FC erstellt werden, der zu der Eingangsgröße Radius den Umfang und die Fläche des Kreises berechnet:

- Für den Radius wird ein Eingangsparameter definiert.
- Für den Umfang des Kreises wird ein Durchgangsparameter definiert, d. h. während des Aufrufs von FB bzw. FC wird der Wert der übergebenen Variable direkt zugewiesen.
- Für die Fläche des Kreises stehen beim FB und bei der FC unterschiedliche Möglichkeiten zur Verfügung:
  - Beim FB wird ein Ausgangsparameter definiert.
  - Bei der FC wird der Rückgabewert der FC verwendet; der Datentyp des Rückgabewerts wird entsprechend definiert.
- Jeder Aufruf des FB und der FC soll in einem Zähler festgehalten werden (lokale Variable). In den Erläuterungen zum Beispiel ist abgegeben, Wir werden sehen, dass dieser Wert nur im FB weitergezählt wird.
- Im Programm-Abschnitt wird der FB bzw. die FC aufgerufen und Aktualparameter den nachstehend angegebenen Formalparametern zugeordnet:
  - Beim FB: Eingangs-, Durchgangs- und Ausgangsparameter
  - Bei der FC: Eingangs- und Durchgangsparameter.

  Nach Aufruf des FB oder der FC sind die Werte für Umfang und Fläche verfügbar:
  - Beim FB: in den Aktualparametern des Durchgangs- und Ausgangsparameters. Auf den Ausgangsparameter kann auch außerhalb des FB lesend zugegriffen werden.
  - Bei der FC: im Rückgabewert der Funktion sowie im Aktualparameter des Durchgangsparameters.

5.2.2 Quelldatei mit Kommentaren

Tabelle 5-7 Beispiel für die Unterschiede zwischen FB und FC

Funktionsbaustein (FB)

```
INTERFACE
  PROGRAM CircleCalc1;
END_INTERFACE
IMPLEMENTATION
  FUNCTION_BLOCK Circle1
    // Konstantendeklaration
    VAR CONSTANT
      PI : LREAL := 3.1415 ;
    END_VAR
    // Eingangsparameter
    VAR_INPUT
      Radius : LREAL ;
    END_VAR
    // Durchgangsparameter
    VAR_IN_OUT
      circumference : LREAL ;
    END_VAR
    // Ausgangsparameter
    VAR_OUTPUT
      Area : LREAL ;
    END_VAR
    // Lokale Variablen, statisch
    VAR
      Counter : DINT ;
      (* Variable behält ihren Wert
      über Aufrufgrenzen hinweg *)
    END_VAR
    // Aufrufzähler
    Counter := Counter + 1 ;
    Circumference := 2 * PI * Radius ;
    Area := PI * Radius**2 ;
  END_FUNCTION_BLOCK
  PROGRAM CircleCalc1
    VAR
      myCircle1        : Circle1 ;
      myArea1, myArea2 : LREAL ;
      myCircf          : LREAL ;
    END_VAR ;
    myCircle1(Radius := 3
      , Circumference := myCircf
      , Area => myArea1) ;
    myArea2 := myCircle1.Area ;
    // myCircf hat den Wert 18.849
    // myArea1 hat den Wert 28.274
    // myArea2 hat den Wert 28.274
  END_PROGRAM
END_IMPLEMENTATION
```

Funktion (FC)

```
INTERFACE
  PROGRAM CircleCalc2;
END_INTERFACE
IMPLEMENTATION
  FUNCTION Circle2 : LREAL
    // Konstantendeklaration
    VAR CONSTANT
      PI : LREAL := 3.1415 ;
    END_VAR
    // Eingangsparameter
    VAR_INPUT
      Radius : LREAL ;
    END_VAR
    // Durchgangsparameter
    VAR_IN_OUT
      circumference : LREAL ;
    END_VAR
    // Ausgangsparameter
    // nicht möglich


    // Lokale Variablen, temporär
    VAR
      Counter : DINT ;
      (* Variable wird bei jedem
      Aufruf mit 0 initialisiert *)
    END_VAR
    // Aufrufzähler
    Counter := Counter + 1 ;
    Circumference := 2 * PI * Radius ;
    Circle2 := PI * Radius**2 ;
  END_FUNCTION
  PROGRAM CircleCalc2
    VAR

      myArea  : LREAL ;
      myCircf : LREAL ;
    END_VAR ;
    myArea := Circle2(Radius := 3
      , Circumference := myCircf);


    // myCircf hat den Wert 18.849
    // myArea hat den Wert 28.274

  END_PROGRAM
END_IMPLEMENTATION
```

Tabelle 5-8 Erläuterungen zum vorstehenden Beispiel für die Unterschiede zwischen FB und FC

Funktionsbaustein (FB)	Funktion (FC)
Kommentare	
Reservierte Wörter für die Definition: FUNCTION_BLOCK und END_FUNCTION_BLOCK	Reservierte Wörter für die Definition: FUNCTION und END_FUNCTION
Kein Rückgabewert erlaubt.	Nach dem Namen muss der Datentyp des Rückgabewertes angegeben werden (Datentyp VOID, falls kein Rückgabewert).
Eingangsparameter ermöglichen die Übergabe von Werten an den FB.	Eingangsparameter ermöglichen die Übergabe von Werten an die FC.
Durchgangsparameter ermöglichen, dass im FB auf die übergebene Variable lesend und schreibend zugegriffen werden kann.	Durchgangsparameter ermöglichen, dass in der FC auf die übergebene Variable lesend und schreibend zugegriffen werden kann.
Ausgangsparameter ermöglichen die Rückgabe von Werten aus einem FB.	Keine Ausgangsparameter erlaubt.
Lokale Variablen sind statisch, d. h. sie behalten ihren Wert bis zum nächsten Aufruf des FB. Die lokale Variable *Counter* wird hochgezählt; ihr Wert bleibt bei Beenden des FB gespeichert. Bei jedem Aufruf wird deshalb die Variable weiter hochgezählt. So sehen Sie dieses Verhalten: Weisen Sie im FB den Wert der lokalen Variablen einer globalen Variablen zu. Beobachten Sie den Wert der globalen Variablen nach mehrmaligem Aufruf des FB.	Lokale Variablen sind temporär, d. h. sie verlieren ihren Wert beim Beenden der Funktion. Die lokale Variable *Counter* wird zwar hochgezählt; ihr Wert geht bei Verlassen der FC jedoch verloren. Beim nächsten Aufruf der FC wird die Variable neu initialisiert (im Beispiel auf 0). So sehen Sie dieses Verhalten: Weisen Sie in der FC den Wert der lokalen Variablen einer globalen Variablen zu. Der Wert der globalen Variablen bleibt nach mehrmaligem Aufruf der FC unverändert.
Im Anweisungsabschnitt werden die Ergebnisse (Rückgabewerte) den Ausgangs- oder Durchgangsparametern zugewiesen.	Im Anweisungsabschnitt wird das Ergebnis (Rückgabewert) dem Funktionsnamen zugewiesen (außer bei Datentyp VOID).
Im Vereinbarungsteil des aufrufenden Bausteins wird eine Instanz des FB deklariert: Sie deklarieren eine Variable; als Datentyp geben Sie den Namen des FB an. Den so vereinbarten Instanznamen verwenden Sie zum Aufruf des FB und für den Zugriff auf dessen Ausgangsparameter. Der Name des FB selbst darf im Anweisungsabschnitt nicht verwendet werden.	

Funktionsbaustein (FB)	Funktion (FC)
Kommentare	
• Den Durchgangsparametern weisen Sie beim Aufruf der Instanz des FB eine Variable zu. • Die Ausgangsparameter können Sie beim Aufruf einer Variablen zuweisen. • Auch außerhalb des FB können Sie auf dessen Ausgangsparameter lesend zugreifen. Verwenden Sie hierzu strukturierte Variablen in folgendem Format: *FB-Instanzname.Ausgangsparameter*.	• Den Durchgangsparametern weisen Sie beim Aufruf der Instanz des FB eine Variable zu. • Den Rückgabewert einer FC erhalten Sie folgendermaßen: – Sie weisen die Funktion einer Variablen zu. – Sie verwenden die Funktion in einem Ausdruck auf der rechten Seite einer Wertzuweisung.
Auf andere Variablen als die Durchgangs- und Ausgangsparameter des FB kann das aufrufende Programm nicht zugreifen. Ausnahme: Bei aktivierter Compileroption "Spracherweiterungen zulassen" (siehe Globale Einstellungen des Compilers (Seite 64) oder Lokale Einstellungen des Compilers (Seite 67)) kann das aufrufende Programm auch auf die Eingangsparameter eines FB zugreifen. Verwenden Sie hierzu strukturierte Variablen in folgendem Format: *FB-Instanzname.Eingangsparameter*.	Auf andere Variablen als den Rückgabewert kann das aufrufende Programm nicht zugreifen.

5.3 Programme

Programme sind eine Folge von Anweisungen zwischen den Schlüsselwörtern PROGRAM und END_PROGRAM.

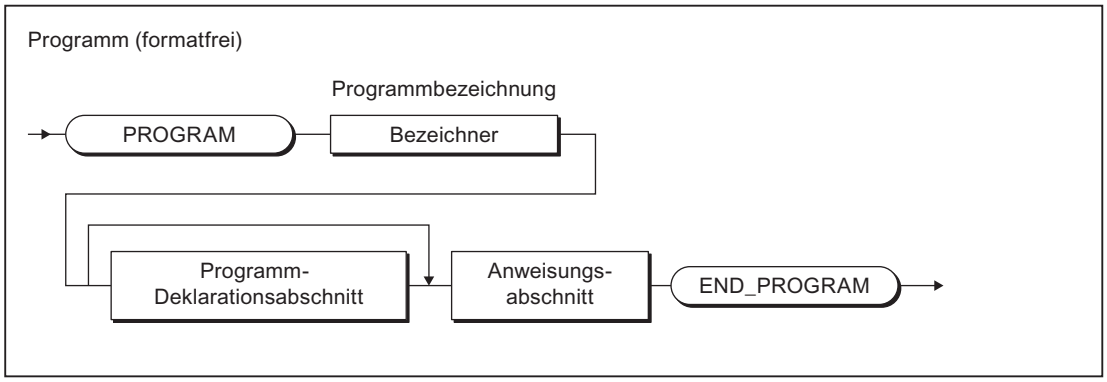


Bild 5-12 Syntax: Programm

Programme werden im Implemenationsabschnitt (Seite 214) einer ST-Quelle vereinbart und sind den FB vergleichbar. Beispielsweise können auch hier lokale Variablen statisch (VAR...END_VAR) oder temporär (VAR_TEMP...END_VAR) angelegt werden. Sie besitzen jedoch keine Formalparameter und können deshalb nicht mit Parametern aufgerufen werden. Beispiele für Programme finden Sie in den Abschnitten Quelldatei mit Kommentaren (Seite 203) und Quelltext des Beispielprogramms (Seite 89).

5.3.1 Zuordnung eines Programms im Ablaufsystem

Standardmäßig werden Programme im Ablaufsystem einer Task zugeordnet. Das Ablaufverhalten der Programme, z. B. Initialisierung der Variablen, wird von der jeweiligen Task bestimmt. Näheres zum Ablaufsystem und zu den Tasks siehe Funktionshandbuch *SIMOTION Basisfunktionen*. Hierfür muss das Programm im Interfaceabschnitt (Seite 212) der ST-Quelle als zu exportierende Programmorganisationseinheit angegeben werden.

5.3.2 Aufruf eines Programms im Programm ("program in program")

Optional kann ein Programm auch innerhalb eines anderen Programms oder eines Funktionsbausteins aufgerufen werden. Hierzu müssen folgende Compileroptionen aktiviert sein (siehe Globale Einstellungen des Compilers (Seite 64) und Lokale Einstellungen des Compilers (Seite 67)):

1. "Spracherweiterungen zulassen" für die Programmquelle des aufrufenden Programms bzw. Funktionsbaustein und
2. "Programminstanzdaten nur einmal anlegen" für die Programmquelle des aufgerufenen Programms.

Der Aufruf erfolgt wie eine Funktion ohne Parameter und Rückgabewert; siehe nachfolgendes Beispiel.

Hinweis

Die aktivierte Compileroption "Programminstanzdaten nur einmal anlegen" bewirkt:

- Die statischen Variablen der Programme (Programminstanzdaten) werden in einem anderen Speicherbereich (Seite 240) abgelegt. Hierdurch ändert sich auch das Initialisierungsverhalten (Seite 251).
- Alle aufgerufenen Programme gleichen Namens verwenden dieselben Programminstanzdaten.

Tabelle 5-9 Beispiel für Aufruf eines Programms in einem Programm

```
PROGRAM my_prog
    ; // ...
END_PROGRAM

PROGRAM main_prog
    ; // ...
    my_prog();
    ; // ...
END_PROGRAM
```

Hinweis

Durch den Aufruf von Programmen innerhalb eines Programms kann die Zuordnung der Programme zu den Tasks weitgehend programmiert werden. Im Ablaufsystem muss nur jeweils ein aufrufendes Programm den betreffenden Tasks zugeordnet werden.

5.4 Expressions

Die Expression ist ein Spezialfall einer Funktionsvereinbarung:

- Der Datentyp des Rückgabewerts ist als BOOL festgelegt und wird nicht explizit angegeben.

Sie wird im Zusammenhang mit der WAITFORCONDITION-Anweisung (Seite 176) verwendet.

Eine Expression kann ausschließlich im Implementationsabschnitt der ST-Quelle vereinbart werden.

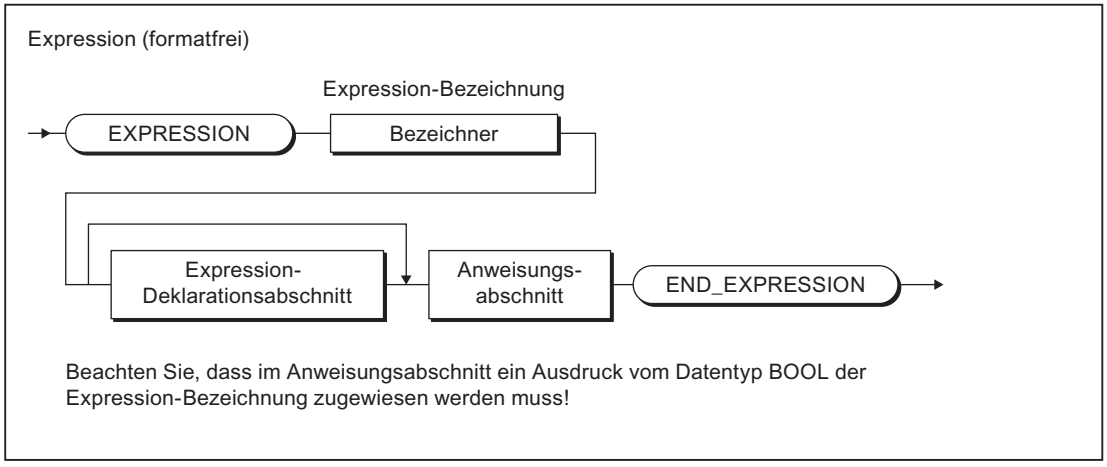


Bild 5-13 Syntax: Expression

Optional können im Deklarationsabschnitt deklariert werden:

- lokale (temporäre) Variablen
- lokale Konstanten
- anwenderdefinierte Datentypen (UDT)
- Eingangs- und Durchgangsparameter (ab Version V4.1 des SIMOTON-Kernels)

Im Anweisungsabschnitt kann zugegriffen werden:

- auf die lokalen Variablen und Konstanten der Expression
- auf die Eingangs- und Durchgangsparameter (sofern deren Deklaration zulässig ist)
- auf Unit-Variablen
- auf geräteglobale Variablen, I/O-Variablen und das Prozessabbild

Im Anweisungsabschnitt der Expression muss ein Ausdruck vom Datentyp BOOL der Expression-Bezeichnung zugeordnet werden (siehe Bild).

Hinweis

Der Anweisungsabschnitt der Expression darf keine Funktionsaufrufe oder Schleifen enthalten.

Beispiel

Im folgenden Beispiel wird unterstellt, dass das Programm feeder in einer MotionTask abläuft. Für diese MotionTask ist die Option Aktivierung nach StartupTask ausgewählt. Die Zuordnung von Programmen zu Tasks nehmen Sie im SIMOTION SCOUT vor (siehe Funktionsbeschreibung SIMOTION Motion Control Basisfunktionen).

Tabelle 5-10 Beispiel für Verwendung einer EXPRESSION und der WAITFORCONDITION-Anweisung

```
INTERFACE
    USEPACKAGE cam;
    PROGRAM feeder; // in MotionTask_1
END_INTERFACE

IMPLEMENTATION
    // Bedingung für WAITFORCONDITION-Anweisung
    EXPRESSION automaticExpr
        automaticExpr := IOfeedCam; // Digitaleingang
    END_EXPRESSION

    PROGRAM feeder
        VAR
            retVal : DINT ;
        END_VAR ;
        retVal := _enableAxis (axis := realAxis,
            enableMode := ALL,
            servoCommandToActualMode := INACTIVE,
            nextCommand := WHEN_COMMAND_DONE,
            commandId := _getCommandId() );

        // Warte, bis Startbedingung erfüllt ist
        WAITFORCONDITION automaticExpr WITH TRUE DO
            // Hochpriorige Ausführung aller Anweisungen
            // bis zum Befehl END_WAITFORCONDITION
            retVal := _pos (axis := realAxis,
                positioningMode := RELATIVE,
                position := 500,
                velocityType := DIRECT,
                velocity := 300,
                velocityProfile := TRAPEZOIDAL,
                mergeMode := IMMEDIATELY,
                nextCommand := WHEN_MOTION_DONE,
                commandId:= _getCommandId() );
        END_WAITFORCONDITION;

        retVal := _disableAxis (axis := realAxis,
            disableMode := ALL,
            servoCommandToActualMode := INACTIVE,
            nextCommand := WHEN_COMMAND_DONE,
            commandId := _getCommandId() );
    END_PROGRAM
END_IMPLEMENTATION
```

Weitere Beispiele sind im Funktionshandbuch SIMOTION Motion Control Basisfunktionen enthalten. Insbesondere ist dort beschrieben, wie Sie ab Version V4.1 des SIMOTION Kernels

eine EXPRESSION mit Parametern verwenden und z. B. eine Zeitüberwachung in einer WAITFORCONDITION-Anweisung programmieren.

Einbindung von ST in SIMOTION 6

Sie erfahren in diesem Kapitel, welches Zusammenspiel zwischen ST-Programmen und dem SIMOTION SCOUT besteht.

6.1 Quelldatei-Abschnitte

Einen Überblick über die Bedeutung der Quelldatei-Abschnitte haben Sie bereits in Gliederung der ST-Quelle (Seite 113) bekommen. Hier erfahren Sie Einzelheiten, z. B. die Syntax der Abschnitte und wie Sie sie für den Import und Export von Daten zwischen mehreren ST-Quelldateien verwenden.

6.1.1 Verwendung der Quelldatei-Abschnitte

Damit sich Ihre ST-Quelldatei übersetzen lässt, müssen Sie bestimmte Strukturen und Syntaxvorschriften ihrer Abschnitte beachten. Hier einige allgemeine Vorschriften; Einzelheiten bezüglich der Quelldatei-Abschnitte folgen weiter unten:

- Achten Sie bei der Erstellung der Quelldatei immer auf die Reihenfolge der Quelldatei-Abschnitte. Ein aufzurufender Abschnitt muss immer vor dem aufrufenden Abschnitt stehen, damit er diesem bekannt ist.
  Beispielsweise müssen Variablen vor der Verwendung deklariert, Funktionen vor dem Aufruf definiert werden.
- Der Quelltext für die häufigsten Quelldatei-Abschnitte – Programm, Funktion oder Funktionsbaustein – besteht aus folgenden Teilen:
  - Abschnittsanfang mit reserviertem Wort und Bezeichner
  - Deklarationsabschnitt (optional)
  - Anweisungsabschnitt
  - Abschnittsende mit reserviertem Wort
- Bezeichner für Quelldatei-Abschnitte – nachfolgend mit *name* oder *name_list* bezeichnet – folgen der allgemeinen Syntax für Bezeichner (siehe Bezeichner im ST (Seite 100)).

Hinweis

Eine Vorlage mit allen möglichen Quelldatei-Abschnitten ist über die Online-Hilfe verfügbar.

6.1.1.1 Interfaceabschnitt

Der Interfaceabschnitt enthält Anweisungen für den Import und Export von Daten (Datentypen, Variablen, Funktionsbausteinen, Funktionen und Programmen). Außerdem können Technologiepakete und Bibliotheken geladen werden.

Der Interfaceabschnitt hat folgende Syntax:

Syntax

```
INTERFACE
    // Interfaceanweisungen (optional)
END_INTERFACE
```

Ein individueller Bezeichner des Abschnitts kann nicht angegeben werden.

Zwischen den reservierten Wörtern INTERFACE und END_INTERFACE stehen optional die Interfaceanweisungen in folgender Reihenfolge:

1. Angabe des verwendeten Technologiepakets. Die Syntax lautet:
   `USEPACKAGE tp-name [AS namespace];`
   Näheres dazu siehe Funktionshandbuch *SIMOTION Basisfunktionen*.
2. Angabe der verwendeten Bibliotheken.
   Die Syntax lautet:
   `USELIB library-name-list [AS namespace];`
   Näheres dazu siehe "Datentypen, Funktionen und Funktionsbausteine aus Bibliotheken verwenden (Seite 289)".
3. Verweis auf andere Units, um deren exportierte Bestandteile zu verwenden.
   Die Syntax lautet
   `USES unit_name-list;`
   Näheres dazu siehe "USES-Anweisung in einer importierenden Unit (Seite 227)".
4. Deklarationen und Angaben zum Export
   - Datentypdefinitionen (Seite 222):
     In der gesamten ST-Quelle gültige und zu exportierende anwenderdefinierte Datentypen (UDT)
   - Variablendeklarationen (Seite 223):
     In der gesamten ST-Quelle gültige und zu exportierende Unit-Variablen und Unit-Konstanten.
     Zulässige Schlüsselwörter: siehe Abschnitt "Variablendeklaration" (Seite 223).
   - Angabe zu exportierender Programmorganisationseinheiten (POE).
     Die Syntax lautet:
     ```
     FUNCTION fc_name;
     FUNCTION_BLOCK fb_name;
     PROGRAM program_name;
     ```
     Bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben" werden sie auch als POE-Prototypen für die Vorwärtsdeklaration (Seite 318) interpretiert.
   - POE-Prototypen für die Vorwärtsdeklaration (Seite 318).
     Angabe der Prototypen für Programmorganisationseinheiten bei Vorwärtsdeklaration (nur wirksam bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben").
     Sie werden auch als zu exportierende POE interpretiert.

Alle im Interfaceabschnitt aufgeführten Technologiepakete, Bibliotheken, importierte Units, Datentypdeklarationen, Variablendeklarationen und Programmorganisationseinheiten werden exportiert. Näheres zum Export siehe "Interfaceabschnitt einer exportierenden Unit (Seite 226)".

Reihenfolge

Der Interfaceabschnitt ist der 1. Abschnitt einer ST-Quelle.

Hinweis

Optional kann dem Interfaceabschnitt die Unit-Anweisung vorangestellt werden, siehe "Bezeichnung der Unit (Seite 225)".

Die Reihenfolge der Interfaceanweisungen 1 bis 4 ist verbindlich.

Innerhalb der Nr. 4 ist die Reihenfolge beliebig. Die einzelnen Deklarationsblöcke für Datentypdefinitionen (einschließlich POE-Prototypen) und Variablendeklarationen können mehrfach auftreten.

Beachten Sie: Bezeichner müssen deklariert werden, bevor sie verwendet werden.

Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben": Bei der Instanzdeklaration eines Funktionsbausteins ist ausreichend, dass der Prototyp des Funktionsbausteins vorher deklariert ist.

Häufigkeit

Einmal pro ST-Quelle.

Pflichtteil

Ja

6.1.1.2 Implementationsabschnitt

Der Implementationsabschnitt enthält die ausführbaren Abschnitte und damit den Hauptteil der ST-Quelle.

Der Implementationsabschnitt hat folgende Syntax:

Syntax

```
IMPLEMENTATION
    // Implementationsanweisungen (optional)
END_IMPLEMENTATION
```

Ein individueller Bezeichner des Abschnitts kann nicht angegeben werden.

Zwischen den reservierten Wörtern IMPLEMENTATION und END_IMPLEMENTATION stehen optional die Implementationsanweisungen (der Hauptteil der ST-Quelldatei) in folgender Reihenfolge:

1. Verweis auf andere Units, um deren exportierte Bestandteile zu verwenden. Die Syntax lautet:
   `USES unit_name-list;`
   Näheres dazu siehe "USES-Anweisung in einer importierenden Unit (Seite 227)".
2. Deklarationen
   - Datentypdefinitionen (Seite 222):
     In der gesamten ST-Quelle gültige anwenderdefinierte Datentypen (UDT)
   - Variablendeklarationen (Seite 223):
     In der gesamten ST-Quelle gültige Unit-Variablen und -Konstanten.
     Zulässige Schlüsselwörter: siehe Abschnitt "Variablendeklaration" (Seite 223).
   - POE-Prototypen für die Vorwärtsdeklaration (Seite 318).
     Angabe der Prototypen für Programmorganisationseinheiten bei Vorwärtsdeklaration (nur wirksam bei aktivierter Compileroption (Seite 64) "Vowärtsdeklarationen erlauben").
3. Programmorganisationseinheiten (Seite 215).

Reihenfolge

Folgt immer dem Interfaceabschnitt.

Die Reihenfolge der o. a. Implementationsanweisungen ist verbindlich; innerhalb der Nr 2 und 3 ist die Reihenfolge beliebig:

Beachten Sie: Bezeichner müssen deklariert werden, bevor sie verwendet werden.

Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben": Bei der Instanzdeklaration eines Funktionsbausteins ist ausreichend, dass der Prototyp des Funktionsbausteins vorher deklariert ist.

Häufigkeit

Einmal pro ST-Quelle.

Pflichtteil

Ja

6.1.1.3 Programmorganisationseinheiten (POE)

Programmorganisationseinheiten (POE) sind die ausführbaren Quelldatei-Abschnitte:

- Funktionen (FC) (Seite 216)
- Funktionsbausteine (FB) (Seite 217)
- Programme (Seite 218)
- Expressions (Seite 219)

Für die Reihenfolge der POE in einer ST-Quelle gilt:

- Standardmäßig müssen in der Quelldatei die aufgerufenen POE vor den aufrufenden POE stehen, damit sie den letzteren bekannt sind.
- **Ausnahme:** Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben". Die Reihenfolge ist beliebig. Für die Angabe eines Prototyps zur Vorwärtsdeklaration (Seite 318) im Interfaceabschnitt (Seite 212) oder Implementationsabschnitt (Seite 214) gilt:
  - Zwingend erforderlich bei Funktionsbausteinen zur Instanzdeklaration
  - Optional beim Aufruf von Funktionen und Programmen

6.1.1.4 Funktionen (FC)

Funktionen (FC) gehören zu den Programmorganisationseinheiten (POE). Sie sind aus Programmen und Funktionsbausteinen aufrufbare und parametrierbare Quelldatei-Abschnitte mit temporären Daten. Alle internen Variablen verlieren nach Verlassen der Funktion ihren Wert; beim nächsten Aufruf werden sie neu initialisiert.

FC haben folgende Syntax:

Syntax

```
FUNCTION name : function_data_type
    // Deklarationsabschnitt
    // Anweisungsabschnitt
END_FUNCTION
```

name steht für den Bezeichner der Funktion, *function_data_type* für den Datentyp des Rückgabewertes.

Zulässige Schlüsselwörter für die Variablendeklaration im Deklarationsabschnitt (Seite 220): siehe Abschnitt "Variablendeklaration" (Seite 223).

Beachten Sie bei Funktionen mit *function_data_type* <> VOID: Im Anweisungsabschnitt (Seite 221) muss dem Funktionsbezeichner ein Ausdruck vom Datentyp *function_data_type* zugewiesen werden!

Reihenfolge

FC können Sie nur im Implementationsabschnitt (Seite 214) definieren.

- Standardmäßig müssen in der Quelldatei Funktionen vor der POE stehen, von der sie aufgerufen werden!
- **Ausnahme:** Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben". Die Reihenfolge ist beliebig. Die Angabe eines Prototyps zur Vorwärtsdeklaration (Seite 318) im Interfaceabschnitt (Seite 212) oder Implementationsabschnitt (Seite 214) ist optional möglich.

Der Deklarationsabschnitt (Seite 220) muss vor dem Anweisungsabschnitt (Seite 221) stehen.

Häufigkeit

Beliebig oft pro ST-Quelle.

Pflichtteil

Nein

Weitere Informationen zu Funktionen (FC)

Siehe Abschnitt "Erstellung und Aufruf von Funktionen und Funktionsbausteinen" (Seite 186).

6.1.1.5 Funktionsbausteine (FB)

Funktionsbausteine (FB) gehören zu den Programmorganisationseinheiten (POE). Sie sind aus Programmen aufrufbare und parametrierbare Quelldatei-Abschnitte mit statischen Daten (interne Variablen behalten ihren Wert zwischen mehreren Aufrufen). Da ein FB über ein Gedächtnis verfügt, kann auf seine Ausgangsparameter zu jeder Zeit an jeder beliebigen Stelle im Anwenderprogramm zugegriffen werden.

FB haben folgende Syntax:

Syntax

```
FUNCTION_BLOCK name
    // Deklarationsabschnitt
    // Anweisungsabschnitt
END_FUNCTION_BLOCK
```

name steht für den Bezeichner des Funktionsbausteins.

Zulässiges Schlüsselwörter für die Variablendeklaration im Deklarationsabschnitt: siehe Abschnitt "Variablendeklaration" (Seite 223).

Besonderheiten

Bevor Sie einen Funktionsbaustein (FB) aufrufen, müssen Sie eine Instanz deklarieren: Sie deklarieren eine Variable und geben als Datentyp den Bezeichner des Funktionsbausteins ein. Diese Instanzdeklaration führen Sie lokal (innerhalb VAR / END_VAR in den Deklarationsabschnitten eines Programms oder eines Funktionsbausteins) durch.

Sie können die Instanzdeklaration auch global (innerhalb VAR_GLOBAL / END_VAR im Interface- oder Implementationsabschnitt) durchführen. Standardmäßig ist dies nur mit Funktionsbausteinen möglich, die importierte Programmquellen und Bibliotheken zur Verfügung stellen, jedoch nicht mit Funktionsbausteinen, die in derselben ST-Quelle definiert werden.

Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben" können Sie globale Instanzen von Funktionsbausteinen deklarieren, die in derselben ST-Quelle definiert werden. Die Angabe eines Prototyps zur Vorwärtsdeklaration (Seite 318) im Interface- oder Implementationsabschnitt ist zwingend erforderlich.

In Funktionen können Sie keine Instanz eines FB deklarieren.

Reihenfolge

FB können Sie nur im Implementationsabschnitt (Seite 214) definieren.

- Standardmäßig müssen in der Quelldatei FB vor der POE stehen, in der eine Instanz als lokale Variable deklariert wird.
- **Ausnahme:** Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben". Die Reihenfolge ist beliebig. Die Angabe eines Prototyps zur Vorwärtsdeklaration (Seite 318) im Interfaceabschnitt (Seite 212) oder Implementationsabschnitt (Seite 214) ist zwingend erforderlich.

Der Deklarationsabschnitt (Seite 220) muss vor dem Anweisungsabschnitt (Seite 221) stehen.

Häufigkeit

Beliebig oft pro ST-Quelle.

Pflichtteil

Nein

Weitere Informationen zu Funktionbausteinen (FB)

Siehe Abschnitt "Erstellung und Aufruf von Funktionen und Funktionsbausteinen" (Seite 186).

6.1.1.6 Programme

Programme gehören zu den Programmorganisationseinheiten (POE). Sie werden auf dem Zielgerät gemäß ihrer Taskzuordnung aufgerufen (siehe *Ablaufsystem konfigurieren* im Funktionshandbuch *SIMOTION Basisfunktionen*) und können FC und FB aufrufen.

Programme haben folgende Syntax:

Syntax

```
PROGRAM name
    // Deklarationsabschnitt
    // Anweisungsabschnitt
END_PROGRAM
```

name steht für den Namen des Programms.

Zulässige Schlüsselwörter für die Variablendeklaration im Deklarationsabschnitt: siehe Abschnitt "Variablendeklaration" (Seite 223).

Reihenfolge

Programme können Sie nur im Implementationsabschnitt (Seite 214) definieren.

- Zweckmäßigerweise stehen Programme nach den Expressions, FC, FB. Damit kennt das Programm diese Quelldateiabschnitte und kann sie verwenden.
- **Ausnahme:** Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben". Die Reihenfolge ist beliebig. Bei Aufruf eines Programms innerhalb eines Programms ist die Angabe eines Prototyps zur Vorwärtsdeklaration (Seite 318) im Interfaceabschnitt (Seite 212) oder Implementationsabschnitt (Seite 214) optional möglich.

Der Deklarationsabschnitt (Seite 220) muss vor dem Anweisungsabschnitt (Seite 221) stehen.

Häufigkeit

Beliebig oft pro ST-Quelle.

Pflichtteil

Nein

Weitere Informationen zu Programmen

Siehe Abschnitt "Programme" (Seite 206).

6.1.1.7 Expressions

Expressions sind ein Spezialfall einer Funktionsvereinbarung mit dem festgelegten Datentyp BOOL des Rückgabewerts. Es wird der Ausdruck innerhalb der reservierten Wörter EXPRESSION <Expression-Bezeichner> … END_EXPRESSION ausgewertet, der dem Funktionsnamen zugewiesen wird.

Mit dem Konstrukt WAITFORCONDITION (Seite 176) können Sie in einer MotionTask direkt auf ein programmierbares Ereignis bzw. eine Bedingung warten. Der Befehl setzt die Task, aus der heraus er aufgerufen wird, so lange aus, bis die Bedingung (Expression) wahr wird.

Expressions haben folgende Syntax:

Syntax

```
EXPRESSION name
    // Deklarationsabschnitt
    // Anweisungsabschnitt
END_EXPRESSION
```

name steht für den Bezeichner der Expression.

Zulässige Schlüsselwörter für die Variablendeklaration im Deklarationsabschnitt: siehe Abschnitt "Variablendeklaration" (Seite 223).

Beachten Sie: Im Anweisungsabschnitt (Seite 221) muss dem Expression-Bezeichner ein Ausdruck vom Datentyp BOOL zugewiesen werden!

Reihenfolge

Expressions können ausschließlich im Implementationsabschnitt (Seite 214) einer ST-Quelle vereinbart werden.

- Standardmäßig stehen sie vor dem Programm, in dem sie aus einer WAITFORCONDITION-Kontrollstruktur aufgerufen werden.
- **Ausnahme:** Nur bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben". Die Reihenfolge ist beliebig. Die Angabe eines Prototyps zur Vorwärtsdeklaration (Seite 318) im Interfaceabschnitt (Seite 212) oder Implementationsabschnitt (Seite 214) ist nicht möglich.

Der Deklarationsabschnitt (Seite 220) muss vor dem Anweisungsabschnitt (Seite 221) stehen.

Häufigkeit

Beliebig oft pro ST-Quelle.

Pflichtteil

Nein

Weitere Informationen zu Expressions

Siehe Abschnitt "Expressions" (Seite 208).

Beispiele n Zusammenhang mit der WAITFORCONDITION-Anweisung: siehe Funktionshandbuch SIMOTION Basisfunktionen.

6.1.1.8 Deklarationsabschnitt

Der Deklarationsabschnitt einer Programmorganisationseinheit (POE) enthält in die Datentypdefinition und Variablendeklaration der POE.

Der Deklarationsabschnitt hat folgenden Aufbau:

Aufbau

```
// Datentypdefinition
// Variablendeklaration
```

Reihenfolge

Der Deklarationsabschnitt hat keine expliziten Schlüsselwörter am Anfang oder Ende. Er beginnt nach dem Schlüsselwort der jeweiligen Programmorganisationseinheit (POE) und endet mit der 1. ausführbaren Anweisung des Anweisungsteils.

Er enthält in beliebiger Reihenfolge:

- Datentypdefinitionen (Seite 222):
  Lokal in der POE gültige anwenderdefinierte Datentypen (UDT)
- Variablendeklarationen (Seite 223):
  Lokal in der POE gültige Variablen und Konstanten.
  Zulässige Schlüsselwörter in Abhängigkeit von der jeweiligen POE: siehe Abschnitt "Variablendeklaration" (Seite 223).

Beachten Sie: Bezeichner müssen deklariert werden, bevor sie verwendet werden.

Häufigkeit

Einmal pro POE

Pflichtteil

Nein

6.1.1.9 Anweisungsabschnitt

Der Anweisungsabschnitt einer POE besteht aus den einzelnen (ausführbaren) Anweisungen.

Der Anweisungsabschnitt hat folgenden Aufbau:

Aufbau

```
// Anweisungen
```

Reihenfolge

Der Anweisungsabschnitt hat keine expliziten Schlüsselwörter am Anfang oder Ende. Er beginnt nach dem Deklarationsabschnitt und endet mit dem Schlüsselwort der jeweiligen POE.

Häufigkeit

Einmal pro POE

Pflichtteil

Nein

Weitere Informationen zu Anweisungen

Siehe folgende Abschnitte:

- Wertzuweisungen und Ausdrücke (Seite 148)
- Kontrollanweisungen (Seite 167)
- Aufruf von Funktionen und Funktionsbausteinen (Seite 193)

6.1.1.10 Datentypdefinition

Bei der Datentypdefinition legen Sie anwenderdefinierte Datentypen (UDT) fest. Sie können diese für Variablendeklarationen verwenden. UDT können im Interfaceabschnitt, im Implementationsabschnitt sowie im Deklarationsabschnitt von FC, FB und Programmen definiert werden.

Die Datentypdefinition hat folgende Syntax:

Syntax

```
TYPE
    name : data_type_specification;
    // ...
END_TYPE
```

name steht für den Namen des individuellen Datentyps, den Sie bei den Variablendeklarationen verwenden.

data_type_specification steht für einen beliebigen Datentyp oder eine Struktur. Zwischen TYPE und END_TYPE können beliebig viele individuelle Datentypen stehen.

Reihenfolge

Sie können UDT definieren:

- im Interfaceabschnitt:
  Die UDT sind innerhalb der ST-Quelle bekannt und werden exportiert
  Sie können im Interface- und Implementationsabschnitt zur Deklaration von Unit-Variablen und in allen POE zur Deklaration lokaler Variablen verwendet werden.
  Außerdem können sie in allen Units verwendet werden, welche diese ST-Quelle importieren (in SIMOTION ST mit der USES-Anweisung).
- im Implementationsabschnitt:
  Die UDT sind innerhalb der ST-Quelle bekannt.
  Sie können im Implementationsabschnitt zur Deklaration von Unit-Variablen und in allen POE zur Deklaration lokaler Variablen verwendet werden.
- im Deklarationsabschnitt einer POE (FC, FB, Programm, Expression
  Die UDT sind nur lokal innerhalb der POE bekannt.
  Sie können nur innerhalb der POE zur Deklaration lokaler Variablen verwendet werden.

UDT müssen definiert sein, bevor sie in einer Variablendeklaration verwendet werden.

Häufigkeit

Der Deklarationsblock TYPE / END_TYPE darf mehrfach in einem Quelldatei-Abschnitt auftreten; in einem Deklarationsblock sind beliebig viele UDT möglich.

Pflichtteil

Nein

Weitere Informationen zu anwenderdefinierten Datentypen (UDT)

Siehe Abschnitt "Anwenderdefinierte Datentypen" (Seite 120).

6.1.1.11 Variablendeklaration

Ein Deklarationsabschnitt enthält Variablendeklarationen und kann in FC, FB und Programmen (POE) sowie im Interface- und Implementationsabschnitt enthalten sein.

Die Variablendeklaration hat folgende Syntax:

Syntax

```
variable_type
    name_list : data_type;
    // ...
END_VAR
```

variable_type steht für das Schlüsselwort der Variablenart, die deklariert wird. Die zulässigen Schlüsselwörter sind abhängig vom Quelldatei-Abschnitt.

- Im Interfaceabschnitt oder Implementationsabschnitt einer ST-Quelle:
  `VAR_GLOBAL`: Nicht remanente Unit-Variable
  `VAR_GLOBAL CONSTANT`: Unit-Konstante
  `VAR_GLOBAL RETAIN`: Remanente Unit-Variable
- Im Deklarationsabschnitt einer Funktion:
  `VAR`: Lokale Variable
  `VAR CONSTANT`: Lokale Konstante
  `VAR_INPUT`: Eingangsparameter
  `VAR_IN_OUT`: Durchgangsparameter
- Im Deklarationsabschnitt eines Funktionsbausteins:
  `VAR`: Lokale Variable
  `VAR CONSTANT`: Lokale Konstante
  `VAR_TEMP`: Temporäre Variable
  `VAR_INPUT`: Eingangsparameter
  `VAR_OUTPUT`: Ausgangsparameter
  `VAR_IN_OUT`: Durchgangsparameter

- Im Deklarationsabschnitt eines Programms:
  `VAR`: Lokale Variable
  `VAR CONSTANT`: Lokale Konstante
  `VAR_TEMP`: Temporäre Variable
- Im Deklarationsabschnitt einer Expression:
  `VAR`: Lokale Variable
  `VAR CONSTANT`: Lokale Konstante
  `VAR_INPUT`: Eingangsparameter (ab Version 4.1 des SIMOTION Kernels)
  `VAR_IN_OUT`: Durchgangsparameter (ab Version 4.1 des SIMOTION Kernels)

name_list ist die Liste der Bezeichner vom Datentyp *data_type*, die deklariert werden sollen.

Reihenfolge

Die Variablendeklaration erfolgt:

- Im Interfaceabschnitt der ST-Quelle:
  Zulässige Schlüsselwörter: siehe oben unter *Syntax*.
  Die Unit-Variablen sind innerhalb der ST-Quelle bekannt und werden exportiert
  Sie können in allen POE der ST-Quelle verwendet werden.
  Außerdem können sie in allen Units verwendet werden, welche diese ST-Quelle importieren (in SIMOTION ST mit der USES-Anweisung).
- Im Implementationsabschnitt der ST-Quelle:
  Zulässige Schlüsselwörter: siehe oben unter *Syntax*.
  Die Unit-Variablen sind innerhalb der ST-Quelle bekannt.
  Sie können in allen POE der ST-Quelle verwendet werden.
- Im Deklarationsabschnitt einer POE (FC, FB, Programm, Expression)
  Zulässige Schlüsselwörter in Abhängigkeit von der Art der POE: siehe oben unter *Syntax*.
  Die Variablen sind nur lokal innerhalb der POE bekannt.
  Sie können nur innerhalb der POE zur Deklaration lokaler Variablen verwendet werden.
  Ausnahmen:
  - Auf die Ausgangsparameter eines Funktionsbausteins können Sie auch außerhalb des FB zugreifen.
  - Auf die Eingangsparameter eines Funktionsbausteins können Sie von außerhalb des FB zugreifen, wenn die Compileroption "Spracherweiterungen zulassen" aktiviert ist. Siehe Globale Einstellungen des Compilers (Seite 64) und Lokale Einstellungen des Compilers (Seite 67).

Variablen müssen definiert sein, bevor sie verwendet werden.

Häufigkeit

Wie häufig der Deklarationsblock *variable_type* / END_VAR eines bestimmten Variablentyps auftreten darf, ist abhängig vom jeweiligen Quelldatei-Abschnitt:

- Im Interface und Implementationsabschnitt der ST-Quelle:
  Die Deklarationsblöcke dürfen mehrfach auftreten.
- Im Deklarationsabschnitt einer POE (FC, FB, Programm, Expression):
  Jeder Deklarationsblock (außer VAR CONSTANT / END_VAR) darf nur einmal im Deklarationsabschnitt auftreten.

Zulässige Deklarationsblöcke und Schlüsselwörter in Abhängigkeit vom jeweiligen Quelldatei-Abschnitt: siehe oben unter *Syntax*.

Innerhalb eines Deklarationsblocks sind beliebig viele Variablendeklarationen möglich.

Pflichtteil

Nein

Weitere Informationen zur Variablendeklaration

Siehe Abschnitt "Variablendeklaration" (Seite 139).

6.1.2 Import und Export zwischen ST-Quellen

ST sieht das Unit-Konzept vor, mit dem Sie auf globale Variablen, Datentypen, Funktionen (FC), Funktionsbausteine (FB) und Programme anderer Quelldateien zugreifen können. Somit können z. B. wieder verwendbare Unterprogramme zusammengefasst und zur Verfügung gestellt werden.

6.1.2.1 Bezeichnung der Unit

Als Unit wird im Folgenden eine Programmquelle (z. B. ST-Quelle, MCC-Quelle) bezeichnet. Als Bezeichnung wird der Name der Programmquelle, wie in SIMOTION SCOUT definiert, übernommen.

Optional können Sie bei einer ST-Quelle die Unit-Anweisung als erste Anweisung (vor dem Interfaceabschnitt) setzen. Die Syntax lautet:

```
UNIT name;
```

name entspricht dem Namen der ST-Quelle, wie in SIMOTION SCOUT definiert, siehe ST-Quelle einfügen (Seite 29) oder Eigenschaften einer ST-Quelle ändern (Seite 31).

Die Unit-Anweisung wird ignoriert, wenn der dort angegebene Name vom Namen der ST-Quelle abweicht.

6.1.2.2 Interfaceabschnitt einer exportierenden Unit

Im Interfaceabschnitt einer exportierenden Unit können Sie folgende Konstrukte eintragen. Die Syntax der Konstrukte ist hier nur angedeutet, Einzelheiten siehe "Interfaceabschnitt (Seite 212)".

- Die zu exportierenden Datentypdeklarationen
  TYPE
  Anwenderdefinierte Datentypen mit ihrer vollständigen Deklaration.
- Die zu exportierenden Variablendeklarationen
  VAR_GLOBAL, VAR_GLOBAL RETAIN oder VAR_GLOBAL CONSTANT
  Nicht remanente und remanente Unit-Variablen sowie Unit-Konstanten mit ihrer vollständigen Deklaration.
- Die zu exportierenden POE (Funktionen, Funktionsbausteine und Programme)
  Geben Sie jeweils die zu exportierende POE (Funktion, Funktionsbaustein oder Programm) mit dem entsprechenden Schlüsselwort an (optional innerhalb des Konstrukts TYPE / END_TYPE). Schließen Sie jeden Eintrag mit einem Strichpunkt ab.
  - FUNCTION_BLOCK *fb_name*;
  - FUNCTION *fc_name*;
  - PROGRAM *program_name*;

  Bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben" werden sie auch als POE-Prototypen für die Vorwärtsdeklaration (Seite 318) interpretiert.

Die Reihenfolge der Angaben ist beliebig; die POE selbst programmieren Sie im Implementationsabschnitt (Seite 214) der ST-Quelle.

Hinweis

Folgende weiteren Angaben sind im Interfaceabschnitt möglich, Sie werden **vor** den exportierten Datentypen, Variablen und POE aufgeführt:

1. Angabe des verwendeten Technologiepakets (USEPACKAGE …).
2. Angabe der verwendeten Bibliotheken (USELIB …).
3. Verweis auf andere Units, um deren exportierte Bestandteile zu verwenden (USES …).

Diese importierten Technologiepakete, Bibliotheken und Units werden auch exportiert. Zur Vererbung siehe "USES-Anweisung in einer importierenden Unit (Seite 227)".

Achten Sie auf die Reihenfolge der Angaben im Interfaceabschnitt einer Unit (ST-Quelle), siehe "Interfaceabschnitt (Seite 212)". Ansonsten ist keine fehlerfreie Übersetzung der ST-Quelle möglich.

Programme, die einer Task im Ablaufsystem zugeordnet werden sollen, müssen im Interfaceabschnitt aufgeführt werden (siehe *Ablaufsystem konfigurieren* im Funktionshandbuch *SIMOTION Basisfunktionen*). Der Compiler gibt eine Warnmeldung aus, wenn Programme nicht im Interfaceabschnitt einer ST-Quelle exportiert werden.

Funktionen, Funktionsbausteine und Programme, die nur innerhalb der ST-Quelle verwendet werden, sollten nicht im Interfaceabschnitt aufgeführt werden.

6.1.2.3 Beispiel für exportierende Unit

Nachfolgend sehen Sie ein Beispiel für eine exportierende Unit (*myUnit_A*). Diese wird von der *myUnit_B* (siehe Beispiel für eine importierende Unit (Seite 229)) importiert.

Tabelle 6-1 Beispiel für exportierende Unit

```
UNIT myUnit_A;    // optional, Name der ST-Quelle

INTERFACE
    // ... hier auch Anweisung USES möglich
    TYPE        // Deklaration zu exportierender Datentypen
        color : (RED, GREEN, BLUE);
    END_TYPE
    VAR_GLOBAL
        cycle : INT := 1;  // Deklaration zu exportierender
                           // Unit-Variablen
    END_VAR
    FUNCTION myFC;         // Exportanweisung einer FC
    FUNCTION_BLOCK myFB;   // Exportanweisung eines FB
    PROGRAM myProgram_A;   // Exportanweisung eines Programms
                           // (zur Anbindung an Ablaufsystem)
END_INTERFACE

IMPLEMENTATION
    Function myFC : LREAL   // Funktion ausformuliert
        ;     // ... (Anweisungen)
    END_FUNCTION

    Function_BLOCK myFB      // Funktionsbaustein ausformuliert
        ;     // ... (Anweisungen)
    END_FUNCTION_BLOCK

    PROGRAM myProgram_A     // Programm ausformuliert
        ;     // ... (Anweisungen)
    END_PROGRAM
END_IMPLEMENTATION
```

6.1.2.4 USES-Anweisung in einer importierenden Unit

In einer importierenden Unit tragen Sie folgende Anweisung im Interface- oder Implementationsabschnitt ein:

```
USES unit_name-list
```

unit_name-list ist eine durch Kommata getrennte Aufzählung der Units, die importiert werden sollen.

Beispiel:

```
USES unit_1, unit_2, unit_3;
```

Sie habe damit Zugriff auf folgende Elemente, die im Interfaceabschnitt der importierten Units (z. B. ST-Quelle, MCC-Quelle) angegeben oder deklariert sind:

- Anwenderdefinierte Datentypen (UDT)
- Unit-Variablen und Unit-Konstanten
- Programme, Funktionen und Funktionsbausteine
- Importierte Technologiepakete, Bibliotheken und Units

Sie können die importierten Elemente so verwenden, als ob sie in der aktuellen Unit vorhanden wären.

Hinweis

Das Schlüsselwort USES darf nur je einmal im Interface- oder Implementationsabschnitt einer Unit auftreten. Mehrere zu importierende Units führen Sie als durch Kommata getrennte Liste hinter dem Schlüsselwort USES an.

Die USES-Anweisung kann im Interfaceabschnitt oder im Implementationsabschnitt einer Unit stehen. Dies hat weit reichende Auswirkungen:

Tabelle 6-2 Auswirkungen, in welchem Abschnitt der ST-Quelle die USES-Anweisung steht.

Auswirkung	USES-Anweisung im Interfaceabschnitt	USES-Anweisung im Implementationsabschnitt
Vererbung	Die aktuelle Unit exportiert die importierte Unit weiter; die importierte Unit wird an alle weiteren Units vererbt, die auf die aktuelle Unit zugreifen. Beispiel: 1. Unit B importiert Unit A im **Interface**abschnitt. 2. Unit C importiert wiederum Unit B. 3. Dann importiert Unit C automatisch auch Unit A. A → B → C ⇒ A → C Wegen der Vererbung muss in Unit C die Unit A nicht explizit importiert werden.	Die Vererbung ist unterbrochen. Beispiel: 1. Unit B importiert Unit A im **Implementation**sabschnitt. 2. Unit C importiert wiederum Unit B. 3. Dann hat Unit C keinen automatischen Zugriff auf Unit A. Unit C muss Unit A explizit importieren, wenn sie auf Unit A zugreifen will.
Variablendeklaration	Die Deklaration einer Unit-Variablen eines importierten Datentyps ist möglich in: • Interfaceabschnitt • Implementationsabschnitt	Die Deklaration einer Unit-Variablen eines importierten Datentyps ist nur im Implementationsabschnitt möglich.

Hinweis

Tipps zur Verwendung von Unit-Variablen erhalten Sie im Funktionshandbuch SIMOTION Basisfunktionen.

6.1.2.5 Beispiel für eine importierende Unit

Nachfolgend sehen Sie ein Beispiel für eine importierende Unit (*myUnit_B*). Diese importiert die Unit *myUnit_A* aus der Beispiel für exportierende Unit (Seite 227).

Tabelle 6-3 Beispiel für importierende Unit

```
UNIT myUnit_B;        // optional, Name der ST-Quelle
INTERFACE
    // ... ggf. USES-Anweisung
    PROGRAM myProgram_B;
    // Angabe zu exportierender Programme, FB, FC,
    // Datentypen und Unit-Variablen
END_INTERFACE

IMPLEMENTATION
    USES myUnit_A;    // Angabe der importierten Unit

    VAR_GLOBAL
        myInstance : myFB;      // Deklaration einer Instanz
                                // des importierten FB
        mycolor : color;        // Deklaration einer Variablen
                                // des importierten Datentyps
    END_VAR

    PROGRAM myProgram_B
        mycolor := GREEN;       // Wertzuweisung an Variable des
                                // importierten Datentyp
        cycle := cycle + 1;     // Wertzuweisung an
                                // importierte Variable
    END_PROGRAM
END_IMPLEMENTATION
```

6.2 Variablen in SIMOTION

Hier erhalten Sie einen Überblick über die Variablen, die in ST zur Verfügung stehen.

6.2.1 Variablenmodell

Die folgenden Tabellen zeigen alle Variablentypen, die Ihnen bei der Programmierung zur Verfügung stehen.

- Systemvariablen des SIMOTION Geräts und der Technologieobjekte
- Globale Anwendervariablen (I/O-Variablen, Geräteglobale Variablen, Unit-Variablen)
- Lokale Anwendervariablen (Variable innerhalb eines Programms, einer Funktion oder eines Funktionsbaustein)

Systemvariablen

Variablentyp	Bedeutung
Systemvariablen des SIMOTION Geräts	Jedes SIMOTION Gerät und jedes Technologieobjekt hat spezifische Systemvariablen. Auf diese können Sie zugreifen: • innerhalb des SIMOTION Geräts von allen Programmen • von HMI-Geräten Systemvariablen können Sie im Symbol-Browser beobachten.
Systemvariablen der Technologieobjekte	

Globale Anwendervariablen

Variablentyp	Bedeutung
I/O-Variablen	Sie können den I/O-Adressen des SIMOTION Geräts oder der Peripherie symbolische Variablennamen zuordnen. Dies ermöglicht Ihnen folgende Direktzugriffe und Prozessabbildzugriffe auf die Peripherie: • innerhalb des SIMOTION Geräts von allen Programmen • von HMI-Geräten Diese Variablen legen Sie im Symbol-Browser an, nachdem Sie im Projektnavigator das Element I/O markiert haben. I/O-Variablen können Sie im Symbol-Browser beobachten.
Geräteglobale Variablen	Anwenderdefinierte Variablen, auf die alle Programme des SIMOTION Geräts sowie HMI-Geräte zugreifen können. Diese Variablen legen Sie im Symbol-Browser an, nachdem Sie im Projektnavigator das Element GERÄTEGLOBALE VARIABLEN markiert haben. Geräteglobale Variablen können Sie als remanent deklarieren. Sie bleiben dann auch nach Ausschalten des SIMOTION Geräts gespeichert. Geräteglobale Variablen können Sie im Symbol-Browser beobachten.
Unit-Variablen	Anwenderdefinierte Variablen, auf die alle Programme, Funktionsbausteine und Funktionen innerhalb einer Unit (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle) zugreifen können. Diese Variablen deklarieren Sie in der Unit: • im Interfaceabschnitt: Diese Variablen werden exportiert und können in andere Units (z. B. ST-Quellen, MCC-Quellen, KOP/FUP-Quellen) importiert werden Außerdem stehen sie standardmäßig auf HMI-Geräten zur Verfügung. • im Implementationsabschnitt: Auf diese Variablen können Sie nur innerhalb der jeweiligen Unit zugreifen. Unit-Variablen können Sie als remanent deklarieren. Sie bleiben dann auch nach Ausschalten des SIMOTION Geräts gespeichert. Unit-Variablen können Sie im Symbol-Browser beobachten.

Lokale Anwendervariablen

Variablentyp	Bedeutung
	Anwenderdefinierte Variablen, auf die nur innerhalb des Programms (bzw. Funktion, Funktionsbaustein) zugegriffen werden kann, in dem sie definiert wurden.
Variable eines Programms (Programmvariable)	Variable wird in einem Programm deklariert. Nur innerhalb dieses Programms kann auf sie zugegriffen werden. Es wird zwischen statischen und temporären Variablen unterschieden: • Statische Variablen werden gemäß dem Speicherbereich initialisiert, in dem sie abgelegt werden. Diesen Speicherbereich legen durch eine Compileroption fest. Standardmäßig werden die statischen Variablen in Abhängigkeit der Task initialisiert, der das Programm zugeordnet wird, (siehe Funktionshandbuch *SIMOTION Basisfunktionen*). Statische Variablen können Sie im Symbol-Browser beobachten. • Temporäre Variablen werden bei jedem Aufruf des Programms in einer Task initialisiert. Temporäre Variablen können nicht im Symbol-Browser beobachtet werden.

Variablentyp	Bedeutung
Variable einer Funktion (FC-Variable)	Variable wird in einer Funktion (FC) deklariert. Nur innerhalb dieser FC kann auf sie zugegriffen werden. FC-Variablen sind temporär, sie werden bei jedem Aufruf der FC initialisiert. Sie können nicht im Symbol-Browser beobachtet werden.
Variable eines Funktionsbausteins (FB-Variable)	Variable wird in einem Funktionsbaustein (FB) deklariert. Nur innerhalb dieses FB kann auf sie zugegriffen werden. Es wird zwischen statischen und temporären Variablen unterschieden: • Statische Variablen behalten beim Beenden des FB ihren Wert. Sie werden nur initialisiert, wenn die Instanz des FB initialisiert wird, Dies ist abhängig vom Variablentyp, mit der die Instanz des FB deklariert wurde. Statische Variablen können Sie im Symbol-Browser beobachten. • Temporäre Variablen verlieren beim Beenden des FB ihren Wert. Beim nächsten Aufruf des FB werden sie neu initialisiert. Temporäre Variablen können nicht im Symbol-Browser beobachtet werden.

Weitere Informationen finden Sie in folgenden Quellen:

- In den entsprechenden Listenhandbüchern finden Sie die komprimierte Information zu allen Systemvariablen der SIMOTION Technologiepakete und SIMOTION Geräte.
- In den Funktionshandbüchern *SIMOTION Motion Control Technologieobjekte* finden Sie Informationen zur Verwendung der Systemvariablen der Technologieobjekte.
- Im Funktionshandbuch *SIMOTION Basisfunktionen* finden Sie Informationen, wie Sie auf Systemvariablen und Konfigurationsdaten zugreifen können.
- In der vorliegenden Dokumentation finden Sie Informationen
  - zum Zugriff auf Peripherieadressen mit I/O-Variablen (siehe Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263)),
  - zum Prozessabbildzugriff (siehe Zugriffe auf festes Prozessabbild der BackgroundTask (Seite 273)),
  - zum Anlegen und zur Verwendung von geräteglobalen Variablen (siehe Verwendung von geräteglobalen Variablen (Seite 239)),
  - zur Verwendung von Unit-Variablen und Lokale Variablen (statische und temporäre Variablen).

Hinweis

Beachten Sie, dass der Download der ST-Quelle ins Zielsystem und ausführende Tasks die Initialisierung von Variablen und damit deren Inhalt beeinflussen, siehe Zeitpunkt der Variableninitialisierung (Seite 246).

6.2.1.1 Unit-Variablen

Unit-Variablen sind in der gesamten ST-Quelle gültig, d. h. Sie können in jedem Quelldatei-Abschnitt auf diese Variablen zugreifen.

Unit-Variablen werden im Interface- und/oder Implementationsabschnitt einer ST-Quelle deklariert; der Ort der Deklaration bestimmt die Gültigkeit der Unit-Variablen:

- Wenn Sie die Unit-Variablen im Interfaceabschnitt deklarieren, erzeugen Sie Variablen, die auch in anderen Programmquellen (z. B. ST-Quellen, MCC-Quellen) verwendet werden können. Mehr zum Thema Import und Export zwischen Programmquellen finden Sie in Import und Export zwischen ST-Quellen (Seite 225).
  Standardmäßig stehen diese Unit-Variablen auch auf HMI-Geräten zur Verfügung. Die Gesamtgröße der Unit-Variablen, die an HMI-Geräte exportiert werden kann, ist auf 64 kByte pro Unit beschränkt.
- Wenn Sie die Unit-Variablen im Implementationsabschnitt deklarieren, erzeugen Sie Variablen, die von allen Programmorganisationseinheiten (POE) der aktuellen Quelle verwendet werden können.

Die Voreinstellung für den HMI-Export der Unit-Variablen können Sie mit einem Pragma innerhalb jedes Deklarationsblocks ändern, siehe Variablen und HMI-Geräte (Seite 256) und Compiler mit Attributen steuern (Seite 312).

Unit-Variablen können Sie mit unterschiedlichem Verhalten, z. B. bei Spannungsausfall definieren:

- Nicht remanente Unit-Variablen (Schlüsselwort VAR_GLOBAL): ihr Wert geht bei einem Spannungsausfall verloren.
- Remanente Unit-Variablen (Schlüsselwort VAR_GLOBAL RETAIN): ihr Wert bleibt bei einem Spannungsausfall erhalten.
- Unit-Konstanten (Schlüsselwort VAR_GLOBAL CONSTANT): ihr Wert bleibt unveränderlich fest (siehe Konstanten (Seite 146)).

Tipps zur effizienten Verwendung von Unit-Variablen erhalten Sie im Funktionshandbuch *SIMOTION Basisfunktionen*.

6.2.1.2 Nicht remanente Unit-Variablen

Nicht remanente Unit-Variablen verlieren bei einem Spannungsausfall ihren Wert.

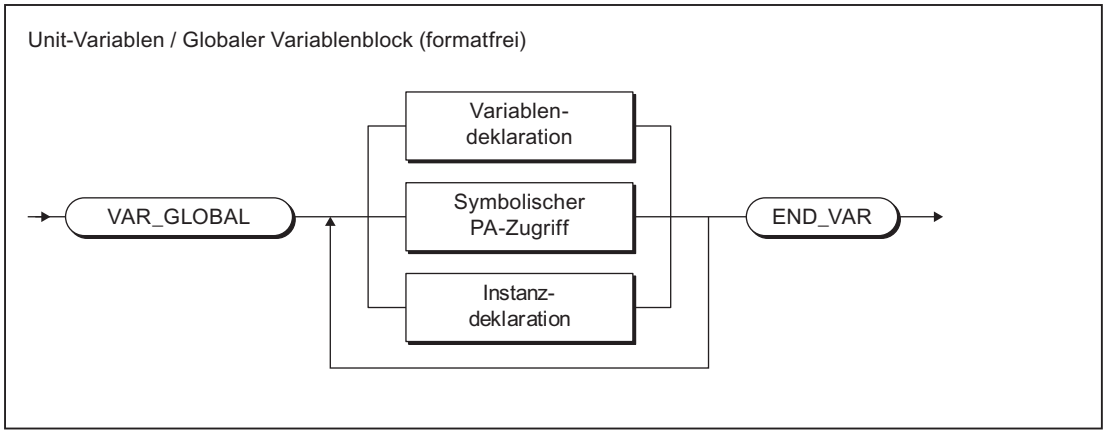


Bild 6-1 Syntax: Unit-Variablen

Dieser Deklarationsblock darf innerhalb eines Interface- oder Implementationsabschnitts mehrfach auftreten. Dabei geben Sie bei der Variablendeklaration Variablenname und Datentyp an (siehe Übersicht aller Variablendeklarationen (Seite 141) und Initialisierung von Variablen oder Datentypen (Seite 142)).

Zur Reichweite der Deklaration und den HMI-Export siehe Unit-Variablen (Seite 232).

Hinweis

Zur Initialisierung der nicht remanenten Unit-Variablen:

- Siehe Initialisierung nicht remanenter globaler Variablen (Seite 248).
- Das Verhalten beim Download ist einstellbar (Menü **Extras > Einstellungen**, Register **Download**, Checkbox **Initialisierung der nicht remanenten Programmdaten und der nicht remanenten geräteglobalen Variablen**.
- Die Art der Versionskennung und damit das Initialisierungsverhalten beim Download ist abhängig von der Version des SIMOTION Kernels. Siehe hierzu Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).

Tabelle 6-4 Beispiele für nicht remanente Unit-Variablen

```
INTERFACE
    VAR_GLOBAL    // Diese Variablen können exportiert werden.
        rotation1        : INT;
        field1           : ARRAY [1..10] OF REAL;
        flag1            : BOOL;
        motor1           : motor;    // Instanzdeklaration
    END_VAR
END_INTERFACE

IMPLEMENTATION
    VAR_GLOBAL    // Diese Variablen können nicht exportiert
                  // werden.
        rotation2        : INT;
        field2           : ARRAY [1..10] OF REAL;
        flag2            : BOOL;
        motor2           : motor;    // Instanzdeklaration
    END_VAR
END_IMPLEMENTATION
```

6.2.1.3 Remanente Unit-Variablen

Remanente Unit-Variablen ermöglichen das dauerhafte Speichern von Variablenwerten auch über einen Spannungsausfall hinweg.

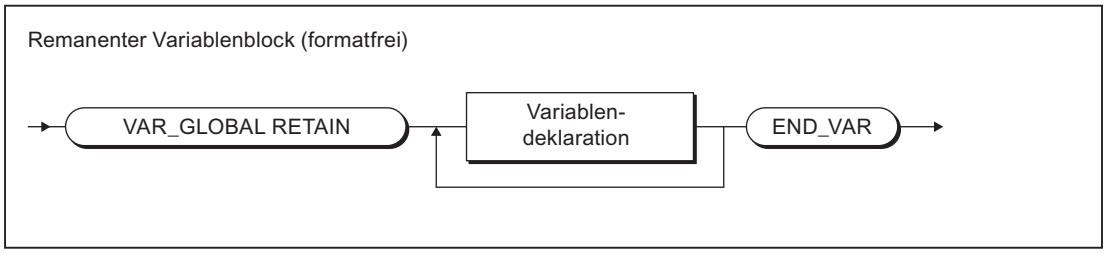


Bild 6-2 Syntax: Remanenter Variablenblock

Dieser Deklarationsblock darf innerhalb eines Interface- oder Implementationsabschnitts mehrfach auftreten. Dabei geben Sie bei der Variablendeklaration Variablenname und Datentyp an (siehe Übersicht aller Variablendeklarationen (Seite 141) und Initialisierung von Variablen oder Datentypen (Seite 142)).

Zur Reichweite der Deklaration und den HMI-Export siehe Unit-Variablen (Seite 232).

Hinweis

- Zur Initialisierung der remanenten Unit-Variablen:
  - Siehe Initialisierung remanenter globaler Variablen (Seite 247).
  - Das Verhalten beim Download ist einstellbar (Menü **Extras > Einstellungen**, Register **Download**, Checkbox **Initialisierung der remanenten Programmdaten und der remanenten geräteglobalen Variablen**.
  - Die Art der Versionskennung und damit das Initialisierungsverhalten beim Download ist abhängig von der Version des SIMOTION Kernels. Siehe hierzu Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).
- Der Speicherplatz für remanente Variablen ist geräteabhängig (siehe Mengengerüst im Projektierungshandbuch SIMOTION SCOUT).
  Sie nutzen den begrenzten Speicherplatz effektiv, wenn Sie ihn in einer einzigen ST-Quelldatei beanspruchen und die Variablen nach absteigender Größe des Datentyps sortieren!
- Überprüfen Sie im SIMOTION SCOUT den Füllstand des remanenten Speichers.
  Rufen Sie im Online-Modus die **Gerätediagnose** des zu überprüfenden SIMOTION Geräts auf (siehe Online-Hilfe). Im Register **Systemauslastung**. sehen Sie unter **Remanente Daten** den zur Verfügung stehenden Speicherplatz.

Tabelle 6-5 Beispiele für remanente Variablen

```
VAR_GLOBAL RETAIN
    Messfeld  : ARRAY[1..10] OF REAL;
    Durchlauf : INT;
    Schalter  : BOOL;
END_VAR
```

6.2.1.4 Lokale Variablen (statische und temporäre Variablen)

Lokale Variablen sind nur in dem Quelldatei-Abschnitt (z. B. Programm, FC oder FB) gültig, in dem sie deklariert wurden. Dabei unterscheidet man:

- Statische Variablen (Seite 237):
  Sie behalten ihren Wert über alle Durchläufe des Quelldatei-Abschnitts hinweg (Bausteingedächtnis).
- Temporäre Variablen (Seite 238):
  Sie werden bei einem erneuten Aufruf des Quelldatei-Abschnitt initialisiert.

Siehe auch: Initialisierung lokaler Variablen (Seite 250).

Hinweis

Auf lokale Variablen kann außerhalb des Quelldatei-Abschnitts, in dem sie deklariert wurden, nicht zugegriffen werden.

Die folgende Tabelle gibt geben einen Überblick über die Deklaration statischer und temporärer Variablen. Sie zeigt, in welchem Quelldatei-Abschnitt und mit welchen Schlüsselwörtern diese Variablen deklariert werden können.

Tabelle 6-6 Schlüsselwörter zur Deklaration statischer und temporärer Variablen in Abhängigkeit vom Quelldatei-Abschnitt.

Quelldatei-Abschnitt	Schlüsselwörter zur Deklaration	
	Statische Variablen	Temporäre Variablen
Funktion	–	VAR / END_VAR oder VAR_INPUT / END_VAR oder VAR_IN_OUT / END_VAR[2]
Expression	–	VAR/ END_VAR oder VAR_INPUT / END_VAR oder VAR_IN_OUT / END_VAR[2]
Funktionsbaustein	VAR / END_VAR[1] oder VAR_INPUT / END_VAR[1] oder VAR_OUTPUT / END_VAR[1]	VAR_TEMP / END_VAR oder VAR_IN_OUT / END_VAR[2]
Programm	VAR / END_VAR[3]	VAR_TEMP / END_VAR

[1] Die Initialisierung der Variablen ist abhängig von der Initialisierung der deklarierten Instanz. Siehe Initialisierung von Instanzen von Funktionsbausteinen (FB) (Seite 253).

[2] Die Referenz (Zeiger) zur übergebenen Variable ist temporär

[3] Die Initialisierung der Variablen ist abhängig vom Speicherbereich, in dem sie abgelegt werden. Siehe Initialisierung statischer Variablen von Programmen (Seite 251).

Hinweis

Beachten Sie, dass der Download der ST-Quelle ins Zielsystem und ausführende Tasks die Initialisierung von Variablen und damit deren Inhalt beeinflussen, siehe Zeitpunkt der Variableninitialisierung (Seite 246).

Tabelle 6-7 Beispiele für statische und temporäre Variablen

```
IMPLEMENTATION
    FUNCTION testFkt
        VAR            // Deklaration temporärer Variablen
            flag : BOOL;
        END_VAR
    END_FUNCTION
    FUNCTION_BLOCK testFbst;
        VAR            // Deklaration statischer Variablen
            rotation1 : INT;
        END_VAR

        VAR_TEMP       // Deklaration temporärer Variablen
            help1, help2 : REAL;
        END_VAR
    END_FUNCTION_BLOCK
    PROGRAM testPrg;
        VAR           // Deklaration statischer Variablen
            rotation2 : INT;
        END_VAR

        VAR_TEMP      // Deklaration temporärer Variablen
            help1, help2 : REAL;
        END_VAR
    END_PROGRAM
END_IMPLEMENTATION
```

6.2.1.5 Statische Variablen

Statische Variablen behalten nach Verlassen des Quelldatei-Abschnitts den zuletzt gespeicherten Wert. Beim nächsten Aufruf wird dieser Wert weiter verwendet.

Statische Variablen gibt es in folgenden Quelldatei-Abschnitten:

- Programme
- Funktionsbausteine

Sie werden im statischen Variablenblock deklariert.

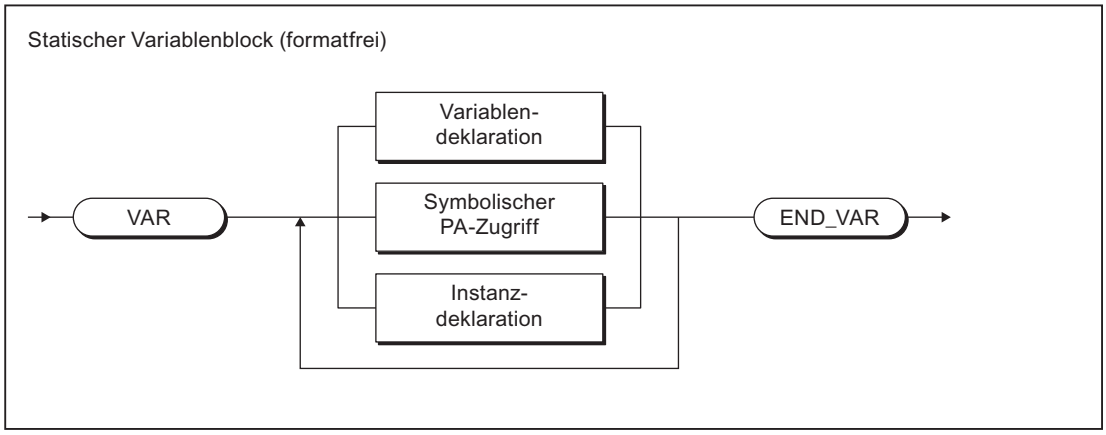


Bild 6-3 Syntax: Statischer Variablenblock

Im statischen Variablenblock können Sie laut Syntax aus dem Bild:

- Variablen deklarieren (Namen und Datentyp), optional mit Initialisierung
- Symbolische Zugriffe auf das Prozessabbild der BackgroundTask deklarieren
- Instanzen der Funktionsbausteine deklarieren

Zur Initialisierung der statischen Variablen:

- In Programmen: abhängig vom Ablaufverhalten der Task, der das Programm zugeordnet ist (siehe Funktionshandbuch *SIMOTION Basisfunktionen*).
  Siehe auch Initialisierung statischer Variablen von Programmen (Seite 251).
- In Funktionsbausteinen: abhängig von der Initialisierung der deklarierten Instanz.
  Siehe auch Initialisierung von Instanzen von Funktionsbausteinen (FB) (Seite 253).

6.2.1.6 Temporäre Variablen

Temporäre Variablen werden bei jedem Aufruf des Quelldatei-Abschnitts initialisiert. Ihr Wert bleibt nur während eines Ablaufs des Quelldatei-Abschnitts erhalten.

Temporäre Variablen gibt es in folgenden Quelldatei-Abschnitten:

- Programme
- Funktionsbausteine
- Funktionen
- Expression

In Funktionsbausteinen und Programmen deklarieren Sie temporäre Variablen im temporären Variablenblock FB und Programm (siehe nachstehendes Bild):

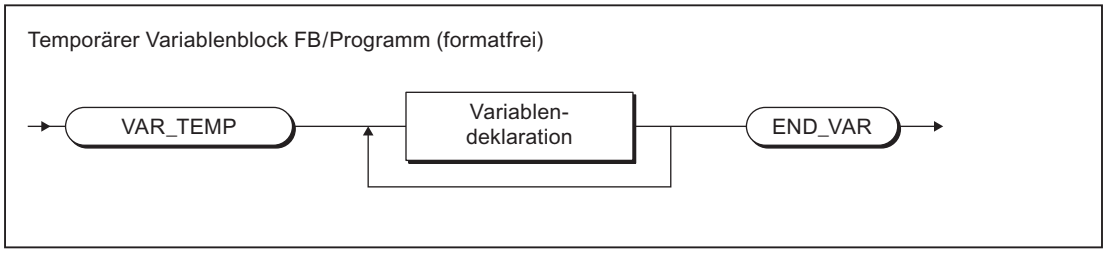


Bild 6-4 Syntax: Temporärer Variablenblock im FB oder Programm

In Funktionen und Expressions deklarieren Sie temporäre Variablen im temporären Variablenblock FC (siehe nachstehendes Bild):

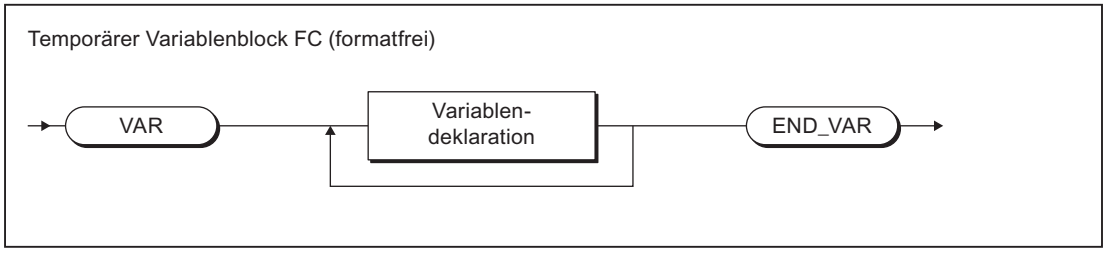


Bild 6-5 Syntax: Temporärer Variablenblock in einer FC

6.2.2 Verwendung von geräteglobalen Variablen

Geräteglobale Variablen sind anwenderdefinierte Variablen, auf die Sie aus allen Programmquellen (z. B. ST-Quellen, MCC-Quellen) eines SIMOTION Gerätes zugreifen können.

Geräteglobale Variablen erstellen Sie im Register Symbolbrowser der Detailanzeige; Sie müssen sich hierzu im Offline-Modus befinden.

Hier in Kurzform die Vorgehensweise:

1. Markieren Sie im Projektnavigator des SIMOTION SCOUT das Element **GERÄTEGLOBALE VARIABLEN** im Teilbaum des SIMOTION Geräts.
2. Wählen Sie in der Detailanzeige das Register **Symbolbrowser** und scrollen Sie bis zum Ende der Variablentabelle (leere Zeile).
3. Geben Sie in die letzte (leere) Zeile der Tabelle ein oder wählen Sie:
   - **Name** der Variablen
   - **Datentyp** der Variablen (nur elementare Datentypen zulässig)
4. Optional können Sie noch eingeben:
   - Aktivierung der Checkbox **Retain** (Hiermit wird die Variable als remanent deklariert; ihr Wert bleibt nach Spannungsausfall erhalten.)
   - **Feldlänge** (Größe des Arrays.)
   - **Anfangswert** (bei Arrays für jedes Element)
   - **Anzeigeformat** (bei Arrays für jedes Element)

Sie können nun mit dem Symbol-Browser oder jedem Programm des SIMOTION Geräts auf diese Variable zugreifen.

In ST-Quellen können Sie eine geräteglobale Variable wie jede andere Variable verwenden.

Hinweis

Wenn Sie Unit-Variablen oder lokale Variablen gleichen Namens (z. B. *var-name*) deklariert haben, spezifizieren Sie die geräteglobale Variable mit *_device.var-name*.

Eine Alternative zu geräteglobalen Variablen ist die Deklaration von Unit-Variablen in einer eigenen Unit, die in andere Units importiert wird. Dies bietet folgende Vorteile:

1. Es können Variablen-Strukturen verwendet werden.
2. Die Initialisierung der Variablen beim Übergang STOP-RUN ist möglich (über Programm in StartupTask).
3. Bei neu angelegten globalen Unit-Variablen ist auch ein Download im RUN möglich.

Siehe Funktionshandbuch SIMOTION Basisfunktionen.

6.2.3 Speicherbereiche der Variablentypen

Die verschiedenen Variablentypen werden in unterschiedlichen Speicherbereichen abgelegt, die zu verschiedenen Zeitpunkten initialisiert werden. Die Tabelle zeigt::

- die verfügbaren Speicherbereiche für Variablentypen, die in ST-Quellen deklariert werden (ggf. abhängig von der Version des SIMOTION Kernels),
- den Initialisierungszeitpunkt des jeweiligen Speicherbereichs.

Eine Erläuterung anhand eines Beispiels finden Sie im Abschnitt Beispiel für Speicherbereiche (Seite 243).

Tabelle 6-8 Durch verschiedene Variablentypen belegte Speicherbereiche und deren Initialisierung

Speicherbereich	Zugeordnete Variablentypen	Initialisierung[3]
Remanenter Speicher	Remanente Unit Variablen	Beim Download gemäß den Einstellungen für Download
Anwenderspeicher der Unit	• Nicht remanente Unit-Variablen • Mit VAR_GLOBAL deklarierte Instanzen von Funktionsbausteinen einschließlich der zugehörigen statischen Variablen (VAR, VAR_INPUT, VAR_OUTPUT) Außerdem bei aktivierter Compileroption (Seite 64) "Programminstanzdaten nur einmal anlegen": • Mit VAR deklarierte lokale Variablen der Programme der Unit • Mit VAR innerhalb der Programme der Unit deklarierte Instanzen von Funktionsbausteinen einschließlich der zugehörigen statischen Variablen (VAR, VAR_INPUT, VAR_OUTPUT)	• Beim Einschalten des Geräts. • Beim Download gemäß den Einstellungen für Download. • Beim Übergang in den Betriebszustand RUN: – Ab Version V4.2 des SIMOTION Kernels: Durch Einstellung am SIMOTION Gerät, Checkbox **Initialisierung der nicht remanenten globalen Variablen und Programmdaten beim STOP-RUN-Übergang**. – Ab Version V4.1 des SIMOTION Kernels: Wenn beim jeweiligen Deklarationsblock das nachstehende Pragma angegeben ist: { BlockInit_OnDeviceRun := ALWAYS; } Siehe auch Compiler mit Attributen steuern (Seite 312).
Anwenderspeicher der Task	Bei deaktivierter Compileroption (Seite 64) "Programminstanzdaten nur einmal anlegen" (Standard): • Mit VAR deklarierte lokale Variablen der zugeordneten Programme • Mit VAR innerhalb der zugeordneten Programme deklarierte Instanzen von Funktionsbausteinen einschließlich der zugehörigen statischen Variablen (VAR, VAR_INPUT, VAR_OUTPUT)	Entsprechend dem Ablaufverhalten der Task: • Sequentielle Tasks: Bei jedem Start der Task • Zyklische Tasks: Beim Übergang in den Betriebszustand RUN.

Speicherbereich	Zugeordnete Variablentypen	Initialisierung[3]
Lokaldatenstack der Task[2]	• Referenz (Zeiger) zu dem in der Task gerufenen Programm • Mit VAR_TEMP deklarierte lokale Variablen des in der Task gerufenen Programms	Bei jedem Aufruf des Programms in der Task
	• Referenz (Zeiger) zu gerufenen Instanzen von Funktionsbausteinen • Mit VAR_TEMP deklarierte lokale Variablen von Funktionsbausteinen • Mit VAR_IN_OUT deklarierte Durchgangsparameter von Funktionsbausteinen[1]	Bei jedem Aufruf der Instanz des Funktionsbausteins.
	• Mit VAR, VAR_INPUT oder VAR_IN_OUT[1] deklarierte Variablen von gerufenen Funktionen • Rückgabewert der gerufenen Funktionen	Bei jedem Aufruf der Funktion

[1] Referenzen (Zeiger) zu den übergebenen Variablen.

[2] Siehe auch Speicherbedarf der Variablen auf dem Lokaldatenstack (Seite 245).

[3] Ausführliche Beschreibung des Initialisierungsverhaltens der einzelnen Variablentypen siehe Zeitpunkt der Variableninitialisierung (Seite 246).

6.2.3.1 Beispiel für Speicherbereiche

Tabelle 6-9 Beispiel für Speicherbereiche der Variablentypen - Teil 1

```
INTERFACE
// Die Anweisungen im Interfaceabschnitt legen fest,
// welche Quellinhalte exportiert werden.
    FUNCTION FC1;
    FUNCTION_BLOCK FB1;
    PROGRAM p1;

    // Unit-Variablen des Interfaceabschnitts sind auch auf
    // HMI-Geräten sichtbar.
    VAR_GLOBAL               // Nicht remanente Unit-Variablen
                             // liegen im Anwenderspeicher der UNIT
        u1_if : INT;
    END_VAR
    VAR_GLOBAL CONSTANT      // Unit-Konstanten liegen
                             // im Anwenderspeicher der UNIT
    END_VAR
    VAR_GLOBAL RETAIN        // Remanente Unit-Variablen liegen
                             // im remanenten (netzausfallsicheren) Speicher
    END_VAR
END_INTERFACE

IMPLEMENTATION
// Der Implementationsabschnitt enthält in verschiedenen
// Programmorganisationseinheiten (POE) die ausführbaren Codeabschnitte
// Eine POE kann ein Programm, ein FC oder ein FB sein.
    // Unit-Variablen des Implementationsabschnitts können nur
    // innerhalb der Quelle verwendet werden.
    VAR_GLOBAL               // Nicht remanente Unit-Variablen liegen
                             // im Anwenderspeicher der UNIT
        u1_glob : INT;
    END_VAR
    VAR_GLOBAL CONSTANT      // Unit-Konstanten liegen
                             // im Anwenderspeicher der UNIT
    END_VAR
    VAR_GLOBAL RETAIN        // Remanente Unit-Variablen liegen
                             // im remanenten (netzausfallsicheren) Speicher
    END_VAR
//-----------------------------------------------------------------------
```

Tabelle 6-10 Beispiel für Speicherbereiche der Variablentypen - Teil 2

```
// Fortsetzung
//------------------------------------------------------------------------
    FUNCTION_BLOCK FB1 // Die Deklaration einer Instanz
    // eines Funktionsbausteins bestimmt, wo deren Daten liegen:
    // - als VAR_GLOBAL in einer Unit:
    //   im Anwenderspeicher der Unit
    // - als VAR in einem Programm:
    //   standardmäßig im Anwenderspeicher der Task
    // - als VAR in einem Funktionsbaustein:
    //   im Anwenderspeicher der Unit bzw. der Task,
    //   abhängig von der Instanzdeklaration des übergeordneten FB
    // Beim Aufruf der Instanz wird ein Zeiger zu den Instanzdaten
    // auf den Stack der aufrufenden Task abgelegt

        VAR_INPUT        // Eingangsparameter
                         // liegen im Anwenderspeicher,
                         // werden bei Aufruf der Instanz geschrieben
            fb_in     : INT;
        END_VAR
        VAR_OUTPUT       // Ausgangsparameter
                         // liegen im Anwenderspeicher
            fb_out    : INT;
        END_VAR
        VAR_IN_OUT       // Durchgangsparameter.
                         // Referenzen liegen im Anwenderspeicher,
                         // werden bei Aufruf der Instanz geschrieben
            fb_in_out : INT;
        END_VAR

        VAR              // Statische Variablen
                         // liegen im Anwenderspeicher
                         // können lokal im FB verwendet werden
             fb_var1   : INT;
        END_VAR

        VAR_TEMP         // Temporäre Variablen
                         // liegen auf dem Stack der aufrufenden Task,
                         // werden bei jedem Aufruf initialisiert
            fb_temp1  : INT;
        END_VAR

        // Code liegt im Anwenderspeicher der Unit.
        fb_var1   := fb_var1 + 1;
        fb_out    := fb_var1;
        fb_temp1  := fb_in_out;
        fb_in_out := fb_temp1 + fb_in;
    END_FUNCTION_BLOCK
//------------------------------------------------------------------------
```

Tabelle 6-11 Beispiel für Speicherbereiche der Variablentypen - Teil 3

```
// Fortsetzung
//----------------------------------------------------------------------
    FUNCTION FC1 : INT     // Die Daten der Funktion liegen auf dem
        // Stack der aufrufenden Task; sie werden bei jedem Aufruf
        // der Funktion initialisiert.
        // Der Rückgabewert liegt auf dem Stack der aufrufenden Task.

        VAR_INPUT        // Eingangsparameter
                         // liegen auf dem Stack der aufrufenden Task,
                         // werden bei Aufruf der Funktion geschrieben
            fc_in   : INT;
        END_VAR

        VAR              // Temporäre Variablen
                         // liegen auf dem Stack der aufrufenden Task
            fc_var  : INT;
        END_VAR
        // Code liegt in Anwenderspeicher der Unit
        fc_var := 567;
        fc1 := fc_in + fc_var;
    END_FUNCTION

    PROGRAM p1
        VAR              // Variablen liegen standardmäßig
                         // im Anwenderspeicher der Task
            p_var   : INT;
            p_varFB : FB1;
        END_VAR

        VAR_TEMP         // Temporäre Variablen
                         // liegen auf dem Stack der Task,
                         // werden bei jedem Durchlauf der Task initialisiert
            p_temp  : INT;
        END_VAR

        // Code liegt im Anwenderspeicher der Unit
        p_temp  := p_var;
        p_varFB (fb_in_out := p_temp);
        u1_glob := 4711;
    END_PROGRAM
END_IMPLEMENTATION
```

6.2.3.2 Speicherbedarf der Variablen auf dem Lokaldatenstack

Die Variablen, die auf dem Lokaldatenstack einer Task abgelegt werden, sind in Speicherbereiche der Variablentypen (Seite 240) aufgeführt. Die Stackgröße stellen Sie für jede Task im Register **Taskkonfiguration** ein. Siehe hierzu die Beschreibung für jede Anwender-Task im Funktionshandbuch Basisfuktionen.

Beachten Sie zum Speicherbedarf auf dem Lokaldatenstack:

- Temporäre lokale Variablen benötigen ihre einfache Größe auf dem Stack.
- Globale Variablen und statische lokale Variablen benötigen keine Ressourcen auf dem Stack.
  Nur wenn sie als Eingangsparameter einer Funktion dienen, benötigen sie die einfache Datengröße auf dem Stack.
- Wird eine Funktion mehrmals in einer Task aufgerufen, belastet sie den Stack nur einmal.
- Variablen vom Typ BOOL benötigen 1 Byte auf dem Stack.

Informationen über den Speicherbedarf einer POE auf dem Lokaldatenstack können Sie mit der Funktion Programmstruktur (Seite 302) erhalten.

6.2.4 Zeitpunkt der Variableninitialisierung

Der Zeitpunkt der Variableninitialisierung wird bestimmt:

- durch den Speicherbereich, dem die Variable zugeordnet ist
- durch Bedienhandlungen (z. B. durch den Download der Quelle ins Zielsystem)
- durch das Ablaufverhalten der Task (sequentiell, zyklisch), welcher das Programm zugeordnet wurde.

Alle Variablentypen und den Zeitpunkt der Variableninitialisierung zeigen Ihnen die folgenden Tabellen. Grundsätzliches zu den Tasks finden Sie im Funktionshandbuch *SIMOTION Basisfunktionen*.

Das Verhalten bzgl. der Variableninitialisierung während des Downloads ist einstellbar: Wählen Sie hierzu zur Voreinstellung das Menü **Extras > Einstellungen** und das Register **Download** bzw. legen Sie die Einstellung während des aktuellen Downloads fest.

Hinweis

Sie können Werte von Unit-Variablen oder geräteglobalen Variablen aus dem SIMOTION-Gerät in SIMOTION SCOUT hochladen (Upload) und im XML-Format speichern.

1. Speichern Sie mit der Funktion *_saveUnitDataSet* die gewünschten Datensegmente der Unit-Variablen oder geräteglobalen Variablen als Datensatz.
2. Wenden Sie in SIMOTION SCOUT die Funktion **Variablen sichern** an.

Mit der Umkehrfunktion **Variablen wiederherstellen** können Sie diese Datensätze und Variablen wieder in das SIMOTION-Gerät laden (Download).

Weitere Informationen siehe Projektierungshandbuch SIMOTION SCOUT.

Damit ist es z. B. möglich, diese Daten zu erhalten, obwohl Sie durch einen Download des Projekts initialisiert oder unbrauchbar werden (z. B. bei Versionswechsel des SIMOTION SCOUT).

6.2.4.1 Initialisierung remanenter globaler Variablen

Remanente Variablen behalten bei Spannungsausfall ihren letzten Wert. Alle anderen Daten werden beim Wiedereinschalten neu initialisiert.

Remanente globale Variablen werden initialisiert:

- Bei Ausfall der Stützung bzw. Pufferung der remanenten Daten.
- Bei Firmware-Update.
- Beim Urlöschen (MRES).
- Mit Funktion Restart (Del. SRAM) bei SIMOTION P320 oder P350.
- Durch Anwenden der Funktion *_resetUnitData*, selektiv für verschiedene Datensegmente der remanenten Daten möglich.
- Beim Download gemäß der nachfolgenden Beschreibung.

Verhalten bei Download

Tabelle 6-12 Initialisierung remanenter globaler Variablen beim Download

Variablentyp	Zeitpunkt der Variableninitialisierung
Remanente geräteglobale Variablen	Das Verhalten beim Download ist abhängig von der Einstellung *Initialisierung der remanenten Programmdaten und der remanenten geräteglobalen Variablen*[1]: • **Ja**[2]: Alle remanenten geräteglobalen Variablen werden initialisiert. • **Nein**[3]: Die remanenten geräteglobalen Variablen werden nur initialisiert, wenn sich deren Versionskennung geändert hat. Siehe: Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).
Remanente Unit-Variablen	Das Verhalten beim Download ist abhängig von der Einstellung *Initialisierung der remanenten Programmdaten und der remanenten geräteglobalen Variablen*[1]: • **Ja**[2]: Alle remanenten Unit-Variablen (aller Units) werden initialisiert. • **Nein**[3]: Ein Datenblock (= Deklarationsblock)[4] der remanenten Unit-Variablen im Interface- bzw. Implementationsabschnitt wir nur initialisiert[5], wenn sich dessen Versionskennung geändert hat. Siehe: Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).

[1] Voreinstellung im Menü **Extras > Einstellungen**, Register **Download**,
bzw. aktuelle Einstellung beim Download.

[2] Die entsprechende Checkbox ist aktiv.

[3] Die entsprechende Checkbox ist inaktiv.

[4] Bei der Programmiersprache **SIMOTION ST**:
Ein Datenblock der remanenten Unit-Variablen entspricht einem Deklarationsblock VAR_GLOBAL RETAIN / END_VAR im Interface- bzw. Implementationsabschnitt.
Bei den Programmiersprachen **SIMOTION MCC** und **SIMOTION KOP/FUP**:
Ein Datenblock der remanenten Unit-Variablen wird folgendermaßen aus den mit VAR_GLOBAL RETAIN deklarierten Variablen im Interface- bzw. Implementationsabschnitt der Deklarationstabelle gebildet: Pragma-Zeilen innerhalb eines Abschnitts der Deklarationstabelle trennen die Variablen in unterschiedliche Datenblöcke auf.

[5] Die Initialisierung eines geänderten Datenblocks erfolgt auch beim Download im RUN, sofern folgende Voraussetzung erfüllt ist:
Bei der Programmiersprache **SIMOTION ST**
Im jeweiligen Deklarationsblock ist innerhalb eines Pragmas das folgende Attribut angegeben: { BlockInit_On-Change := TRUE; }.
Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
In der Deklarationstabelle ist eine Pragma-Zeile eingefügt, bei der die folgende Checkbox aktiviert ist: *Initialisierung VAR_GLOBAL RETAIN bei Änderung*. Alle bis zur folgenden Pragma-Zeile oder dem Tabellenende mit VAR_GLOBAL RETAIN deklarierten Variablen bilden einen entsprechenden Datenblock.
Zu den allgemeinen Voraussetzungen für einen Download im RUN siehe Funktionshandbuch SIMOTION Basisfunktionen.

6.2.4.2 Initialisierung nicht remanenter globaler Variablen

Nicht remanente globale Variablen verlieren ihren Wert bei Spannungsausfall. Sie werden initialisiert:

- Bei der Initialisierung remanenter globaler Variablen (Seite 247) , z. B. bei Firmware-Update oder beim Urlöschen (MRES).
- Beim Einschalten.
- Durch Anwenden der Funktion *_resetUnitData*, selektiv für verschiedene Datensegmente der nicht remanenten Daten möglich.

- Beim Download gemäß der Beschreibung in der nachfolgenden Tabelle
- Beim Übergang vom Betriebszustand STOP in den Betriebszustand RUN gemäß der Beschreibung am Ende des Abschnitts.

Verhalten beim Download

Tabelle 6-13 Initialisierung nicht remanenter globaler Variablen beim Download

Variablentyp	Zeitpunkt der Variableninitialisierung
Nicht remanente geräteglobale Variablen	Das Verhalten beim Download ist abhängig von der Einstellung *Initialisierung der nicht remanenten Programmdaten und der nicht remanenten geräteglobalen Variablen*[1]: • **Ja**[2]: Alle nicht remanenten geräteglobalen Variablen werden initialisiert. • **Nein**[3]: Die nicht remanenten geräteglobalen Variablen werden nur initialisiert, wenn sich deren Versionskennung geändert hat. Siehe: Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).
Nicht remanente Unit-Variablen	Das Verhalten beim Download ist abhängig von der Einstellung *Initialisierung der nicht remanenten Programmdaten und der nicht remanenten geräteglobalen Variablen*[1]: • **Ja**[2]: Alle nicht remanenten Unit-Variablen (aller Units) werden initialisiert. • **Nein**[3]: Ein Datenblock (= Deklarationsblock)[4] der nicht remanenten Unit-Variablen im Interface- bzw. Implementationsabschnitt wir nur initialisiert[5], wenn sich dessen Versionskennung geändert hat. Siehe: Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).

[1] Voreinstellung im Menü **Extras > Einstellungen**, Register **Download** bzw. aktuelle Einstellung beim Download.

[2] Die entsprechende Checkbox ist aktiv.

[3] Die entsprechende Checkbox ist inaktiv.

[4] Bei der Programmiersprache **SIMOTION ST**:
Ein Datenblock der nicht remanenten Unit-Variablen entspricht einem Deklarationsblock VAR_GLOBAL / END_VAR im Interface- bzw. Implementationsabschnitt.
Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
Ein Datenblock der nicht remanenten Unit-Variablen wird folgendermaßen aus den mit VAR_GLOBAL deklarierten Variablen im Interface- bzw. Implementationsabschnitt der Deklarationstabelle gebildet: Pragma-Zeilen innerhalb eines Abschnitts der Deklarationstabelle trennen die Variablen in unterschiedliche Datenblöcke auf.

[5] Die Initialisierung eines geänderten Datenblocks erfolgt auch beim Download im RUN, sofern folgende Voraussetzung erfüllt ist:
Bei der Programmiersprache **SIMOTION ST**
Im jeweiligen Deklarationsblock ist innerhalb eines Pragmas das folgende Attribut angegeben: { BlockInit_On-Change := TRUE; }.
Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
In der Deklarationstabelle ist eine Pragma-Zeile eingefügt, bei der die folgende Checkbox aktiviert ist: *Initialisierung VAR_GLOBAL bei Änderung*. Alle bis zur folgenden Pragma-Zeile oder dem Tabellenende mit VAR_GLOBAL deklarierten Variablen bilden einen entsprechenden Datenblock.
Zu den allgemeinen Voraussetzungen für einen Download im RUN siehe Funktionshandbuch SIMOTION Basisfunktionen.

Verhalten beim STOP-RUN-Übergang

Standardmäßig bleiben die Werte nicht remanenter globaler Variablen beim Übergang vom Betriebszustand STOP in den Betriebszustand RUN erhalten.

Sie können jedoch einstellen, dass die nicht remanenten globalen Variablen beim STOP-RUN-Übergang initialisiert werden:

- **Ab Version V4.2 des SIMOTION Kernels** durch Einstellung am SIMOTION Gerät (Seite 317), Checkbox **Initialisierung der nicht remanenten globalen Variablen und Programmdaten beim STOP-RUN-Übergang**.
  Diese Einstellung kann bei nicht remanenten Unit-Variablen durch ein Pragma oder eine Pragma-Zeile in den jeweiligen Datenblöcken der Programmquellen überschrieben werden.
- **Ab Version V4.1 des SIMOTION Kernels** durch ein Pragma bzw. eine Pragma-Zeile in den jeweiligen Datenblöcken der Programmquellen (nur bei nicht remanenten Unit-Variablen):
  - Bei der Programmiersprache **SIMOTION ST**:
    Im jeweiligen Deklarationsblock VAR_GLOBAL / END_VAR ist innerhalb eines Pragmas das nachstehende Attribut angegeben: { BlockInit_OnDeviceRun := ALWAYS; }
  - Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
    In der Deklarationstabelle ist eine Pragma-Zeile mit folgender Einstellung eingefügt: "*Initialisierung bei STOP-RUN-Übergang* = **Immer**". Alle bis zur folgenden Pragma-Zeile oder dem Tabellenende mit VAR_GLOBAL deklarierten Variablen bilden einen Datenblock, der beim STOP-RUN-Übergang initialisiert wird.

Hinweis

Bei SIMOTION Geräten bis Version V4.0 des SIMOTION Kernels werden nicht remanente globale Variablen beim STOP-RUN-Übergang niemals initialisiert.

6.2.4.3 Initialisierung lokaler Variablen

Lokale Variablen werden initialisiert:

- Bei der Initialisierung remanenter Unit-Variablen (Seite 247)
- Bei der Initialisierung nicht remanenter Unit-Variablen (Seite 248)
- Zusätzlich gemäß nachfolgender Beschreibung

Tabelle 6-14 Initialisierung lokaler Variablen

Variablentyp	Zeitpunkt der Variableninitialisierung
Lokale Variablen von Programmen	Lokale Variablen von Programmen werden unterschiedlich initialisiert: • Statische Variablen (VAR) werden gemäß dem Speicherbereich initialisiert, in dem sie abgelegt werden. Siehe: Initialisierung statischer Variablen von Programmen (Seite 251). • Temporäre Variablen (VAR_TEMP) werden bei jedem Aufruf des Programms in der Task initialisiert.
Lokale Variablen von Funktionsbausteinen (FB)	Lokale Variablen von Funktionsbausteinen werden unterschiedlich initialisiert: • Statische Variablen (VAR, VAR_IN, VAR_OUT) werden nur dann initialisiert, wenn die Instanz des FB initialisiert wird. Siehe: Initialisierung von Instanzen von Funktionsbausteinen (FB) (Seite 253). • Temporäre Variablen (VAR_TEMP) werden bei jedem Aufruf der Instanz des FB initialisiert.
Lokale Variablen von Funktionen (FC)	Lokale Variablen von Funktionen sind temporär, sie werden bei jedem Aufruf der Funktion initialisiert.

Hinweis

Informationen über den Speicherbedarf einer POE auf dem Lokaldatenstack können Sie mit der Funktion Programmstruktur (Seite 302) erhalten.

6.2.4.4 Initialisierung statischer Variablen von Programmen

Die folgenden Ausführungen betreffen folgende statische Variablen:

- mit VAR deklarierte lokale Variablen der Programme einer Unit,
- mit VAR innerhalb der Programme einer Unit deklarierte Instanzen von Funktionsbausteinen einschließlich der zugehörigen statischen Variablen (VAR, VAR_INPUT, VAR_OUTPUT).

Das Initialisierungsverhalten wird bestimmt vom Speicherbereich, in der die statischen Variablen abgelegt werden. Dies wird über die Compileroption (Seite 64) "Programminstanzdaten nur einmal anlegen" bestimmt.

- Bei nicht aktivierter Compileroption "Programminstanzdaten nur einmal anlegen" (Standard):
  Die statischen Variablen werden im Anwenderspeicher jeder Task abgelegt, der das Programm zugeordnet ist.
  Die Initialisierung der Variablen ist deshalb abhängig vom Ablaufverhalten der Task, der das Programm zugeordnet ist (siehe Funktionshandbuch SIMOTION Basisfunktionen):
  - Sequentielle Tasks (MotionTasks, UserInterruptTasks, SystemInterruptTasks, StartupTask, ShutdownTask): Die statischen Variablen werden bei jedem Start der Task initialisiert.
  - Zyklische Tasks (BackgroundTask, SynchronousTasks, TimerInterruptTasks): Die statischen Variablen werden nur beim Übergang vom Betriebszustand STOP in den Betriebszustand RUN initialisiert.
- Bei aktivierter Compileroption "Programminstanzdaten nur einmal anlegen":
  Diese Einstellung ist z. B. nötig, wenn ein Programm innerhalb eines Programms aufgerufen werden soll.
  Die statischen Variablen aller Programme der betreffenden Programmquelle (Unit) werden nur einmal im Anwenderspeicher der Unit abgelegt.
  Sie werden deshalb zusammen mit den nicht remanenten Unit-Variablen initialisiert, siehe Initialisierung nicht remanenter globaler Variablen (Seite 248).
  Beim Übergang vom Betriebszustand STOP in den Betriebszustand RUN werden sie standardmäßig nicht initialisiert. Sie können jedoch einstellen, dass sie beim STOP-RUN-Übergang initialisiert werden:
  - **Ab Version V4.2 des SIMOTION Kernels** durch Einstellung am SIMOTION Gerät (Seite 317) , Checkbox **Initialisierung der nicht remanenten globalen Variablen und Programmdaten beim STOP-RUN-Übergang**.
    Diese Einstellung kann durch ein Pragma oder eine Pragma-Zeile im Datenblock der jeweiligen Programmorganisationseinheit (POE) überschrieben werden.
  - **Ab Version V4.1 des SIMOTION Kernels** durch ein Pragma bzw. eine Pragma-Zeile im Datenblock der jeweiligen Programmorganisationseinheit (POE):
    Bei der Programmiersprache **SIMOTION ST**:
    Im Deklarationsblock VAR / END_VAR ist innerhalb eines Pragmas das nachstehende Attribut angegeben: { BlockInit_OnDeviceRun := ALWAYS; }
    Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
    Die Deklarationstabelle beginnt mit einer Pragma-Zeile mit folgender Einstellung: "*Initialisierung bei STOP-RUN-Übergang* = **Immer**". Alle in der Tabelle mit VAR deklarierten Variablen werden beim STOP-RUN-Übergang initialisiert.

6.2.4.5 Initialisierung von Instanzen von Funktionsbausteinen (FB)

Die Initialisierung einer Instanz eines Funktionsbausteins (Seite 198) wird vom Ort ihrer Deklaration bestimmt:

- Globale Deklaration (innerhalb VAR_GLOBAL / END_VAR im Interface- oder Implementationsabschnitt):
  Initialisierung wie nicht remanente Unit-Variable, siehe Initialisierung nicht remanenter globaler Variablen (Seite 248).
- Lokale Deklaration in einem Programm (innerhalb VAR / END_VAR):
  Initialisierung wie statische Variable von Programmen, siehe Initialisierung statischer Variablen von Programmen (Seite 251).
- Lokale Deklaration in einem Funktionsbaustein (innerhalb VAR / END_VAR):
  Initialisierung wie eine Instanz dieses Funktionsbausteins.
- Deklaration als Durchgangsparameter in einem Funktionsbaustein oder einer Funktion (innerhalb VAR_IN_OUT / END_VAR):
  Bei der Initialisierung der POE wird nur die Referenz (Zeiger) initialisiert, die Instanz des Funktionsbausteins bleibt unverändert.

Hinweis

Informationen über den Speicherbedarf einer POE auf dem Lokaldatenstack können Sie mit der Funktion Programmstruktur (Seite 302) erhalten.

6.2.4.6 Initialisierung von Systemvariablen der Technologieobjekte

Die Systemvariablen eines Technologieobjekts sind in der Regel nicht remanent. Abhängig vom Technologieobjekt werden einige wenige Systemvariablen im remanenten Speicherbereich abgelegt (z. B. Absolutwertgeberjustage).

Das Initialisierungsverhalten entspricht (außer beim Download) dem für remanente bzw. nicht remanente globale Variablen. Siehe Initialisierung remanenter globaler Variablen (Seite 247) und Initialisierung nicht remanenter globaler Variablen (Seite 248).

Das Verhalten beim Download ist nachfolgend dargestellt für:

- Nicht remanente Systemvariablen
- Remanente Systemvariablen

Tabelle 6-15 Initialisierung der Systemvariablen der Technologieobjekte beim Download

Variablentyp	Zeitpunkt der Variableninitialisierung
Nicht remanente Systemvariablen	Verhalten beim Download, abhängig von der Einstellung *Initialisierung aller nicht remanenten Daten von Technologieobjekten*[1]: • **Ja**[2]: Alle Technologieobjekte werden initialisiert. – Alle Technologieobjekte werden neu aufgebaut, alle nicht remanenten Systemvariablen werden initialisiert. – Alle technologischen Alarme werden gelöscht. • **Nein**[3]: Nur in SIMOTION SCOUT veränderte Technologieobjekte werden initialisiert. – Die betreffenden Technologieobjekte werden neu aufgebaut, alle nicht remanenten Systemvariablen werden initialisiert. – Alle Alarme, die an den betreffenden Technologieobjekten anstehen, werden gelöscht. – Wenn an einem nicht zu initialisierenden Technologieobjekt ein Alarm ansteht, der nur mit *Power On* quittiert werden kann, wird der Download abgebrochen.
Remanente Systemvariablen	Nur wenn ein Technologieobjekt in SIMOTION SCOUT verändert wurde, werden dessen remanente Systemvariablen initialisiert. Die remanenten Systemvariablen aller anderen Technologieobjekte bleiben erhalten (z. B. Absolutwertgeberjustage)

[1] Voreinstellung im Menü **Extras > Einstellungen**, Register **Download** bzw. aktuelle Einstellung beim Download.

[2] Entsprechende Checkbox ist aktiv.

[3] Entsprechende Checkbox ist inaktiv.

6.2.4.7 Versionskennung globaler Variablen und deren Initialisierung beim Download

Tabelle 6-16 Versionskennung globaler Variablen und deren Initialisierung beim Download

Datensegment		Beschreibung der Versionskennung
Gerätegloabale Variablen		
	Remanente gerätegloabale Variablen Nicht remanente gerätegloabale Variablen	• Eigene Versionskennung für jedes Datensegment der gerätegloabalen Variablen • Die Versionskennung eines Datensegments ändert sich bei: – Hinzufügen oder Entfernen einer Variablen innerhalb des Datensegments – Änderung des Bezeichners oder des Datentyps einer Variablen innerhalb des Datensegments • Keine Änderung dieser Versionskennung bei: – Änderungen im anderen Datensegment – Änderung von Initialisierungswerten[1] • Beim Download[2] gilt: Nur wenn sich die Versionskennung eines Datensegments geändert hat, wird dieses Datensegment initialisiert
Unit-Variablen einer Unit		

Datensegment		Beschreibung der Versionskennung
	Remanente Unit-Variablen im Interfaceabschnitt Remanente Unit-Variablen im Implementations-abschnitt Nicht remanente Unit-Variablen im Interfaceabschnitt Nicht remanente Unit-Variablen im Implementations-abschnitt	• Mehrere Datenblöcke (= Deklarationsblöcke)[3] in jedem Datensegment möglich • **Eigene Versionskennung** für jeden Datenblock • Die Versionskennung eines Datenblocks ändert sich bei: – Hinzufügen oder Entfernen einer Variablen im betreffenden Deklarationsblock – Änderung der Reihenfolge der Variablen im betreffenden Deklarationsblock – Änderung des Bezeichners oder des Datentyps einer Variablen im betreffenden Deklarationsblock – Änderung einer Datentypdefinition (aus eigener oder importierter[4] Unit), die im betreffenden Deklarationsblock verwendet wird – Hinzufügen oder Entfernen von Deklarationsblöcken innerhalb desselben Datensegments **vor** dem betreffenden Deklarationsblock • Keine Änderung dieser Versionskennung bei: – Hinzufügen oder Entfernen von Deklarationsblöcken in anderen Datensegmenten – Hinzufügen oder Entfernen von Deklarationsblöcken innerhalb desselben Datensegments **hinter** dem betreffenden Deklarationsblock – Änderungen in anderen Deklarationsblöcken – Änderung von Initialisierungswerten[1] – Änderungen von Datentypdefinitionen, die im betreffenden Deklarationsblock nicht verwendet werden – Änderungen von Funktionen • Beim Download[2] gilt: Nur wenn sich die Versionskennung eines Datenblocks geändert hat, wird dieser Datenblock initialisiert[5] • Funktionen zur Datensicherung und -initialisierung berücksichtigen die Versionskennung der Datenblöcke

[1] Geänderte Initialisierungswerte werden erst wirksam, wenn der betreffende Datenblock bzw. Datensegment initialisiert wird.

[2] Bei Einstellung *Initialisierung der remanenten Programmdaten und der remanenten geräteglobalen Variablen* = Nein und *Initialisierung der nicht remanenten Programmdaten und der nicht remanenten geräteglobalen Variablen* = Nein.
Bei anderen Einstellungen: siehe die Abschnitte "Initialisierung remanenter globaler Variablen (Seite 247)" bzw. "Initialisierung nicht remanenter globaler Variablen (Seite 248)".

[3] Bei der Programmiersprache **SIMOTION ST**:
Ein Datenblock entspricht einem Deklarationsblock VAR_GLOBAL / END_VAR bzw. VAR_GLOBAL RETAIN / END_VAR im Interface- bzw. Implementationsabschnitt.
Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
Ein Datenblock der nicht remanenten Unit-Variablen wird folgendermaßen aus den mit VAR_GLOBAL bzw. VAR_GLOBAL RETAIN deklarierten Variablen im Interface- bzw. Implementationsabschnitt der Deklarationstabelle gebildet: Pragma-Zeilen innerhalb eines Abschnitts der Deklarationstabelle trennen die Variablen in unterschiedliche Datenblöcke auf.

[4] Der Import von Units ist abhängig von der Programmiersprache, siehe den entsprechenden Abschnitt (Seite 227).

[5] Die Initialisierung eines geänderten Datenblocks erfolgt auch beim Download im RUN, sofern folgende Voraussetzung erfüllt ist:
Bei der Programmiersprache **SIMOTION ST**
Im jeweiligen Deklarationsblock ist innerhalb eines Pragmas (Seite 307) das folgende Attribut (Seite 312) angegeben:
{ BlockInit_OnChange := TRUE; }.
Bei den Programmiersprachen **SIMOTION MCC** oder **SIMOTION KOP/FUP**:
In der Deklarationstabelle ist eine Pragma-Zeile eingefügt, bei der die folgende Checkbox aktiviert ist: *Initialisierung VAR_GLOBAL bei Änderung*. Alle bis zur folgenden Pragma-Zeile oder dem Tabellenende mit VAR_GLOBAL deklarierten Variablen bilden einen entsprechenden Datenblock.
Zu den allgemeinen Voraussetzungen für einen Download im RUN siehe Funktionshandbuch SIMOTION Basisfunktionen.

6.2.5 Variablen und HMI-Geräte

Folgende Variablen werden an HMI-Geräte exportiert und stehen dort zur Verfügung:

- Systemvariablen des SIMOTION Geräts
- Systemvariablen der Technologieobjekte
- I/O-Variablen
- Geräteglobale Variablen
- Remanente und nicht remanente Unit-Variablen des Interfaceabschnitts (Voreinstellung). Diese Voreinstellung ändern Sie folgendermaßen:
  - In der Programmiersprache SIMOTION ST:
    Für jeden Deklarationsblock mit nachstehendem Pragma:
    `{ HMI_Export := FALSE; }.`
    Siehe auch Compiler mit Attributen steuern (Seite 312).
  - In den Programmiersprachen SIMOTION MCC und SIMOTION KOP/FUP:
    Für die auf eine Pragma-Zeile folgenden Variablendeklarationen, wenn Sie in dieser Pragma-Zeile die Parameter *VAR_GLOBAL für HMI-Geräte* bzw. *VAR_GLOBAL RETAIN für HMI Geräte* deaktivieren.

  Unit-Variablen eines solchen Datenblocks werden nicht an HMI-Geräte exportiert. Beim Download entfällt für sie auch die HMI-Konsistenzprüfung.

Folgende Variablen werden **nicht** an HMI-Geräte exportiert und stehen dort **nicht** zur Verfügung:

- Remanente und nicht remanente Unit-Variablen des Implementationsabschnitts (Voreinstellung)
  Diese Voreinstellung ändern Sie folgendermaßen:
  - In der Programmiersprache SIMOTION ST:
    Für jeden Deklarationsblock mit nachstehendem Pragma:
    `{ HMI_Export := TRUE; }`
    Siehe auch Compiler mit Attributen steuern (Seite 312).
  - In den Programmiersprachen SIMOTION MCC und SIMOTION KOP/FUP:
    Für die auf eine Pragma-Zeile folgenden Variablendeklarationen, wenn Sie in dieser Pragma-Zeile die Parameter *VAR_GLOBAL für HMI-Geräte* bzw. *VAR_GLOBAL RETAIN für HMI Geräte* aktivieren.

  Unit-Variablen eines solchen Datenblocks werden an HMI-Geräte exportiert. Beim Download unterliegen sie deshalb der HMI-Konsistenzprüfung.
- Lokale Variablen einer POE

Hinweis

Die Gesamtgröße der Unit-Variablen, die an HMI-Geräte exportiert werden kann, ist auf 64 kByte pro Unit beschränkt.

Die Wirkung des Pragmas `{ HMI_Export := FALSE; } / { HMI_Export := TRUE; }` (bzw. der Einstellungen *VAR_GLOBAL für HMI-Geräte* oder *VAR_GLOBAL RETAIN für HMI Geräte* in einer Pragma-Zeile) ist abhängig von der Version des SIMOTION Kernels:

- Ab Version V4.1 des SIMOTION Kernels:
  Das Pragma wirkt auf den Export des entsprechenden Deklarationsblocks an HMI-Geräte **und** auf den Aufbau des HMI-Adressraums:
  - Nur Variablen in Deklarationsblöcken, die an HMI-Geräte exportiert werden, belegen den HMI-Adressraum.
  - Innerhalb des HMI-Adressraums sind die Variablen gemäß ihrer Deklarationsreihenfolge angeordnet.
- Bis Version V4.0 des SIMOTION Kernels:
  Das Pragma wirkt **nur** auf den Export des entsprechenden Deklarationsblocks an HMI-Geräte.
  Der HMI-Adressraum wird auch von Unit-Variablen des Interfaceabschnitts belegt, deren Deklarationsblöcke nicht an HMI-Geräte exportiert werden.
  Innerhalb des HMI-Adressraums werden die Variablen in folgender Reihenfolge sortiert:
  - Remanente Unit-Variablen des Interfaceabschnitts (exportierte und nicht exportierte)
  - Remanente Unit-Variablen des Implementationsabschnitts (nur exportierte)
  - Nicht remanente Unit-Variablen des Interfaceabschnitts (exportierte und nicht exportierte)
  - Nicht remanente Unit-Variablen des Implementationsabschnitts (nur exportierte)

  Innerhalb dieser Segmente sind die Variablen gemäß ihrer Deklarationsreihenfolge angeordnet.

Beispiel für Programmiersprache SIMOTION ST

Tabelle 6-17 Beispiel für die Steuerung des HMI-Exports mit dem entsprechenden Pragma

```
INTERFACE
    VAR_GLOBAL
        // HMI-Export
        x1 : DINT;
    END_VAR
    VAR_GLOBAL
        { HMI_Export := FALSE; }
        // Kein HMI-Export
        x2 : DINT;
    END_VAR
    // ...
END_INTERFACE

IMPLEMENTATION
    VAR_GLOBAL
        // Kein HMI-Export
        y1 : DINT;
    END_VAR
    VAR_GLOBAL
        { HMI_Export := TRUE; }
        // HMI-Export
        y2 : DINT;
    END_VAR
    // ...
END_IMPLEMENTATION
```

6.3 Zugriff auf Ein- und Ausgänge (Prozessabbild, I/O-Variablen)

6.3.1 Überblick Zugriff auf Ein- und Ausgänge

SIMOTION bietet mehrere Möglichkeiten, auf die Ein- und Ausgänge des SIMOTION Geräts sowie auf die zentrale und dezentrale Peripherie zuzugreifen:

- über Direktzugriff mit I/O-Variablen
  Mit dem Direktzugriff greifen Sie direkt auf die entsprechende I/O-Adresse zu.
  Sie definieren eine I/O-Variable (Name und I/O-Adresse), ohne ihr eine Task zuzuordnen.
  Der gesamte Adressraum des SIMOTION Geräts ist nutzbar.
  Den Direktzugriff verwenden Sie bevorzugt bei sequenzieller Programmierung (in MotionTasks); hier ist der Zugriff auf den aktuellen Wert der Ein- und Ausgänge zu einem bestimmten Zeitpunkt besonders wichtig.
  Weitere Informationen: Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263).
- über das Prozessabbild der zyklischen Tasks mit I/O-Variablen
  Das Prozessabbild der zyklischen Tasks ist ein Speicherbereich im RAM des SIMOTION Geräts, auf dem der gesamte I/O-Adressraum des SIMOTION Geräts gespiegelt ist. Das Spiegelbild jeder I/O-Adresse ist einer zyklischen Task zugeordnet und wird mit dieser aktualisiert. Es ist während des gesamten Zyklus der Task konsistent. Dieses Prozessabbild wird bevorzugt bei der Programmierung der zugeordneten Task (zyklische Programmierung) eingesetzt.
  Sie definieren eine I/O-Variable (Name und I/O-Adresse) und ordnen ihr eine Task zu. Der gesamte Adressbereich des SIMOTION Geräts ist nutzbar.
  Der Direktzugriff auf diese I/O-Variable ist dennoch möglich: Spezifizieren Sie den Direktzugriff mit *_direct.var-name*.
  Weitere Informationen: Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263).
- mit dem festen Prozessabbild der BackgroundTask
  Das Prozessabbild der BackgroundTask ist ein Speicherbereich im RAM des SIMOTION Geräts, auf dem eine Untermenge des I/O-Adressraums des SIMOTION Geräts gespiegelt ist. Dieses Spiegelbild wird mit der BackgroundTask aktualisiert und ist während deren gesamten Zyklus konsistent. Dieses Prozessabbild wird bevorzugt bei der Programmierung der BackgroundTask (zyklische Programmierung) eingesetzt.
  Der Adressbereich 0 .. 63 ist nutzbar. Ausgenommen sind I/O-Adressen, auf die mit dem Prozessabbild der zyklischen Tasks zugegriffen wird.
  Weitere Informationen: Zugriffe auf festes Prozessabbild der BackgroundTask (Seite 273).

Eine Gegenüberstellung der wichtigsten Eigenschaften finden Sie in "Wichtige Eigenschaften von Direktzugriff und Prozessabbild" (Seite 260).

I/O-Variablen können Sie wie jede andere Variable verwenden, siehe "Auf I/O-Variablen zugreifen" (Seite 285).

Hinweis

Der Zugriff über das Prozessabbild ist performanter als der Direktzugriff.

6.3.2 Wichtige Eigenschaften von Direktzugriff und Prozessabbild

Tabelle 6-18 Wichtige Eigenschaften von Direktzugriff und Prozessabbild

	Direktzugriff	Zugriff auf Prozessabbild der zyklischen Tasks	Zugriff auf festes Prozessabbild der BackgroundTask
Zulässiger Adressbereich	Gesamter Adressbereich des SIMOTION Geräts. **Ausnahme**: I/O-Variablen, die mehrere Bytes umfassen, dürfen die Adressen 63 und 64 nicht zusammenhängend enthalten (Beispiel: PIW63 oder PQD62 sind nicht erlaubt.).		Adressen 0 .. 63. **Ausnahme**: Bis Version V4.1 des SIMOTION Kernels oder bei der Einstellung "Separates Prozessabbild" sind Adressen unzulässig, die beim Prozessabbild der zyklischen Tasks verwendet werden.
Projektierung der Adressen	Notwendig. Verwendete Adressen müssen in der Peripherie vorhanden und entsprechend projektiert sein. Die "Regeln für I/O-Adressen für Direktzugriff und das Prozessabbild der zyklischen Tasks" (Seite 267) sind zu beachten.		Nicht notwendig. Es können auch in der Peripherie nicht vorhandene oder projektierte Adressen verwendet werden.
Zugeordnete Task	Keine.	Zyklische Task wählbar: • SynchronousTasks, • TimerInterruptTasks, • BackgroundTask.	BackgroundTask.
Speicherbereich der Prozessabbilder	-	Abhängig von der Version des SIMOTION Kernels: • Bis Version V4.1: Immer separate Speicherbereiche. • Ab Version V4.2: Gemeinsamer Speicherbereich wählbar.	

	Direktzugriff	Zugriff auf Prozessabbild der zyklischen Tasks	Zugriff auf festes Prozessabbild der BackgroundTask
Aktualisierung	• Bei der Onboard-Peripherie der SIMOTION-Geräte C230-2, C240 und C240 PN: Die Aktualisierung erfolgt in einem Takt von 125 µs. • Bei Peripherie über taktsynchronen PROFIBUS DP bzw. PROFINET, über DRIVE-CLiQ sowie bei der Onboard-Peripherie der SIMOTION D-Geräte: Die Aktualisierung erfolgt im Lageregler-Takt[1]. • Bei Peripherie über nicht taktsynchronen PROFIBUS DP bzw. PROFINET sowie über P-Bus: Die Aktualisierung erfolgt im Interpolator-Takt[1]. Die Eingänge werden zu Beginn des Takts gelesen. Die Ausgänge werden am Ende des Takts geschrieben.	Die Aktualisierung erfolgt mit der zugeordneten Task: • Die Eingänge werden vor Start der zugeordneten Task gelesen und in das Prozessabbild der Eingänge übertragen. • Das Prozessabbild der Ausgänge wird nach Beenden der zugeordneten Task auf die Ausgänge geschrieben.	Die Aktualisierung erfolgt mit der *BackgroundTask*: • Die Eingänge werden vor Start der *BackgroundTask* gelesen und in das Prozessabbild der Eingänge übertragen. • Das Prozessabbild der Ausgänge wird nach Beenden der *BackgroundTask* auf die Ausgänge geschrieben.
Konsistenz	–	Während des gesamten Zyklus der zugeordneten Task. **Ausnahme**: Direktzugriff auf Ausgang erfolgt.	Während des gesamten Zyklus der *BackgroundTask*. **Ausnahme**: Direktzugriff auf Ausgang erfolgt.
	Die Konsistenz ist nur für elementare Datentypen gewährleistet. Bei der Verwendung von Arrays hat der Anwender selbst für die Konsistenz der Daten zu sorgen.		
Verwendung	Bevorzugt in MotionTasks	Bevorzugt in der zugeordneten Task	Bevorzugt in der Background-Task
Deklaration als Variable	Notwendig, geräteglobal als I/O-Variable im Symbolbrowser. Jedes Byte des Adressbereichs darf nur einer einzigen I/O-Variablen zugeordnet werden. Syntax der I/O-Adresse: z. B. PIW1022, PQ63.3.		Möglich, nicht notwendig: • Geräteglobal als I/O-Variable im Symbolbrowser, • Als Unit-Variable, • Als lokale statische Variable in einem Programm.
Download neuer oder geänderter I/O-Variablen	Nur im Betriebszustand STOP möglich.		-
Verwenden der absoluten Adresse	Nicht möglich.		Möglich, mit folgender Syntax: z. B. %IW62, %Q63.3.

	Direktzugriff	Zugriff auf Prozessabbild der zyklischen Tasks	Zugriff auf festes Prozessabbild der BackgroundTask
Byte-Anordnung beim Bilden des Prozessabbilds	-	Wie von Peripherie geliefert.	Abhängig von Version des SIMOTION Kernels und der Einstellung zum Speicherbereich der Prozessabbilder: • Bis Version V4.1 oder Einstellung "Separates Prozessabbild": Immer Big Endian. • Ab Version V4.2 und Einstellung "Gemeinsames Prozessabbild": Wie von Peripherie geliefert.
Byte-Anordnung beim Zugriff	Abhängig von Peripherie		Immer Big Endian
Beschreibbarkeit der Eingänge	Nein.	Abhängig von Version des SIMOTION Kernels: • Bis Version V4.1: Nein. • Ab Version V4.2: Ja.	Ja.
Schreibschutz für Ausgänge	Möglich; Status **Nur Lesen** wählbar.	Nicht möglich.	Nicht möglich.
Deklaration von Arrays	Möglich.		Nicht möglich.
Weitere Informationen	Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263).		Zugriffe auf festes Prozessabbild der BackgroundTask (Seite 273).
Verhalten im Fehlerfall	Fehler beim Zugriff aus Anwenderprogramm, Reaktion wählbar: • CPU-Stop[2] • Ersatzwert • Letzter Wert Siehe Funktionsbeschreibung SIMOTION Basisfunktionen.	Fehler beim Bilden des Prozessabbilds, Reaktion wählbar: • CPU-Stop[3] • Ersatzwert • Letzter Wert	Fehler beim Bilden des Prozessabbilds, Reaktion: CPU-Stop[3]. **Ausnahme**: Wenn auf dieselbe Adresse ein Direktzugriff erstellt wurde, gilt das dort eingestellte Verhalten.
		Siehe Funktionsbeschreibung SIMOTION Basisfunktionen.	
Zugriffe			
• Im Betriebszustand RUN	Uneingeschränkt möglich.	Uneingeschränkt möglich.	Uneingeschränkt möglich.

	Direktzugriff	Zugriff auf Prozessabbild der zyklischen Tasks	Zugriff auf festes Prozessabbild der BackgroundTask
• Während der StartupTask	Eingeschränkt möglich: • Eingänge können gelesen werden. • Ausgänge werden erst am Ende der StartupTask geschrieben.	Eingeschränkt möglich: • Eingänge werden zu Beginn der StartupTask gelesen. • Ausgänge werden erst am Ende der StartupTask geschrieben.	Eingeschränkt möglich: • Eingänge werden zu Beginn der StartupTask gelesen. • Ausgänge werden erst am Ende der StartupTask geschrieben.
• Während der ShutdownTask	Uneingeschränkt möglich.	Eingeschränkt möglich: • Eingänge behalten den Status der letzten Aktualisierung • Ausgänge werden nicht mehr geschrieben.	Eingeschränkt möglich: • Eingänge behalten den Status der letzten Aktualisierung • Ausgänge werden nicht mehr geschrieben.

[1] Bei folgenden SIMOTION-Geräten erfolgt die Aktualisierung im Servo_fast-Takt bzw. IPO_fast-Takt, falls diese Takte konfiguriert sind: D445-2 DP/PN, D455-2 DP/PN (ab Version V4.2) und D435-2 DP/PN (ab Version V4.3).

[2] Aufruf der ExecutionFaultTask.

[3] Aufruf der PeripheralFaultTask

6.3.3 Direktzugriff und Prozessabbild der zyklischen Tasks

Eigenschaft

Der Direktzugriff auf Ein- und Ausgänge und der Zugriff auf das Prozessabbild der zyklischen Tasks erfolgt immer über I/O-Variablen. Der gesamte Adressbereich des SIMOTION Geräts (Seite 266) ist nutzbar.

Eine Gegenüberstellung der wichtigsten Eigenschaften auch im Vergleich zum festen Prozessabbild der BackgroundTask (Seite 273) finden Sie in "Wichtige Eigenschaften von Direktzugriff und Prozessabbild (Seite 260)".

Hinweis

Beachten Sie die Regeln für I/O-Adressen für Direktzugriff und das Prozessabbild der zyklischen Tasks (Seite 267).

Insbesondere muss jede Adresse, die in einer I/O-Variablen verwendet wird, in der Peripherie vorhanden und projektiert sein; jedes Byte des Adressbereichs darf höchstens einer I/O-Variablen zugeordnet sein (außer Zugriffe mit Datentyp BOOL).

Ein detaillierter Status der I/O-Variablen (Seite 271) kann ab Version V4.2 des SIMOTION Kernels gelesen werden; um z. B. die Verfügbarkeit der I/O-Variablen zu prüfen.

Direktzugriff

Mit dem Direktzugriff greifen Sie direkt auf die entsprechende I/O-Adresse zu. Den Direktzugriff verwenden Sie bevorzugt bei sequenzieller Programmierung (in MotionTasks). Hier ist der Zugriff auf den aktuellen Wert der Ein- und Ausgänge zu einem bestimmten Zeitpunkt besonders wichtig.

Für den Direktzugriff definieren Sie eine I/O-Variable (Seite 267), ohne ihr eine Task zuzuordnen.

Prozessabbild der zyklischen Task

Das Prozessabbild der zyklischen Tasks ist ein Speicherbereich im RAM des SIMOTION Geräts, auf dem der gesamte I/O-Adressraum des SIMOTION Geräts gespiegelt ist. Das Spiegelbild jeder I/O-Adresse ist einer zyklischen Task zugeordnet und wird mit dieser aktualisiert. Es ist während des gesamten Zyklus der Task konsistent. Dieses Prozessabbild wird bevorzugt bei der Programmierung der zugeordneten Task (zyklische Programmierung) eingesetzt. Hier ist die Konsistenz während des gesamten Zyklus der Task besonders wichtig.

Für das Prozessabbild der zyklischen Tasks definieren Sie eine I/O-Variable (Seite 267) und ordnen ihr eine Task zu.

Der **Direktzugriff** auf diese I/O-Variable ist dennoch möglich: Spezifizieren Sie den Direktzugriff mit *_direct.var-name*.

Hinweis

Der Zugriff über das Prozessabbild ist performanter als der Direktzugriff.

Zusätzliche Eigenschaften ab Version V4.2 des SIMOTION Kernels

Ab Version V4.2 des SIMOTION Kernels verfügt der Direktzugriff auf die Ein- und Ausgänge sowie das Prozessabbild der zyklischen Tasks über zusätzliche Eigenschaften;

- Beim Prozessabbild der zyklischen Tasks ist ein gemeinsamer Speicherbereich mit dem festen Prozessabbild der BackgroundTask einstellbar (Standard bei neu angelegten Geräten).
- Beim Prozessabbild der zyklischen Tasks sind die I/O-Variablen für Eingänge beschreibbar, d. h. ihnen können Werte zugewiesen werden.
- Ein detaillierter Status der I/O-Variablen (Seite 271) kann gelesen werden, um z. B. die Verfügbarkeit der I/O-Variablen zu prüfen.

Speicherbereich mit dem festen Prozessabbild der BackgroundTask

- **Ab Version V4.2 des SIMOTION Kernels** kann durch eine Einstellung am Gerät (Seite 317) "Gemeinsames Prozessabbild" gewählt werden, dass der Speicherbereich für das feste Prozessabbild der BackgroundTask eine Untermenge des Speicherbereichs für das Prozessabbilds der zyklischen Tasks ist.
- **Bis Version V4.1 des SIMOTION Kernels** bzw. bei der Einstellung am Gerät "Separates Prozessabbild" (ab Version V4.2 des SIMOTION Kernels) belegen das feste Prozessabbild der BackgroundTask und das Prozessabbild der zyklischen Tasks unterschiedliche Speicherbereiche.

Hinweis

Nur wenn Sie zusätzlich das feste Prozessabbild der BackgroundTask verwenden: Beachten Sie die Auswirkungen der Einstellungen "Gemeinsames Prozessabbild" oder "Separates Prozessabbild" auf das feste Prozessabbild der BackgroundTask (Seite 273).

Tabelle 6-19 Einfluss der Einstellungen "Gemeinsames Prozessabbild" bzw. "Separates Prozessabbild" auf das Prozessabbild der zyklischen Tasks

	Gemeinsames Prozessabbild	Separates Prozessabbild
Verfügbarkeit	Erst ab Version V4.2 des SIMOTION Kernels verfügbar: • Einstellung wählbar. • Standard für neu angelegte Geräte.	Bis Version V4.1 des SIMOTION Kernels gilt: • Systemeigenschaft, nicht konfigurierbar. Ab Version V4.2 des SIMOTION Kernels gilt: • Einstellung wählbar. • Standard beim Hochrüsten des Geräts.
Download neuer oder geänderter I/O-Variablen	Nur im Betriebszustand STOP möglich.	Nur im Betriebszustand STOP möglich.
Byte-Anordnung beim Bilden des Prozessabbilds und beim Zugriff	Entsprechend der angeschlossenen Peripherie.	
Auswirkungen auf das feste Prozessabbild der BackgroundTask	Siehe entsprechende Tabelle in "Zugriffe auf festes Prozessabbild der BackgroundTask" (Seite 273).	
Weitere Informationen	Gemeinsames Prozessabbild (Seite 275).	Separates Prozessabbild (Seite 278).

6.3.3.1 Adressbereich der SIMOTION Geräte

Nachfolgend ist der Adressbereich der SIMOTION-Geräte in Abhängigkeit von der Version des SIMOTION Kernels angegeben. Bei Direktzugriff und Prozessabbild der zyklischen Tasks ist der gesamte Adressbereich nutzbar.

Tabelle 6-20 Adressbereich der SIMOTION-Geräte in Abhängigkeit von der Version des SIMOTION Kernels

SIMOTION Gerät	Adressbereich bei Version des SIMOTION Kernels					
	V3.2	V4.0	V4.1	4.2	V4.3	V4.4
C230-2	0 .. 2047 [3]	0 .. 2047 [3]	0 .. 2047 [3]	–	–	-
C240	–	0 .. 4095 [3]	0 .. 4095 [3]	0 .. 4095 [3]	0 .. 4095 [3]	0 .. 4095 [3]
C240 PN [1]	–	–	0 .. 4095 [4]	0 .. 4095 [4]	0 .. 4095 [4]	0 .. 4095 [4]
D410 DP	–	–	0 .. 8191 [3]	0 .. 8191 [3]	0 .. 8191 [3]	–
D410 PN	–	–	0 .. 8191 [4]	0 .. 8191 [4]	0 .. 8191 [4]	–
D410-2	–	–	–	–	0 .. 8191 [3 4]	0 .. 8191 [3 4]
D425	0 .. 4095 [3]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	–
D425-2	–	–	–	–	0 .. 16383 [3 4]	0 .. 16383 [3 4]
D435	0 .. 4095 [3]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	–
D435-2	–	–	–	–	0 .. 16383 [3 4]	0 .. 16383 [3 4]
D445	0 .. 4095 [3]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	–	–
D445-1 [1]	–	–	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]	–
D445-2	–	–	–	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]
D455-2	–	–	–	0 .. 16383 [3 4]	0 .. 16383 [3 4]	0 .. 16383 [3 4]
P320 [2]	–	–	0 .. 4095 [4]	0 .. 4095 [4]	0 .. 4095 [4]	0 .. 4095 [4]
P350	0 .. 2047 [3]	0 .. 4095 [3]	0 .. 4095 [3 4]	0 .. 4095 [3 4]	0 .. 4095 [3 4]	0 .. 4095 [3 4]

[1] Ab V4.1 SP2 HF4 verfügbar.

[2] Ab V4.1 SP5 verfügbar.

[3] Für die dezentrale Peripherie über PROFIBUS DP ist der Übertragungsumfang auf 1024 Bytes je PROFIBUS DP-Strang beschränkt.

[4] Für die dezentrale Peripherie über PROFINET ist der Übertragungsumfang auf 4096 Bytes je PROFINET-Segment beschränkt.

6.3.3.2 Regeln für I/O-Adressen für Direktzugriff und das Prozessabbild der zyklischen Tasks

Hinweis

Bei den Adressen für I/O-Variablen für Direktzugriff und das Prozessabbild der zyklischen Task (Seite 263) müssen Sie folgende Regeln beachten. Die Einhaltung der Regeln wird bei der Konsistenzprüfung des SIMOTION Projekts (z. B. beim Download) überprüft.

1. Für I/O-Variablen verwendete Adressen müssen in der Peripherie vorhanden und in HW Konfig entsprechend projektiert sein.
2. I/O-Variablen, die mehrere Bytes umfassen, dürfen die Adressen 63 und 64 nicht zusammenhängend enthalten.
   Folgende I/O-Adressen sind nicht erlaubt:
   - Eingänge: PIW63, PID61, PID62, PID63
   - Ausgänge: PQW63, PQD61, PQD62, PQD63
3. Alle Adressen einer I/O-Variablen, die mehrere Bytes umfasst, (z. B. Datentyp WORD, ARRAY) müssen innerhalb eines in HW Konfig projektierten zusammenhängenden Adressbereichs liegen, z. B. innerhalb des Adressbereichs (Slot bzw. Subslot) *einer* Peripheriebaugruppe.
4. Eine I/O-Adresse (Eingang bzw. Ausgang) kann nur von einer einzigen I/O-Variablen vom Datentyp BYTE, WORD oder DWORD bzw. einem Array dieser Datentypen verwendet werden. Zugriffe auf einzelne Bits mit I/O-Variablen vom Datentyp BOOL sind möglich.
5. Wenn mehrere Prozesse (z. B. I/O-Variable, Technologieobjekt, PROFIdrive-Telegramm) auf eine I/O-Adresse zugreifen, gilt:
   - Auf eine I/O-Adresse eines Ausgangs (Datentyp BYTE, WORD oder DWORD) kann nur ein einziger Prozess schreibend zugreifen.
     Lesender Zugriff mit einer I/O-Variablen auf einen Ausgang, der von einem anderen Prozess schreibend genutzt wird, ist möglich.
   - Alle Prozesse müssen mit dem gleichen Datentyp (BYTE, WORD, DWORD oder ARRAY dieser Datentypen) auf diese I/O-Adresse zugreifen. Zugriffe auf einzelne Bits sind unabhängig davon möglich.
     Achten Sie z. B. darauf, wenn Sie mit einer I/O-Variablen das zum Antrieb bzw. vom Antrieb übertragene PROFIdrive-Telegramm lesen wollen: Die Länge der I/O-Variablen muss der Länge des Telegramms entsprechen.
   - Schreibender Zugriff auf verschiedene Bits einer Adresse ist von mehreren Prozessen aus möglich; ein schreibender Zugriff mit den Datentypen BYTE, WORD oder DWORD ist dann nicht möglich.

Hinweis

Bei Zugriffen auf das feste Prozessabbild der BackgroundTask (Seite 273) gelten diesen Regeln nicht. Diese Zugriffe werden bei einer Konsistenzprüfung des Projekts (z. B. beim Download) nicht berücksichtigt.

6.3.3.3 I/O-Variable für Direktzugriff oder Prozessabbild der zyklischen Tasks erstellen

I/O-Variablen für Direktzugriff oder Prozessabbild der zyklischen Tasks erstellen Sie in der Adressliste der Detailanzeige.

Sie müssen sich hierzu im Offline-Modus befinden.

Hier in Kurzform die Vorgehensweise:

1. Wählen Sie in der Detailanzeige das Register "Adressliste" und wählen Sie das SIMOTION Gerät aus.
   oder
   Doppelklicken Sie im Projektnavigator des SIMOTION SCOUT auf das Element "ADRESSLISTE" im Teilbaum des SIMOTION Geräts.
2. Markieren Sie die Zeile, vor der Sie die I/O-Variable einfügen wollen, und wählen Sie im Kontextmenü **Neue Zeile einfügen**
   oder
   Scrollen Sie bis zum Ende der Variablentabelle (leere Zeile).
3. Geben Sie in die leere Zeile der Tabelle ein oder wählen Sie:
   - Namen der **I/O-Variablen**.
   - **I/O-Adresse**
     Wählen Sie die Einträge "IN" bzw. "OUT" aus, wenn Sie die I/O-Variabe (Eingang bzw. Ausgang) symbolisch zuordnen wollen (Ab Version V4.2 des SIMOTION Kernels. Die symbolische Zuordnung muss aktiviert sein, Menü **Projekt > Symbolische Zuordnung verwenden**.).
     Oder geben Sie eine feste Adresse ein gemäß "Syntax für Eingabe der I/O-Adressen" (Seite 270).
   - optional bei Ausgängen:
     Markieren Sie die Checkbox **Nur lesen**, wenn Sie auf den Ausgang nur lesend zugreifen wollen.
     Sie können dann einen Ausgang lesen, der bereits durch einen anderen Prozess beschrieben wird (z. B. Ausgang eines Nockens, PROFIdrive-Telegramm).
     Eine nur lesbare Ausgangsvariable kann dem Prozessabbild einer zyklischen Task nicht zugeordnet werden.
   - **Datentyp** der Variablen gemäß "Mögliche Datentypen der I/O-Variablen" (Seite 271).

4. Optional können Sie noch eingeben oder wählen (nicht bei Datentyp BOOL):
   - **Feldlänge** (Größe des Arrays)
   - **Prozessabbild** oder Direktzugriff:
     Die Zuordnung ist nur möglich, wenn die Checkbox **Nur lesen** deaktiviert ist.
     Für Prozessabbild wählen Sie die zyklische Task, der Sie die I/O-Variable zuordnen wollen. Damit Sie eine Task auswählen können, muss sie im Ablaufsystem aktiviert sein.
     Für Direktzugriff wählen Sie den leeren Eintrag.
   - **Strategie** für das Verhalten im Fehlerfall, siehe Funktionshandbuch SIMOTION Basisfunktionen.
   - **Anzeigeformat** (bei Arrays für jedes Element), wenn Sie die Variable in der Adressliste beobachten.
   - **Ersatzwert** (bei Arrays für jedes Element).
5. Nur wenn Sie als I/O-Adresse "IN" bzw. "OUT" gewählt haben (symbolische Zuordnung).
   - Klicken Sie in der Spalte **Zuordnung** auf die Schaltfläche **[…]**.
     Ein Fenster wird geöffnet, in dem die möglichen Zuordnungsziele des SIMOTION-Geräts und gegebenenfalls des SINAMICS Integrated angezeigt werden. Es werden nur die Zuordnungsziele angezeigt, die von der Datenrichtung (Eingang bzw. Ausgang) und dem Datentyp passen.
   - Wählen Sie das Zuordnungsziel aus.
     In der Spalte **Zuordnung-Status** wird angezeigt, ob die Zuordnung erfolgreich war.

   Zur symbolischen Zuordnung siehe Funktionshandbuch SIMOTION Basisfunktionen.

Sie können nun mit dem Adressliste oder jedem Programm des SIMOTION-Geräts auf diese Variable zugreifen.

Wie Sie mit der Adressliste umgehen, ist ausführlich in der Online-Hilfe beschrieben.

Hinweis

Beachten Sie beim Prozessabbild für zyklische Tasks:

- Eine Variable kann nur einer einzigen Task zugeordnet werden.
- Jedes Byte eines Ein- oder Ausgangs kann nur einer einzigen I/O-Variablen zugeordnet werden.

Beachten Sie beim Datentyp BOOL:

- Das Prozessabbild für zyklische Tasks und eine Strategie für den Fehlerfall können nicht definiert werden. Es gilt das Verhalten, das mittels einer I/O-Variablen für das gesamte Byte definiert wurde (Voreinstellung: Direktzugriff bzw. CPU-Stop).
- Zugriffe auf die einzelnen Bits einer I/O-Variablen sind auch mithilfe der Bitzugriffsfunktionen möglich.

Beachten Sie bei Änderungen innerhalb der I/O-Variablen (z. B. Einfügen und Löschen von I/O-Variablen, Namensänderungen, Adressänderungen):

- Die interne Adressierung anderer I/O-Variablen kann sich unter Umständen ändern; alle I/O-Variablen werden dadurch inkonsistent.
- Alle Programmquellen, die Zugriffe, auf I/O-Variablen enthalten, müssen gegebenenfalls neu übersetzt werden.

Hinweis

I/O-Variablen können Sie nur im Offline-Modus anlegen. Sie legen sie im SIMOTION SCOUT an und verwenden sie in Ihren Programmquellen (z. B. ST-Quellen, MCC-Quellen, KOP/FUP-Quellen).

Ausgänge können Sie lesen und beschreiben, Eingänge jedoch nur lesen.

Bevor Sie neue oder aktualisierte I/O-Variablen beobachten und steuern können, müssen Sie das Projekt ins Zielsystem laden.

Sie können eine I/O-Variable wie jede andere Variable verwenden, siehe "Auf I/O-Variablen zugreifen" (Seite 285).

6.3.3.4 Syntax für Eingabe der I/O-Adressen

Syntax

Für die Eingabe der I/O-Adresse bei der Definition einer I/O-Variablen für Direktzugriff oder Prozessabbild der zyklischen Tasks (Seite 263) verwenden Sie die nachstehende Syntax. Hierdurch spezifizieren Sie neben der Adresse zugleich den Datentyp des Zugriffs und die Art des Zugriffs (Eingang/Ausgang).

Tabelle 6-21 Syntax für Eingabe der I/O-Adressen für Direktzugriff oder Prozessabbild der zyklischen Tasks

Datentyp	Syntax für		Zulässiger Adressbereich					
	Eingang	Ausgang	Direktzugriff		Prozessabbild		z. B. Direktzugriff D435 V4.1	
BOOL	PIn.x	PQn.x	n: x:	0 .. *MaxAddr* 0 .. 7		-[1]	n: x:	0 .. 16383 0 .. 7
BYTE	PIBn	PQBn	n:	0 .. *MaxAddr*	n:	0 .. *MaxAddr*	n:	0 .. 16383
WORD	PIWn	PQWn	n:	0 .. 62 64 .. *MaxAddr* - 1	n:	0 .. 62 64 .. *MaxAddr* - 1	n:	0 .. 62 64 .. 16382
DWORD	PIDn	PQDn	n:	0 .. 60 64 .. *MaxAddr* - 3	n:	0 .. 60 64 .. *MaxAddr* - 3	n:	0 .. 60 64 .. 16380
n = logische Adresse x = Bitnummer								
MaxAddr =	Maximale I/O-Adresse des SIMOTION Geräts in Abhängigkeit von der Version des SIMOTION Kernels, siehe Adressbereich der SIMOTION-Geräte (Seite 266).							
[1] Für den Datentyp BOOL kann das Prozessabbild für zyklische Tasks nicht definiert werden. Es gilt das Verhalten, das mittels einer I/O-Variablen für das gesamte Byte definiert wurde (Voreinstellung: Direktzugriff).								

Beispiele

Eingang an logischer Adresse 1022, Datentyp WORD: **PIW1022**.

Ausgang an logischer Adresse 63, Bit 3, Datentyp BOOL: **PQ63.3**.

Hinweis

Beachten Sie die Regeln für I/O-Adressen für Direktzugriff und das Prozessabbild der zyklischen Tasks (Seite 267).

6.3.3.5 Mögliche Datentypen der I/O-Variablen

Den I/O-Variablen für Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263) können nachstehende Datentypen zugeordnet werden. Die Breite des Datentyps muss der Breite des Datentyps der I/O-Adresse entsprechen.

Wenn Sie der I/O-Variablen einen numerischen Datentyp zuordnen, können Sie auf diese Variable als Ganzzahl zugreifen.

Tabelle 6-22 Mögliche Datentypen der I/O Variablen für Direktzugriff und Prozessabbild der zyklischen Tasks

Datentyp der I/O-Adresse	Mögliche Datentypen der IO-Variablen
BOOL (PIn.x, PQn.x)	BOOL
BYTE (PIBn, PQBn)	BYTE, SINT, USINT
WORD (PIWn, PQWn)	WORD, INT, UINT
DWORD (PIDn, PQDn)	DWORD, DINT, UDINT

Zum Datentyp der I/O-Adresse siehe auch "Syntax für Eingabe der I/O-Adressen" (Seite 270).

6.3.3.6 Detaillierter Status der I/O-Variablen (ab Kernel V4.2)

Ab Version V4.2 des SIMOTION Kernels kann der Status einer I/O-Variablen mit *_quality.varname* abgefragt werden, um z. B. die Verfügbarkeit der I/O-Variablen zu prüfen. Er wird als ODER-Verknüpfung der nachfolgenden Statuswerte im Datentyp DWORD geliefert und kann z. B. einer entsprechenden Variablen zugewiesen werden. Der Wert 16#0000_0000 signalisiert, dass die angeschlossene Peripherie fehlerfrei arbeitet.

Für jede I/O-Variable innerhalb eines in HW-Konfig projektierten Adressbereichs (Slot bzw. Subslot) wird derselbe Wert geliefert.

Tabelle 6-23 Bedeutung der Statuswerte von I/O-Variablen

Wert (DWORD)	Bit x = 1	Bedeutung
16#0000_0000	-	Kein Fehler aufgetreten
16#0000_0001	0	**Wartungsanforderung** (Maintenance Required) Die angeschlossene Baugruppe meldet eine Wartungsanforderung. Innerhalb eines absehbaren Zeitraumes ist eine Überprüfung der Komponente erforderlich (z. B. die Druckerpatrone muss innerhalb eines Zeitraums von mehreren Tagen ausgetauscht werden).
16#0000_0002	1	**Wartungsbedarf** (Maintenance Demanded) Die angeschlossene Baugruppe meldet Wartungsbedarf. Innerhalb eines kurzen Zeitraumes ist eine Überprüfung der Komponente erforderlich (z. B. die Druckerpatrone muss sofort ausgetauscht werden).

Wert (DWORD)	Bit x = 1	Bedeutung
16#0000_0004	2	**Anstehende Warnung** (Antriebswarnung, TM17-Warnung, …) Die angeschlossene Baugruppe hat eine Warnung gemeldet. Dies wurde im Diagnosepuffer eingetragen. Die genaue Ursache ist der Dokumentation der betreffenden Baugruppe zu entnehmen.
16#0000_0008	3	**Anstehende Störung** (Diagnosealarm, Antriebsstörung, TM17-Störung, …) Die angeschlossene Baugruppe hat einen Fehler gemeldet. Dies wurde im Diagnosepuffer eingetragen. Die genaue Ursache ist der Dokumentation der betreffenden Baugruppe zu entnehmen.
16#0000_0010	4	**Eigene Parametrierung passt nicht zur Gegenseite** Beim Vergleich der eigenen Parametrierung mit der Parametrierung der angeschlossenen Baugruppe wurde ein Unterschied festgestellt. Die gewünschte Funktionalität ist damit nicht gewährleistet. Abhilfe: Projekt speichern und Änderungen übersetzen, beide Seiten neu laden.
16#0000_0020	5	**Applikation und Gegenseite sind nicht taktsynchron** (Fehler beim dynamischen Lebenszeichen) Bestimmte Telegramme (Achs-, Gleichlauf, Nocken-, Messtaster-Telegramm) werden über den Austausch zyklischer Lebenszeichen synchronisiert. Die Überprüfung des zyklischen Lebenszeichens liefert Fehler. Die Daten im Telegramm sind damit ungültig. Abhilfe: Synchronisation abwarten, Parametrierung überprüfen (z. B. passt der in HW-Konfig am Gerät eingestellte Master-Application-Cycle zum Servo-Takt), Projekt speichern und Änderungen übersetzen, beide Seiten neu laden.
16#0000_0040	6	**Peripherie kann nicht in allen Takten synchron benutzt werden** Schneller Applikationstakt (Servo_fast) und langsamer Applikationstakt (Servo) laufen asynchron zueinander. Die Peripherie kann nur in den zum Bustakt gehörenden Takten synchron benutzt werden. Ein Zugriff aus anderen Takten erfolgt asynchron und inkonsistent! Abhilfe: Funktion _synchroniseDpInterfaces() aufrufen.
16#0000_0080	7	**Peripherie kann nicht synchron benutzt werden** Die SIMOTION Steuerung ist Sync-Slave an einem Bus. Der Busanschluss läuft synchron zum Sync-Master aber noch nicht synchron zu den Applikationstakten der SIMOTION Steuerung. Ein Zugriff auf die Peripherie erfolgt asynchron und inkonsistent! Abhilfe: Funktion _synchroniseDpInterfaces() aufrufen.
16#0000_0100	8	**Busanschluss (Sync-Slave) ist nicht taktsynchron zum Sync-Master** Die SIMOTION Steuerung ist Sync-Slave an einem Bus und hat ihren Busanschluss noch nicht zum Sync-Master synchronisiert. Die taktsynchrone Peripherie an diesem Bus kann noch nicht genutzt werden. Abhilfe: Sync-Master anschalten/anstecken.
16#0000_0200	9	**DP-Station ist deaktiviert** Die Partner-Baugruppe wurde deaktiviert. Abhilfe: Partner-Baugruppe aktivieren (Funktion _activateDpSlave()).
16#0000_0400	10	**Der Partner von Eingängen (z. B. I-Device, I-Slave) ist im STOP** Die angeschlossene Baugruppe befindet sich im Betriebszustand STOP und sendet deshalb keine neuen Daten. Abhilfe: angeschlossene Baugruppe in RUN schalten.
16#0000_0800	11	**PROFINET: Ausfall durch Submodul erkannt** (z. B. Kanal-Fehler) Die Verbindung zu dem angeschlossenen Gerät ist in Ordnung. Der Fehler ist in dem angeschlossenen Gerät zu suchen. Fehlersuche: Diagnosepuffer, Gerätediagnose mit HW-Konfig.

Wert (DWORD)	Bit x = 1	Bedeutung
16#0000_1000	12	**PROFINET: Ausfall durch Modul erkannt** (z. B. Submodul ausgefallen, gezogen, …) Die Verbindung zu dem angeschlossenen Gerät ist in Ordnung. Der Fehler ist in dem angeschlossenen Gerät zu suchen. Fehlersuche: Diagnosepuffer, Gerätediagnose mit HW-Konfig.
16#0000_2000	13	**PROFINET: Ausfall durch Gerät erkannt** (z. B. Gerät im STOP, Modul gezogen, …) Die Verbindung zu dem angeschlossenen Gerät ist in Ordnung. Der Fehler ist in dem angeschlossenen Gerät zu suchen. Fehlersuche: Diagnosepuffer, Gerätediagnose mit HW-Konfig.
16#0000_4000	14	**PROFINET: Ausfall durch Controller erkannt** (z. B. nicht verbunden, …) Zu einem Partner am Profinet besteht keine Verbindung. Mögliche Ursache: Partner ist ausgeschaltet, Kabel gezogen, falsche Verbindungs-Parametrierung Fehlersuche: am besten mit Profinet-Topologieeditor in HW-Konfig
16#0000_8000	15	**Slot/Subslot ist nicht verbunden** (Ziehen-Alarm) Die Verbindung zu dem angeschlossenen Gerät ist in Ordnung. Der Fehler ist in dem angeschlossenen Gerät zu suchen (z. B. Modul/Submodul gezogen). Fehlersuche: Diagnosepuffer, Gerätediagnose mit HW-Konfig
16#0001_0000	16	**Gerät ist nicht verbunden** (Stationsausfall) Zu einem Partner besteht keine Verbindung. Mögliche Ursache: Partner ist ausgeschaltet, Kabel gezogen.
16#0002_0000	17	**Ersatzwertverhalten beim Zugriff** Es besteht keine Verbindung zur Gegenseite (Summensignal aus Bit 9 … 16), d. h. es liegen keine gültigen Eingangsdaten vor bzw. die Ausgangsdaten kommen nicht an der Klemme an. Beim Direktzugriff auf diese Adresse bzw. bei der Prozessabbildaktualisierung wirkt das eingestellte Ersatzwertverhalten (Ersatzwert, letzter Wert).
16#4000_0000	30	**Nur Diagnoseadresse** Für diese Adresse sind keine zyklischen I/O-Daten projektiert. Es können aber Diagnoseinformationen des Submoduls abgefragt werden.
16#8000_0000	31	**Adresslücke** Für diese logische Adresse ist keine Hardware projektiert.

6.3.4 Zugriffe auf festes Prozessabbild der BackgroundTask

Das feste Prozessabbild der BackgroundTask ist ein Speicherbereich im RAM des SIMOTION-Geräts, auf dem eine Untermenge des I/O-Adressraums des SIMOTION-Geräts gespiegelt ist. Es wird bevorzugt bei der Programmierung der BackgroundTask (zyklische Programmierung) eingesetzt, da es während deren gesamten Zyklus konsistent ist.

Die Größe des festen Prozessabbilds der BackgroundTask beträgt für alle SIMOTION-Geräte 64 Byte (Adressbereich 0 .. 63).

Hinweis

Mit dem festen Prozessabbild der BackgroundTask kann auf Adressen zugegriffen werden, die in der Peripherie nicht vorhanden oder in HW-Konfig nicht projektiert sind. Diese werden wie normale Speicheradressen behandelt.

Speicherbereich

- **Ab Version V4.2 des SIMOTION Kernels** kann durch eine Einstellung am Gerät (Seite 317) "Gemeinsames Prozessabbild" gewählt werden, dass der Speicherbereich für das feste Prozessabbild der BackgroundTask eine Untermenge des Speicherbereichs für das Prozessabbilds der zyklischen Tasks ist.
  I/O-Adressen können sowohl mit dem festen Prozessabbild der BackgroundTask als auch Prozessabbild der zyklischen Tasks gelesen und beschrieben werden.
- **Bis Version V4.1 des SIMOTION Kernels** bzw. bei der Einstellung am Gerät "Separates Prozessabbild" (ab Version V4.2 des SIMOTION Kernels) belegen das feste Prozessabbild der BackgroundTask und das Prozessabbild der zyklischen Tasks unterschiedliche Speicherbereiche.
  I/O-Adressen, auf die mit dem Prozessabbild der zyklischen Tasks zugegriffen wird, können mit dem festen Prozessabbild der BackgroundTask nicht gelesen oder beschrieben werden. Sie werden wie normale Speicheradressen behandelt.

Tabelle 6-24 Einfluss der Einstellungen "Gemeinsames Prozessabbild" bzw. "Separates Prozessabbild" auf das feste Prozessabbild der BackgroundTask

	Gemeinsames Prozessabbild	Separates Prozessabbild
Verfügbarkeit	Erst ab Version V4.2 des SIMOTION Kernels verfügbar: • Einstellung wählbar. • Standard für neu angelegte Geräte.	Bis Version V4.1 des SIMOTION Kernels gilt: • Systemeigenschaft, nicht konfigurierbar. Ab Version V4.2 des SIMOTION Kernels gilt: • Einstellung wählbar. • Standard beim Hochrüsten des Geräts.
Speicherbereich	Untermenge des Speicherbereichs für das Prozessabbild der zyklischen Tasks	Getrennter Speicherbereich zum Prozessabbild der zyklischen Tasks.
Verwendung von I/O-Adressen, auf die mit dem Prozessabbild der zyklischen Tasks zugegriffen wird	Möglich. Aktualisierung erfolgt mit der konfigurierten zyklischen Task.	Nicht möglich. Die Adressen werden wie normale Speicheradressen behandelt
Byte-Anordnung beim Bilden des Prozessabbild	Wie von der Peripherie geliefert.	Immer Big Endian.
Byte-Anordnung beim Zugriff auf das Prozessabbild	Immer Big Endian.	Immer Big Endian.

	Gemeinsames Prozessabbild	Separates Prozessabbild
Zugriff auf Peripherie, die in der Byte-Anordnung Little Endian arbeit	Gleiches Ergebnis wie beim Direktzugriff oder beim Prozessabbild der zyklischen Tasks (außer beim Datentyp WORD oder DWORD).	Abhängig von angelegten I/O-Variablen für Direktzugriff sind die Ergebnisse unterschiedlich.
Auswirkungen auf das Prozessabbild der zyklischen Tasks	Siehe entsprechende Tabelle in "Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263)".	
Weitere Informationen	Gemeinsames Prozessabbild (Seite 275).	Separates Prozessabbild (Seite 278).

Zur Anordnung der Bytes Little Endian und Big Endian: siehe Funktionshandbuch SIMOTION Basisfunktionen.

Eine Gegenüberstellung der wichtigsten Eigenschaften im Vergleich zum Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263) finden Sie in "Wichtige Eigenschaften von Direktzugriff und Prozessabbild (Seite 260)".

Hinweis

Die Regeln für I/O-Adressen für Direktzugriff und das Prozessabbild der zyklischen Tasks (Seite 267) gelten **nicht**. Die Zugriffe auf das feste Prozessabbild der BackgroundTask werden bei einer Konsistenzprüfung des Projekts (z. B. beim Download) nicht berücksichtigt.

In der Peripherie nicht vorhandene oder in HW Konfig nicht projektierte Adressen werden wie normale Speicheradressen behandelt.

Auf das feste Prozessabbild der BackgroundTask können Sie zugreifen:

- Mittels eines absoluten PA-Zugriffs (Seite 280): der Bezeichner des absoluten PA-Zugriffs enthält die Adresse des Ein-/Ausgangs sowie den Datentyp.
- Mittels eines symbolischen PA-Zugriffs (Seite 282): Sie deklarieren eine Variable, die auf den entsprechenden absoluten PA-Zugriff verweist:
  - eine Unit-Variable,
  - eine statische lokale Variable in einem Programm.
- Mittels einer I/O-Variablen (Seite 284): Sie definieren im Symbolbrowser eine geräteglobal gültige I/O-Variable, die auf den entsprechenden absoluten PA-Zugriff verweist.

6.3.4.1 Gemeinsames Prozessabbild (ab Kernel V4.2)

Ab Version V4.2 des SIMOTION Kernels kann am SIMOTION Gerät die Einstellung (Seite 317) "Gemeinsames Prozessabbild" ausgewählt werden. Dies bedeutet, dass die Adressen 0 .. 63 des Prozessabbilds der zyklischen Task und das feste Prozessabbild der BackgroundTask denselben Speicherbereich belegen.

Für SIMOTION Geräte ab Version V4.2, die im Projekt neu angelegt werden, ist dies die Voreinstellung.

Eigenschaft des gemeinsamen Prozessabbilds

1. Der Speicherbereich für das feste Prozessabbild der BackgroundTask (Seite 273) ist eine Untermenge des Speicherbereichs für das Prozessabbild der zyklischen Tasks (Seite 263).
2. Deshalb können I/O-Adressen, auf die bereits mit dem Prozessabbild der zyklischen Tasks zugegriffen wird, auch weiterhin für das feste Prozessabbild der BackgroundTask genutzt werden. Die Aktualisierung erfolgt allerdings mit der konfigurierten zyklischen Task.
3. Beim Bilden des festen Prozessabbilds der Backgroundtask gilt:
   Die Byte-Anordnung ist dieselbe, wie sie von Peripherie geliefert wird:
   - Big Endian, z. B. bei Peripherie über PROFIBUS DP, PROFINET, P-Bus, DRIVE-CLiQ
   - Little Endian, z. B. bei der Onboard-Peripherie der SIMOTION Geräte C240, C240 PN.

   Eine eventuell für die betreffenden Adressen angelegte I/O-Variable für Direktzugriff oder Prozessabbild der zyklischen Tasks hat keinen Einfluss auf die Byte-Anordnung.
4. Auf das feste Prozessabbild der BackgroundTask wird immer mit der Byte-Anordnung Big Endian zugegriffen.
5. Diese beiden letzten Eigenschaften (Nr. 3 und 4) wirken sich aus beim Zugriff auf Ein- und Ausgänge, die mit der Byte-Anordnung Little Endian arbeiten (z. B. Onboard-Peripherie der SIMOTION Geräte C240, C240 PN).
   Wenn mit dem festen Prozessabbild der BackgroundTask zugegriffen wird, führt dies zu folgendem Verhalten, unabhängig davon, ob für die betreffenden Adressen I/O-Variablen für Direktzugriff oder das Prozessabbild der zyklischen Tasks angelegt sind:
   - Der Zugriff auf die einzelnen Bytes liefert über eine I/O-Variable oder dem festen Prozessabbild der BackgroundTask immer das gleiche Ergebnis.
   - Nur beim Zugriff mit dem Datentyp WORD sind beim festen Prozessabbild der BackgroundTask die Bytes vertauscht.

   Siehe auch nachfolgendes Beispiel.

Zur Anordnung der Bytes Little Endian und Big Endian: siehe Funktionshandbuch SIMOTION Basisfunktionen.

Beispiel für das gemeinsame Prozessabbild: Zugriff auf eine Peripherie, die mit der Byte-Anordnung Litte Endian arbeitet

Die Digitaleingänge des SIMOTION Geräts C240 arbeiten mit der Byte-Anordnung Little Endian und belegen standardmäßig die Adressen 66 (Bit 0 ..7) und 67 (Bit 0.. 3). In HW-Konfig wird die Startadresse auf 60 geändert, damit sie im Bereich des festen Prozessabbilds der BackgroundTask liegt. Auf die Adressen 60 und 61 wird nun mit verschiedenen I/O-Variablen und dem Prozessabbild der BackgroundTask zugegriffen.

Es werden folgende drei Fälle betrachtet, die die unterscheiden, ob und welche I/O-Variablen für Direktzugriff oder das Prozessabbild der zyklischen Tasks angelegt sind:

1. Fall A:
   Für die Adressen 60 und 61 sind **keine I/O-Variablen** angelegt
2. Fall B:
   Für die Adressen 60 und 61 sind **zwei I/O-Variablen** mit Datentyp BYTE angelegt: io_byte_60 (PIB60) und io_byte_61 (PIB61).
3. Fall C:
   Für die Adresse 60 ist **eine I/O-Variable** mit Datentyp WORD angelegt, die auch die Adresse 61 umfasst: io_word_60 (PIW60).

Zusätzlich sind in jedem der drei Fälle zwei I/O-Variablen angelegt, die den Zugriff auf das Bit 3 ermöglichen: io_bit_60_3 (PI60.3) und io_bit_61_3 (PI61.3).

In der folgenden Tabelle ist aufgelistet, welche Werte sich bei den nachfolgenden Zugriffen ergeben:

- Direktzugriffe oder Zugriffe auf das Prozessabbild der zyklischen Task:
  - Zugriff auf die einzelnen Bytes oder das Wort mit der jeweiligen I/O-Variablen.
  - Zugriff auf jedes einzelne Byte mit der Funktion _getInOutByte (nur Direktzugriff).
  - Zugriff auf jeweilige Bit 3 mit der entsprechenden I/O-Variablen.
- Zugriff auf das feste Prozessabbild der BackgroundTask:
  - Zugriff auf die einzelnen Bytes mit Absolutbezeichner.
  - Zugriff auf das Wort mit Absolutbezeichner.
  - Zugriff auf jeweilige Bit 3 mit Absolutbezeichner.

Tabelle 6-25 Einstellung "Gemeinsames Prozessabbild" (ab Kernel V4.2): Verschiedene Zugriffe auf die Prozessabbilder eines Eingangs, der mit Byte-Anordnung Little Endian arbeitet

	Zugriff mit	**Fall A [1]**	**Fall B [1]**	**Fall C [1]**
Direktzugriff oder Zugriff auf das Prozessabbild der zyklischen Task	io_byte_60 (PIB60)	-	16#08	-
	io_byte_61 (PIB61)	-	16#00	-
	io_word_60 (PIW60)	-	-	16#0008
	_getInOutByte (IN, 60)	16#08	16#08	16#08
	_getInOutByte (IN, 61)	16#00	16#00	16#00
	io_bit_60_3 (PI60.3)	TRUE	TRUE	TRUE
	io_bit_61_3 (PI61.3)	FALSE	FALSE	FALSE

	Zugriff mit	Fall A [1]	Fall B [1]	Fall C [1]
Zugriff auf das feste Prozessabbild der BackgroundTask	%IB60	16#08	16#08	16#08
	%IB61	16#00	16#00	16#00
	%IW60	16#0800 [2]	16#0800 [2]	16#0800 [2]
	%I60.3	TRUE	TRUE	TRUE
	%I61.3	FALSE	FALSE	FALSE

[1] Die Fälle A, B oder C unterscheiden, ob und welche I/O-Variablen für den Direktzugriff oder das Prozessabbild der zyklischen Tasks angelegt sind, siehe Erklärung im Text.

[2] Die beiden Bytes im Wort sind vertauscht, da ein in der Byte-Anordnung Little Endian gespeicherter Wert mit Big Endian gelesen wird.

6.3.4.2 Separates Prozessabbild (bis Kernel V4.1)

Bis Version V4.1 des SIMOTION Kernels werden das Prozessabbild der zyklischen Task und das feste Prozessabbild der BackgroundTask in verschiedenen Speicherbereichen abgelegt (Separates Prozessabbild).

Ab Version V4.2 des SIMOTION Kernels kann am SIMOTION Gerät die Einstellung (Seite 317) "Separates Prozessabbild" ausgewählt werden. Diese Einstellung sichert die Kompatibilität zu früheren Kernelversionen.

Für SIMOTION Geräte, die auf eine Version ab V4.2 hochgerüstet werden, ist dies die Voreinstellung.

Eigenschaft des separaten Prozessabbilds

1. Das feste Prozessabbild der BackgroundTask (Seite 273) und das Prozessabbild der zyklischen Tasks (Seite 263) werden in verschiedenen Speicherbereichen abgelegt.
2. Deshalb können I/O-Adressen, auf die bereits mit dem Prozessabbild der zyklischen Tasks zugegriffen wird, nicht mit dem festen Prozessabbild der BackgroundTask gelesen oder beschrieben werden. Sie werden wie normale Speicheradressen behandelt.
3. I/O-Variablen für Direktzugriff beeinflussen das feste Prozessabbild der BackgroundTask:
   – Das feste Prozessabbild der BackgroundTask wird für die betreffenden Adressen immer in der Byte-Anordnung Big Endian gebildet.

4. Auf das feste Prozessabbild der BackgroundTask wird immer mit der Byte-Anordnung Big Endian zugegriffen.
5. Diese beiden letzten Eigenschaften (Nr. 3 und 4) wirken sich aus beim Zugriff auf Ein- und Ausgänge, die mit der Byte-Anordnung Little Endian arbeiten (z. B. Onboard-Peripherie der SIMOTION Geräte C230-2, C240, C240 PN).
   Wenn für die betreffenden Adressen eine I/O-Variable für Direktzugriff mit dem Datentyp WORD angelegt ist und mit dem festen Prozessabbild der BackgroundTask zugegriffen wird, führt dies zu folgendem Verhalten:
   - Der Zugriff mit Datentyp WORD liefert über die I/O-Variable und das feste Prozessabbild der BackgroundTask das gleiche Ergebnis.
   - Der Zugriff auf die einzelnen Bytes mit der Funktion _getInOutByte (siehe Funktionshandbuch SIMOTION Basisfunktionen) liefert diese in der Anordnung Little Endian.
   - Der Zugriff auf die einzelnen Bytes oder Bits mit dem festen Prozessabbild der BackgroundTask liefert diese in der Anordnung Big Endian.

   Siehe auch nachfolgendes Beispiel.

Zur Anordnung der Bytes Little Endian und Big Endian: siehe Funktionshandbuch SIMOTION Basisfunktionen.

Beispiel für das separate Prozessabbild: Zugriff auf eine Peripherie, die mit der Byte-Anordnung Litte Endian arbeitet

Die Digitaleingänge des SIMOTION Geräts C240 arbeiten mit der Byte-Anordnung Little Endian und belegen standardmäßig die Adressen 66 (Bit 0 ..7) und 67 (Bit 0.. 3). In HW-Konfig wird die Startadresse auf 60 geändert, damit sie im Bereich des festen Prozessabbilds der BackgroundTask liegt. Auf die Adressen 60 und 61 wird nun mit verschiedenen I/O-Variablen und dem Prozessabbild der BackgroundTask zugegriffen.

Es werden folgende drei Fälle betrachtet, die unterscheiden, ob und welche I/O-Variablen für Direktzugriff angelegt sind:

1. Fall A:
   Für die Adressen 60 und 61 sind **keine I/O-Variablen** angelegt
2. Fall B:
   Für die Adressen 60 und 61 sind **zwei I/O-Variablen** mit Datentyp BYTE angelegt: io_byte_60 (PIB60) und io_byte_61 (PIB61).
3. Fall C:
   Für die Adresse 60 ist **eine I/O-Variable** mit Datentyp WORD angelegt, die auch die Adresse 61 umfasst: io_word_60 (PIW60).

Zusätzlich sind in jedem der drei Fälle zwei I/O-Variablen angelegt, die den Zugriff auf das Bit 3 ermöglichen: io_bit_60_3 (PI60.3) und io_bit_61_3 (PI61.3).

In der folgenden Tabelle ist aufgelistet, welche Werte sich bei den nachfolgenden Zugriffen ergeben:

- Direktzugriffe:
  - Zugriff auf die einzelnen Bytes oder das Wort mit der jeweiligen I/O-Variablen.
  - Zugriff auf jedes einzelne Byte mit der Funktion _getInOutByte (Seite 263).
  - Zugriff auf jeweilige Bit 3 mit der entsprechenden I/O-Variablen.
- Zugriffe auf das feste Prozessabbild der BackgroundTask:
  - Zugriff auf die einzelnen Bytes mit Absolutbezeichner.
  - Zugriff auf das Wort mit Absolutbezeichner.
  - Zugriff auf jeweilige Bit 3 mit Absolutbezeichner.

Tabelle 6-26 Einstellung "Separates Prozessabbild" oder Kernel bis Version V4.1: Verschiedene Zugriffe auf die Prozessabbilder eines Eingangs, der mit Byte-Anordnung Little Endian arbeitet

	Zugriff mit	**Fall A [1]**	**Fall B [1]**	**Fall C [1]**
Direktzugriff	io_byte_60 (PIB60)	-	16#08	-
	io_byte_61 (PIB61)	-	16#00	-
	io_word_60 (PIW60)	-	-	16#0008
	_getInOutByte (IN, 60)	16#08	16#08	16#08
	_getInOutByte (IN, 61)	16#00	16#00	16#00
	io_bit_60_3 (PI60.3)	TRUE	TRUE	TRUE
	io_bit_61_3 (PI61.3)	FALSE	FALSE	FALSE
Zugriff auf das feste Prozessabbild der BackgroundTask	%IB60	16#08	16#08	**16#00** [3]
	%IB61	16#00	16#00	**16#08** [3]
	%IW60	**16#0800** [2]	**16#0800** [2]	16#0008
	%I60.3	TRUE	TRUE	**FALSE** [3]
	%I61.3	FALSE	FALSE	**TRUE** [3]

[1] Die Fälle A, B oder C unterscheiden, ob und welche I/O-Variablen für den Direktzugriff angelegt sind, siehe Erklärung im Text.

[2] Die beiden Bytes im Wort sind vertauscht, da ein in der Anordnung Little Endian gespeicherter Wert mit Big Endian gelesen wird.

[3] Die beiden benachbarten Bytes sind vertauscht, da das entsprechende Wort in der Anordnung Big Endian gespeichert ist.

6.3.4.3 Absoluter Zugriff auf das feste Prozessabbild der BackgroundTask (Absoluter PA-Zugriff)

Sie greifen absolut auf das feste Prozessabbild der BackgroundTask (Seite 273) zu, indem Sie direkt den Bezeichner für die Adresse (mit implizitem Datentyp) verwenden. Die Syntax des Bezeichners (Seite 281) ist im nachfolgenden Abschnitt beschrieben.

Sie können den Bezeichner für den absoluten PA-Zugriff wie eine normale Variable verwenden (Seite 281).

Hinweis

Ausgänge können Sie lesen und beschreiben, Eingänge nur lesen.

6.3.4.4 Syntax für den Bezeichner für einen absoluten PA-Zugriff

Für den absoluten Zugriff auf das feste Prozessabbild der BackgroundTask (Seite 280) verwenden Sie nachstehende Syntax. Hierdurch spezifizieren Sie neben der Adresse zugleich den Datentyp des Zugriffs und die Art des Zugriffs (Eingang/Ausgang).

Diese Bezeichner verwenden Sie auch:

- bei der Deklaration eines symbolischen Zugriffs auf das feste Prozessabbild der BackgroundTask (Seite 282).
- beim Erstellen einer I/O-Variablen für den Zugriff auf das feste Prozessabbild der BackgroundTask (Seite 284).

Tabelle 6-27 Syntax für den Bezeichner für einen absoluten PA-Zugriff

Datentyp	Syntax für		Zulässiger Adressbereich	
	Eingang	Ausgang		
BOOL	%In.x oder %IXn.x [1]	%Qn.x oder %QXn.x [1]	n: x:	0 .. 63 [2] 0 .. 7
BYTE	%IBn	%QBn	n:	0 .. 63 [2]
WORD	%IWn	%QWn	n:	0 .. 63 [2]
DWORD	%IDn	%QDn	n:	0 .. 63 [2]
n = logische Adresse x = Bitnummer				
[1] Die Syntax %IXn.x bzw. %QXn.x ist bei der Definition von I/O-Variablen nicht zulässig. [2] Bei separatem Prozessabbild (Seite 278) gilt: Keine Adressen, die beim Prozessabbild der zyklischen Tasks verwendet werden. Siehe Hinweis unten.				

Beispiele

Eingang an logischer Adresse 62, Datentyp WORD: **%IW62**.

Ausgang an logischer Adresse 63, Bit 3, Datentyp BOOL: **%Q63.3**.

Hinweis

Bis Version V4.1 des SIMOTION Kernels bzw. bei der Einstellung am Gerät "Separates Prozessabbild" (Seite 278) (ab Version V4.2 des SIMOTION Kernels) gilt:

- Adressen, auf die mit dem Prozessabbild der zyklischen Tasks zugegriffen wird, können mit dem festen Prozessabbild der BackgroundTask nicht gelesen oder beschrieben werden.

Ab Version V4.2 des SIMOTION Kernels und bei der Einstellung am Gerät "Gemeinsames Prozessabbild" (Seite 275) gilt diese Einschränkung nicht.

Hinweis

Die Regeln für I/O-Adressen für Direktzugriff und das Prozessabbild der zyklischen Tasks (Seite 267) gelten **nicht**. Die Zugriffe auf das feste Prozessabbild der BackgroundTask werden bei einer Konsistenzprüfung des Projekts (z. B. beim Download) nicht berücksichtigt.

In der Peripherie nicht vorhandene oder in HW Konfig nicht projektierte Adressen werden wie normale Speicheradressen behandelt.

Es folgen einige Beispiele für die Zuweisung typgleicher Variablen:

Tabelle 6-28 Beispiele für den absoluten CPU-Speicherzugriff

```
status1 := %I1.1; // Datentyp BOOL
status2 := %IB10; // Datentyp BYTE
status3 := %IW20; // Datentyp WORD
status4 := %ID20; // Datentyp DWORD

%Q1.1 := status1; // Datentyp BOOL
%QB20 := status2; // Datentyp BYTE
%QW20 := status3; // Datentyp WORD
%QD20 := status4; // Datentyp DWORD
```

6.3.4.5 Symbolischer Zugriff auf das feste Prozessabbild der BackgroundTask (Symbolischer PA-Zugriff)

Sie können auf das feste Prozessabbild der BackgroundTask (Seite 273) symbolisch zugreifen und müssen nicht immer den absoluten PA-Zugriff angeben.

Sie können einen symbolischen Zugriff deklarieren:

- als statische Variable eines Programms (innerhalb der Struktur VAR / END_VAR im Deklarationsabschnitt),
- als Unit-Variable (innerhalb der Struktur VAR_GLOBAL / END_VAR im Interface- oder Implementationsabschnitt der ST-Quelle).

Die Syntax für die Deklaration eines symbolischen Namens für den Prozessabbild-Zugriff ist im Bild angegeben:

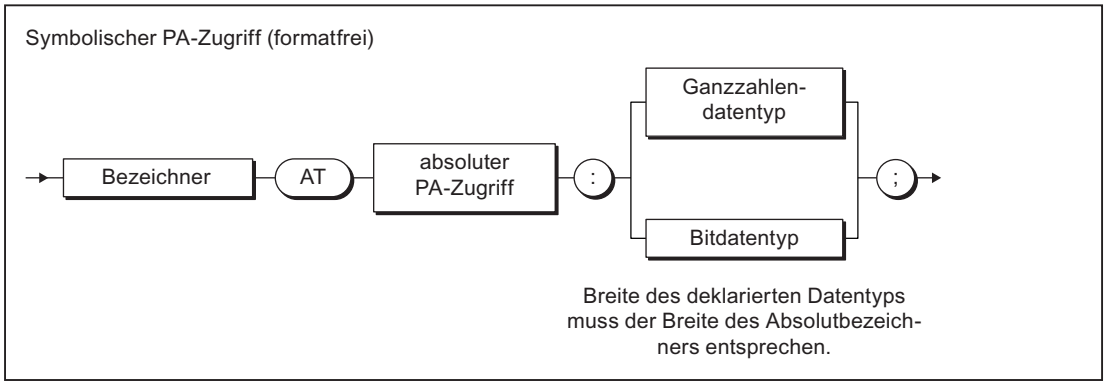


Bild 6-6 Deklaration eines symbolischen Zugriffs auf das Prozessabbild

Zum absoluten PA-Zugriff siehe "Syntax für den Bezeichner für einen absoluten PA-Zugriff (Seite 281)".

Die Breite des deklarierten Ganzzahlen oder Bitdatentyps muss der Breite des absoluten PA-Zugriffs entsprechen, siehe "Mögliche Datentypen des symbolischen PA-Zugriffs (Seite 283)". Nach Deklaration eines numerischen Datentyps können Sie den Inhalt des Prozessabbilds als Ganzzahl ansprechen.

Siehe auch Beispiel für die Deklaration (Seite 284).

6.3.4.6 Mögliche Datentypen des symbolischen PA-Zugriffs

In folgenden Fällen kann dem festen Prozessabbild der BackgroundTask (Seite 273) ein Datentyp zugewiesen werde, der vom Datentyp des absoluten PA-Zugriffs abweicht. Die Breite des Datentyps muss der Breite des Datentyps des absoluten PA-Zugriffs entsprechen.

- bei der Deklaration eines symbolischen PA-Zugriffs (Seite 282).
- beim Erstellen einer I/O-Variablen (Seite 284).

Wenn Sie dem symbolischen PA-Zugriff bzw. der I/O-Variablen einen numerischen Datentyp zuordnen, können Sie auf diese Variable als Ganzzahl zugreifen.

Tabelle 6-29 Mögliche Datentypen des symbolischen PA-Zugriffs

Datentyp des absoluten PA-Zugriffs	Mögliche Datentypen des symbolischen PA-Zugriffs
BOOL (%In.x, %IXn.x, %Qn.x. %QXn.x)	BOOL
BYTE (%IBn, %QBn)	BYTE, SINT, USINT
WORD (%IWn, %QWn)	WORD, INT, UINT
DWORD (%IDn, %PQDn)	DWORD, DINT, UDINT

Zum Datentyp des absoluten PA-Zugriffs siehe auch "Syntax für den Bezeichner für einen absoluten PA-Zugriff (Seite 281)".

6.3.4.7 Beispiel für symbolischen PA-Zugriff

Sie wollen beispielsweise auf den CPU-Speicherbereich (absoluter PA-Zugriff (Seite 281)) %IB10 zugreifen, aber flexibel auf Änderungen in Ihrem Programm reagieren können. Dann deklarieren Sie eine Variable *myInput* mit diesem CPU-Speicherbereich folgendermaßen:

```
VAR
    myInput AT %IB10 : BYTE;
END_VAR
```

Wenn Sie mit dem ganzzahligen Wert des Speicherbereiches arbeiten wollen, deklarieren Sie die Variable *myInput* folgendermaßen:

```
VAR
    myInput AT %IB10 : SINT;
END_VAR
```

Wenn Sie später einen anderen CPU-Speicherbereich als %IB10 in Ihrem Programm verwenden wollen, müssen Sie den absoluten PA-Zugriff nur in der Variablendeklaration ändern.

6.3.4.8 I/O-Variable für Zugriff auf das feste Prozessabbild der BackgroundTask erstellen

I/O-Variablen für den Zugriff auf das feste Prozessabbild der BackgroundTask erstellen Sie im Symbol-Browser der Detailanzeige; Sie müssen sich hierzu im Offline-Modus befinden.

Hier in Kurzform die Vorgehensweise:

1. Wählen Sie in der Detailanzeige das Register "Adressliste" und wählen Sie das SIMOTION Gerät aus.
   oder
   Doppelklicken Sie im Projektnavigator des SIMOTION SCOUT auf das Element "ADRESSLISTE" im Teilbaum des SIMOTION Geräts.
2. Markieren Sie die Zeile, vor der Sie die I/O-Variable einfügen wollen, und wählen Sie im Kontextmenü Neue Zeile einfügen
   oder
   Scrollen Sie bis zum Ende der Variablentabelle (leere Zeile).
3. Wählen Sie in der Detailanzeige das Register Symbolbrowser und scrollen Sie bis zum Ende der Variablentabelle (leere Zeile).
4. Geben Sie in die leere Zeile der Tabelle ein oder wählen Sie:
   - **Name** der Variablen.
   - Unter **I/O-Adresse** den absoluten PA-Zugriff gemäß "Syntax für den Bezeichner für einen absoluten PA-Zugriff" (Seite 281)
     (Ausnahme: Die Syntax %IXn.x bzw. %QXn.x für den Datentyp BOOL ist nicht zulässig).
   - **Datentyp** der I/O-Variablen gemäß "Mögliche Datentypen des symbolischen PA-Zugriffs" (Seite 283).
5. Wählen Sie optional das Anzeigeformat, wenn Sie die Variable im Symbol-Browser beobachten.

Sie können nun mit der Adressliste oder jedem Programm des SIMOTION Geräts auf diese Variable zugreifen.

Hinweis

I/O-Variablen können Sie nur im Offline-Modus anlegen. Sie legen sie im SIMOTION SCOUT an und verwenden sie in Ihren Programmquellen.

Beachten Sie, dass Sie Ausgänge lesen und beschreiben, Eingänge jedoch nur lesen dürfen.

Bevor Sie neue oder aktualisierte I/O-Variablen beobachten und steuern können, müssen Sie das Projekt ins Zielsystem laden.

Sie können eine I/O-Variable wie jede andere Variable verwenden, siehe "Auf I/O-Variablen zugreifen" (Seite 285).

6.3.5 Auf I/O-Variablen zugreifen

Sie haben eine I/O-Variable erstellt für:

- Direktzugriff oder Prozessabbild der zyklischen Tasks (Seite 263).
- Zugriff auf das feste Prozessabbild der BackgroundTask (Seite 273).

Sie können diese I/O-Variable wie jede andere Variable verwenden.

Hinweis

Die Konsistenz ist nur für elementare Datentypen gewährleistet.

Bei der Verwendung von Arrays hat der Anwender selbst für die Konsistenz der Daten zu sorgen.

Hinweis

Wenn Sie Unit-Variablen oder lokale Variablen gleichen Namens (z. B. *var-name*) deklariert haben, spezifizieren Sie die I/O-Variable mit *_device.var-name* (vordefinierter Namensraum, siehe Tabelle "Vordefinierte Namensräume" in "Namensräume").

Der Direktzugriff auf eine I/O-Variable, die Sie als Prozessabbild einer zyklischen Task erstellt haben, ist möglich: Spezifizieren Sie den Direktzugriff mit *_direct.var-name* bzw. *_device._direct.var-name*.

Wenn Sie bei Fehlern bei Variablenzugriffen vom voreingestellten Verhalten abweichen wollen, können Sie die Funktionen _getSafeValue und _setSafeValue (siehe Funktionshandbuch *SIMOTION Basisfunktionen*) verwenden.

Zu Fehlern beim Zugriff auf I/O-Variablen siehe Funktionshandbuch *SIMOTION Basisfunktionen*.

6.4 Bibliotheken verwenden

Bibliotheken stellen Ihnen anwenderdefinierte Datentypen, Funktionen und Funktionsbausteine zur Verfügung, die Sie von allen SIMOTION Geräten aus nutzen können.

Bibliotheken können in allen Programmiersprachen geschrieben werden; sie können in allen Programmquellen (z. B. ST-Quellen, MCC-Quellen) verwendet werden.

Nähere Einzelheiten zum Einfügen und Verwalten von Bibliotheken erhalten Sie in der Online-Hilfe.

Hinweis

Für die Bibliotheksnamen gelten dieselben Regeln wie für die Namen von Programmquellen, siehe ST-Quelle einfügen (Seite 29). Insbesondere ist die zulässige Länge der Namen abhängig von der Version des SIMOTION Kernels:

- Ab Version V4.1 des SIMOTION Kernels: maximal 128 Zeichen.
- Bis Version V4.0 des SIMOTION Kernels: maximal 8 Zeichen.

Bei Versionen des SIMOTION Kernels bis V4.0 wird eine Überschreitung der zulässigen Länge des Bibliotheksnamens unter Umständen erst bei einer Konsistenzprüfung oder beim Download des Projekts erkannt!

Optional kann eine Bibliothek auch Programme zur Verfügung stellen, die in von anderen Programmen oder Funktionsbausteinen aufgerufen werden können. Beachten Sie notwendigen Voraussetzungen zum Aufruf eines "program in program" (Seite 206). Die statischen Daten des aufgerufenen Programms werden jeweils einmalig im Anwenderspeicher des Geräts abgelegt, auf dem das Bibliotheksprogramm aufgerufen wird. Bei jedem Aufruf des Programms auf demselben Gerät werden dieselben Programminstanzdaten verwendet. Die Zuordnung eines Bibliotheksprogramms zum Ablaufsystem ist nicht möglich.

6.4.1 Bibliothek übersetzen

In Bibliotheken können alle ST-Befehle, mit Ausnahme der in der Tabelle angegebenen, verwendet werden. Darüber hinaus sind auch einige Variablenzugriffe nicht erlaubt, auch diese sind in der Tabelle vermerkt.

Tabelle 6-30 In Bibliotheken nicht erlaubte ST-Befehle und Variablenzugriffe

Nicht erlaubte Befehle: • Funktion *_getTaskId* (siehe Funktionshandbuch *SIMOTION Basisfunktionen*) • Funktion *_getAlarmId* (siehe Funktionshandbuch *SIMOTION Basisfunktionen*) • Funktion *_checkEqualTask* (siehe Funktionshandbuch *SIMOTION Basisfunktionen*) • Sofern die Bibliothek geräteunabhängig übersetzt wird (d. h. ohne Bezug zu einem SIMOTION Gerät und einer Version des SIMOTION Kernels): – Systemfunktionen der SIMOTION Geräte (siehe Listenhandbuch der SIMOTION Geräte) – Versionsabhängige Systemfunktionen
Nicht erlaubte Variablenzugriffe: • Unit-Variablen (remanente und nicht remanente) • Geräteglobale Variablen (remanente und nicht remanente) • I/O-Variablen • Instanzen der Technologieobjekte und deren Systemvariablen • Variablen der Tasknamen und der projektierten Meldungen (Namensräume *_task* und *_alarm*, siehe Namensräume (Seite 294), Tabelle *Vordefinierte Namensräume*) • Sofern die Bibliothek geräteunabhängig übersetzt wird (d. h. ohne Bezug zu einem SIMOTION Gerät und einer Version des SIMOTION Kernels): – Systemvariablen der SIMOTION Geräte (siehe Listenhandbuch der SIMOTION Geräte) – Konfigurationsdaten der Technologieobjekte (siehe Listenhandbuch der Konfigurationsdaten des jeweiligen SIMOTION Technologiepakets)

Hinweis

In Bibliotheken steht Ihnen die Debug-Funktion **Programm Status** nicht zur Verfügung.

Einzelne Bibliothek übersetzen

So übersetzen Sie eine einzelne Bibliothek:

1. Markieren Sie die Bibliothek im Projektnavigator.
2. Wählen Sie Menü **Bearbeiten > Objekteigenschaften**.
3. Wählen Sie Register **TPs/TOs**.
4. Wählen Sie die SIMOTION Geräte (mit Version des SIMOTION Kernels) und die Technologiepakete aus, bezüglich derer die Bibliothek übersetzt werden soll, siehe Funktionshandbuch *SIMOTION Basisfunktionen*.
5. Wählen Sie das Kontextmenü **Übernehmen und Übersetzen**.

Die Bibliothek wird bezüglich **aller** ausgewählten SIMOTION Geräte, Versionen des SIMOTION Kernels und Technologiepakete (bzw. geräteunabhängig) übersetzt.

Hinweis

Beachten Sie, wenn die zu übersetzende Bibliothek eine andere importiert:

1. An der importierten Bibliothek müssen mindestens dieselben Geräte und Versionen des SIMOTION Kernels ausgewählt werden wie an der importierenden Bibliothek.
   Alternativ kann die importierte Bibliothek geräteunabhängig übersetzt werden, wenn die Voraussetzungen hierfür vorliegen, siehe Funktionshandbuch *SIMOTION Basisfunktionen*.
2. Die importierte Bibliothek muss bezüglich aller konfigurierten Geräte, Kernelversionen und Technologiepakete bereits einzeln übersetzt sein.
   Die Übersetzung der Bibliothek im Rahmen einer projektweiten Übersetzung ist in der Regel nicht ausreichend.

Übersetzen einer Bibliothek im Rahmen einer projektweiten Übersetzung

Beim Übersetzen des gesamten SIMOTION Projekts (z. B. mit Menü **Projekt > Speichern und alles neu übersetzen** oder beim XML-Import) werden auch die verwendeten Bibliotheken übersetzt.

Hinweis

Beachten Sie beim projektweiten Übersetzen:

1. Abhängigkeiten zwischen Bibliotheken werden automatisch erkannt und die entsprechende Übersetzungsreihenfolge gewählt.
2. Eine Bibliothek wird **nur** bezüglich derjenigen SIMOTION Geräte (einschließlich Versionen des SIMOTION Kernels) übersetzt, die im Projekt konfiguriert sind und die Bibliothek verwenden.
3. Andere SIMOTION Geräte und Kernelversionen, die an der Bibliothek eingestellt sind, werden ignoriert.

6.4.2 Know-how-Schutz für Bibliotheken

Sie können Bibliotheken oder deren Quellen gegen unbefugten Zugriff Dritter schützen. Geschützte Bibliotheken oder Quellen können Sie dann nur unter Angabe des Passworts öffnen oder als Klartextdateien exportieren.

Sie können:

- Die einzelnen Quellen einer Bibliothek mit Know-how-Schutz versehen:
  Es sind nur die Quellen vor unbefugtem Zugriff geschützt.
  Die Einstellung der SIMOTION Geräte einschließlich der Versionen des SIMOTION Kernels und der Technologiepakete, für welche die Bibliothek übersetzt werden soll, kann weiterhin vom Anwender geändert und angepasst werden. Siehe Funktionshandbuch *SIMOTION Basisfunktionen*.
  Der Anwender kann somit die Bibliothek für andere SIMOTION Geräte und Kernelversionen einsetzen.
- Die Bibliothek mit Know-how-Schutz versehen:
  Es sind dann vor unbefugten Zugriff geschützt:
  - Alle Quellen der Bibliothek
  - Die Einstellung der SIMOTION Geräte einschließlich der Versionen des SIMOTION Kernels und der Technologiepakete, für welche die Bibliothek übersetzt werden soll,

  Sie verhindern dadurch, dass der Anwender die Bibliothek für andere SIMOTION Geräte und Kernelversionen einsetzen kann.
  Verwenden Sie diese Einstellung nur, wenn dies beabsichtigt ist.

Weitere Informationen zum Know-how-Schutz finden Sie in der SIMOTION Online-Hilfe.

Hinweis

Beim Export im XML-Format werden die Bibliotheken oder Quellen verschlüsselt exportiert. Beim Import der verschlüsselten XML-Dateien bleibt der Know-how-Schutz einschließlich Login und Passwort erhalten.

6.4.3 Datentypen, Funktionen und Funktionsbausteine aus Bibliotheken verwenden

Bevor Sie Datentypen, Funktionen oder Funktionsbausteine aus Bibliotheken verwenden, müssen Sie diese der ST-Quelle bekannt machen. Hierzu verwenden Sie im Interfaceabschnitt der ST-Quelle folgendes Konstrukt:

```
USELIB library-name [AS namespace];
```

Dabei ist *library-name* der Name der Bibliothek, wie er im Projektnavigator angezeigt wird.

Mehrere Bibliotheken geben Sie als durch Kommata getrennte Liste an, z. B.:

```
USELIB library-name_1 [AS namespace_1],
    library-name_2 [AS namespace_2],
    library-name_3 [AS namespace_1] , ...
```

Mit dem optionalen Zusatz *AS namespace* können Sie einen Namensraum (Seite 294) definieren.

- Sie können dann auch auf Datentypen, Funktionen und Funktionsbausteine der Bibliothek zugreifen, die namensgleich mit einer solchen ST-Quelle des SIMOTION Geräts (im Ordner PROGRAMME) sind.
- Sie können damit auch Namensgleichheiten zwischen Datentypen, Funktionen, und Funktionsbausteinen der Bibliotheken beseitigen.

Sie können verschiedenen Bibliotheken den gleichen Namensraum zuweisen.

Tabelle 6-31 Beispiel zur Verwendung von Namensräumen bei Bibliotheken

```
INTERFACE
    USELIB Bib_1 AS NS_1, Bib_2 AS NS_2;
    PROGRAM Main_Program;
END_INTERFACE

IMPLEMENTATION
    FUNCTION Function1 : VOID
        VAR
            ComID : CommandIdType;
        END_VAR
        ComId := _getCommandId();
    END_FUNCTION

    PROGRAM Main_program
        function1();         // Funktion aus dieser Quelle
        NS_1.Var1:=1;
        NS_2.Var1:=2;
        NS_1.function1();    // Funktion aus Bibliothek Bib1
        NS_2.function1();    // Funktion aus Bibliothek Bib2
    END_PROGRAM
END_IMPLEMENTATION
```

6.5 Verwendung von gleichen Bezeichnern, Namensräume

6.5.1 Verwendung von gleichen Bezeichnern

Allgemeines

Die Verwendung von Unit-Variablen und lokalen Variablen (Programmvariablen, FB-Variablen, FC-Variablen) mit gleichem Namen ist möglich. Der Compiler sucht beim Übersetzen einer Programmquelle die Bezeichner beginnend mit der aktuellen POE. Der kleinere Gültigkeitsbereich hat immer Priorität vor dem größeren.

Sie können somit gleiche Bezeichner in verschiedenen Quelldatei-Abschnitten verwenden, sofern die nachstehenden Regeln eingehalten werden. Wenn durch einen Bezeichner in einer Unit oder POE ein übergeordneter Bezeichner verdeckt wird, gibt der Compiler eine Warnung aus.

Hinweis

Der Compiler kann unter bestimmten Umständen keine Warnung absetzen, wenn z. B. das zugehörige Technologiepaket nicht importiert ist.

Bezeichner in einer Programmorganisationseinheit (POE)

Alle folgenden Bezeichner in einer POE müssen eindeutig sein:

- Lokale Variablen der POE.
- Lokale Datentypen der POE.

Sie dürfen auch nicht mit nachstehenden Bezeichnern identisch sein:

- Reservierte Bezeichner.
- Bezeichner der POE selbst.

Der Compiler gibt eine Warnung aus, wenn folgende Bezeichner verdeckt werden:

- Unit-Variablen, Datentypen und POE derselben oder importierter Units.
- Standardsystemfunktionen, Standardsystemfunktionsbausteine und zugehörige Datentypen.
- Systemfunktionen und Systemdatentypen des SIMOTION Geräts.
- Programmorganisationseinheiten (POE) und Datentypen aus importierten Bibliotheken.
  - Abhilfe durch Angabe eines anwenderdefinierten Namensraums möglich.
- Systemfunktionen und Systemdatentypen aus importierten Technologiepaketen.
  - Abhilfe durch Angabe eines anwenderdefinierten Namensraums möglich.

- Variablen des SIMOTION Geräts (Systemvariablen, I/O-Variablen, geräteglobale Variablen).
  - Abhilfe durch Angabe des vordefinierten Namensraums *_device* möglich.
- Am SIMOTION Gerät konfigurierte Technologieobjekte
  - Abhilfe durch Angabe des vordefinierten Namensraums *_to* möglich.

Bezeichner in einer Unit

Exportierte Bezeichner alle Units (Unit-Variablen, Datentypen und POE) müssen geräteweit eindeutig sein.

Innerhalb einer Unit müssen alle folgenden Bezeichner eindeutig sein:

- Unit-Variablen (im Interface- oder Implementationsabschnitt deklariert).
- Datentypen (im Interface- oder Implementationsabschnitt deklariert).
- Programmorganisationseinheiten (POE).

Sie dürfen auch nicht mit nachstehenden Bezeichnern identisch sein:

- Reservierte Bezeichner.
- Unit-Variablen, Datentypen und POE importierter Units.
- Standardsystemfunktionen, Standardsystemfunktionsbausteine und zugehörige Datentypen.
- Systemfunktionen und Systemdatentypen des SIMOTION Geräts.
- Programmorganisationseinheiten (POE) und Datentypen aus importierten Bibliotheken.
  - Abhilfe durch Angabe eines anwenderdefinierten Namensraums möglich.
- Systemfunktionen und Systemdatentypen aus importierten Technologiepaketen.
  - Abhilfe durch Angabe eines anwenderdefinierten Namensraums möglich.

Der Compiler gibt eine Warnung aus, wenn folgende Bezeichner verdeckt werden:

- Variablen des SIMOTION Geräts (Systemvariablen, I/O-Variablen, geräteglobale Variablen).
  - Abhilfe durch Angabe des vordefinierten Namensraums *_device* möglich.
- Am SIMOTION Gerät konfigurierte Technologieobjekte.
  - Abhilfe durch Angabe des vordefinierten Namensraums *_to* möglich.

Bezeichner am SIMOTION Gerät (z. B. I/O-Variablen, geräteglobale Variablen)

Alle folgenden Bezeichner am SIMOTION Gerät müssen eindeutig sein:

- I/O-Variablen
- Geräteglobale Variablen
- Systemvariablen des SIMOTION Geräts
- Systemfunktionen und Systemdatentypen des SIMOTION Geräts.

Sie dürfen auch nicht mit nachstehenden Bezeichnern identisch sein:

- Reservierte Bezeichner.
- Standardsystemfunktionen, Standardsystemfunktionsbausteine und zugehörige Datentypen.

Beispiel

Das nachstehende Beispiel veranschaulicht diesen Sachverhalt. Es zeigt, dass bei gleicher Bezeichnung von Unit-Variablen (großer Gültigkeitsbereich) und FC-Variablen (kleiner Gültigkeitsbereich) die in der Funktion deklarierten Variablen nur in diesem Quelldatei-Abschnitt gültig sind. Die Unit-Variablen sind nur in den POE gültig, in denen keine lokalen Variablen gleichen Namens deklariert wurden. Siehe hierzu das Beispiel.

Beispiel für die Gültigkeit von Bezeichnern

```
TYPE
    type_a : (enum1, enum2, enum3);
END_TYPE

VAR_GLOBAL
    var_a, var_b : DINT; // Unit-Variablen
END_VAR

FUNCTION fc_1 : VOID
    VAR
        var_a : type_a;  // Deklaration verdeckt Unit-Variable
        var_c : DINT;    // Lokale Variable
    END_VAR
    // zulässige Anweisungen
    var_a := enum2;      // Zugriff auf lokale Variable
    var_b := 100;        // Zugriff auf Unit-Variable
    var_c := -1;         // Zugriff auf lokale Variable
    // unzulässige Anweisung
    // var_a := 200;
END_FUNCTION

FUNCTION fc_2 : VOID
    VAR
        var_b : type_a;  // Deklaration verdeckt Unit-Variable
        var_c : type_a;  // Lokale Variable
    END_VAR
    // zulässige Anweisungen
    var_a := -100;       // Zugriff auf Unit-Variable
    var_b := enum3;      // Zugriff auf lokale Variable
    var_c := enum1;      // Zugriff auf lokale Variable
    // unzulässige Anweisung
    // var_b := 200;
END_FUNCTION
```

6.5.2 Namensräume

Auch auf die außerhalb einer Programmquelle (z. B. in Bibliotheken, in Technologiepaketen, am SIMOTION Gerät) definierten Datentypen, Variablen, Funktionen oder Funktionsbausteinen können Sie mit deren Namen zugreifen.

Der Compiler sucht beim Übersetzen einer Programmquelle die Bezeichner beginnend mit der aktuellen POE. Die in einer Programmquelle deklarierten Datentypen, Variablen, Funktionen oder Funktionsbausteine verdecken deshalb namensgleiche Bezeichner, die außerhalb der Quelle definiert wurden, siehe Verwendung von gleichen Bezeichnern (Seite 291). Um dennoch diese Bezeichner zu erreichen, können Sie in bestimmten Fällen Namensräume verwenden.

Anwenderdefinierter Namensraum

Sie können in der Importanweisung für Bibliotheken und Technologiepakete Namensräume definieren, um Datentypen, Funktionen oder Funktionsbausteine dieser Bibliotheken und Technologiepaketen erreichen zu können.

```
USELIB library-name_1 [AS lib_namespace_1],
    library-name_2 [AS lib_namespace_2],
    library-name_3 [AS lib_namespace_1], ...

USEPACKAGE tp-name_1 [AS tp_namespace_1],
    tp-name_2 [AS tp_namespace_2],
    tp-name_3 [AS tp_namespace_1], ...
```

Auch Namensgleichheiten innerhalb verschiedener Bibliotheken können Sie dadurch beseitigen.

Wenn Sie eine Funktion, einen Funktionsbaustein oder einen Datentyp aus einer Bibliothek oder einem Technologiepaket verwenden wollen, setzen Sie die Bezeichnung des Namensraums durch Punkt getrennt vor den entsprechenden Namen, z. B. *namespace.fc-name*, *namespace.fb-name*, *namespace.type-name*.

Beispiel

Das folgende Beispiel zeigt, wie Sie das Technologiepaket Cam wählen, ihm den Namensraum Cam1 zuordnen und den Namensraum anwenden:

Beispiel für die Wahl eines Technologiepaketes und Verwendung eines Namensraums

```
INTERFACE
    USEPACKAGE Cam AS Cam1;
    USES ST_2;
    FUNCTION function1;
END_INTERFACE

IMPLEMENTATION
    FUNCTION function1 : VOID
    VAR_INPUT
        p_Axis : posAxis;
    END_VAR
    VAR
        retVal : DINT;
    END_VAR

    retVal:= Cam1._enableAxis (
            axis := p_Axis,
            nextCommand := Cam1.WHEN_COMMAND_DONE,
            commandId := _getCommandId() );
    END_FUNCTION
END_IMPLEMENTATION
```

Hinweis

Falls ein Namensraum für eine importierte Bibliothek oder Technologiepaket definiert ist, muss dieser **immer** angegeben werden, wenn eine Funktion, Funktionsbaustein oder Datentyp aus dieser Bibliothek oder diesem Technologiepaket verwendet wird. Siehe in obigem Beispiel: Cam1._enableAxis, Cam1.WHEN_COMMAND_DONE.

Vordefinierter Namensraum

Für geräte- und projektspezifische Variablen sowie für die Variablen der TaskId und AlarmId sind Namensräume vordefiniert. Setzen Sie ggf. deren Bezeichnung durch Punkt getrennt vor den Variablennamen, z. B. *_device.var-name* oder *_task.task-name*.

Tabelle 6-32 Vordefinierte Namensräume

Namensraum	Beschreibung
_alarm	Für AlarmId: Die Variable _alarm.*name* enthält die AlarmId der Meldung mit dem Bezeichner *name* (siehe Funktionshandbuch SIMOTION Basisfunktionen).
_device	Für gerätespezifische Variablen (geräteglobale Variablen, I/O-Variablen, Systemvariablen des SIMOTION Geräts).
_direct	Für Direktzugriff auf I/O-Variablen – siehe Direktzugriff und Prozessabbild der zyklischen Tasks (Seite 263). Lokaler Namensraum zu _device. Die Schachtelung _device._direct.*name* ist zulässig.

Namensraum	Beschreibung
_project	Für Namen der SIMOTION Geräte im Projekt; nur in Verwendung mit Technologieobjekten auf anderen Geräten. Bei projektweit eindeutigen Namen der Technologieobjekte auch für diese und deren Systemvariablen.
_task	Für TaskId: Die Variable _task.*name* enthält die TaskId der Task mit dem Bezeichner *name* (siehe Funktionshandbuch SIMOTION Basisfunktionen).
_quality	Ab Version V4.2 des SIMOTION Kernels: Für den detaillierten Status von I/O-Variablen (Seite 271). Ein Wert mit Datentyp DWORD wird geliefert. Lokaler Namensraum zu _device. Die Schachtelung _device._quality.*name* ist zulässig.
_to	Für Technologieobjekte, die am SIMOTION Gerät konfiguriert sind, sowie deren Systemvariablen und Konfigurationsdaten. Nicht für Systemfunktionen und Datentypen der Technologieobjekte. Hier verwenden Sie gegebenenfalls den anwenderdefinierten Namensraum für das importierte Technologiepaket

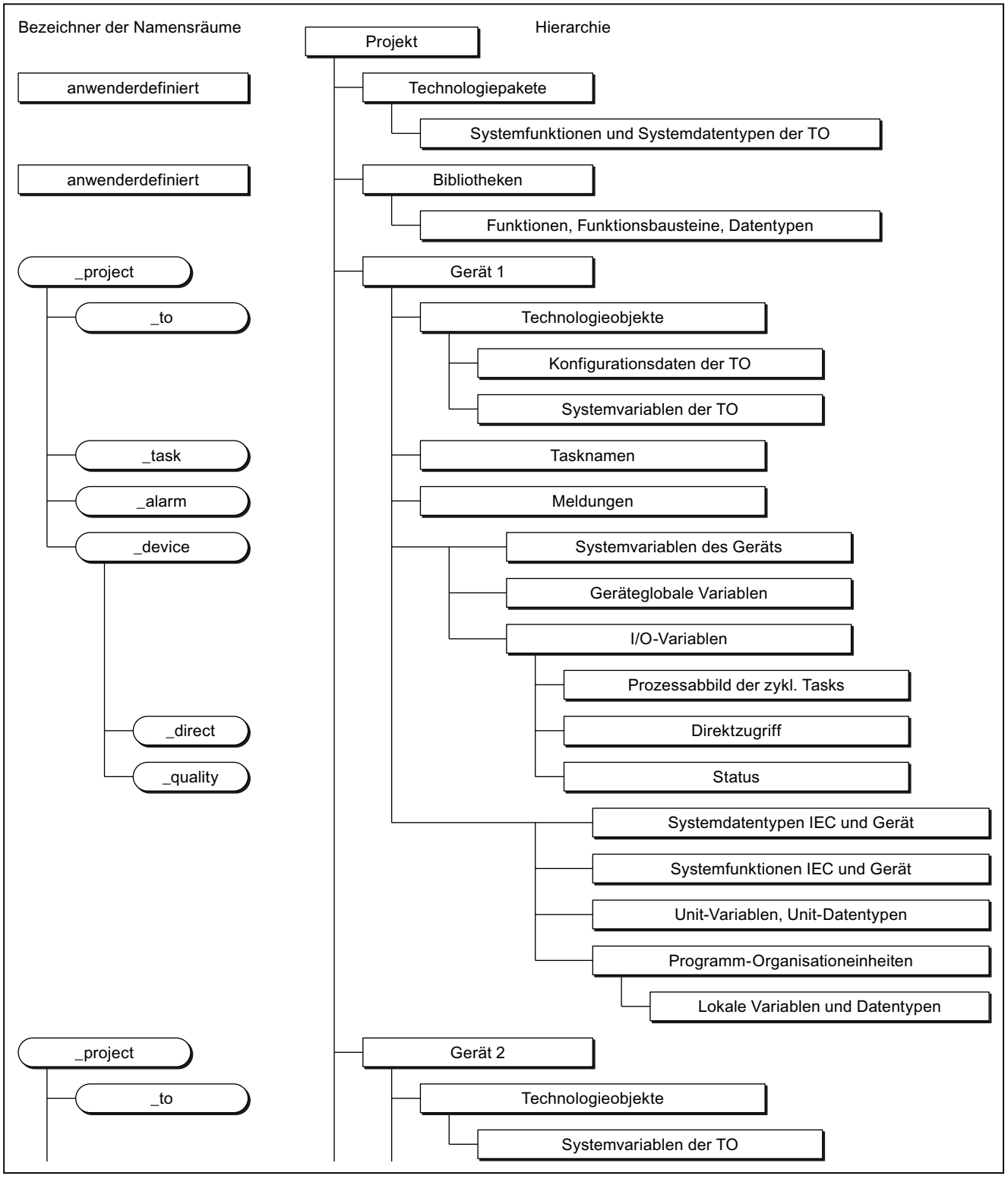


Bild 6-7 Namensräume und Hierarchie der Bezeichner

6.6 Referenzdaten

Mithilfe der Referenzdaten erhalten Sie einen Überblick:

- über verwendete Bezeichner mit Angabe Ihrer Deklaration und Verwendung (Querverweisliste (Seite 298)).
- über Funktionsaufrufe und deren Schachtelung (Programmstruktur (Seite 302))
- über den Speicherbedarf verschiedener Datenbereiche der Programmquellen (Codeattribute (Seite 304))

6.6.1 Querverweisliste

In der Querverweisliste sehen Sie alle Bezeichner in Programmquellen (z. B. ST-Quellen, MCC-Quellen):

- die als Variablen, als Datentypen oder als Programmorganisationseinheiten (Programm, Funktion, Funktionsbaustein) deklariert sind,
- die als vorher definierte Datentypen in Deklarationen verwendet werden,
- die als Variablen im Anweisungsabschnitt einer Programmorganisationseinheit verwendet werden.

6.6.1.1 Querverweisliste erzeugen und aktualisieren

Die Querverweisliste wird initial beim Öffnen automatisch erzeugt. Öffnen Sie eine Querverweisliste z. B. nach Auswahl einer ST-Quelle oder Bibliothek über das Menü **Bearbeiten > Referenzdaten anzeigen > Querverweise**. Nach Änderungen erfolgt die Aktualisierung zum Teil automatisch und kann immer auch manuell in der Detailanzeige über den Button  angestoßen werden. Bei der Aktualisierung über den Button wird auch geprüft, ob eine Übersetzung erforderlich ist. Falls eine Übersetzung erforderlich ist wird das mit einem gelben Dreiecksymbol neben dem Button signalisiert.

Aktualisierung der Querverweisliste bei Auswahl eines einzelnen Programms

Für das ausgewählte Programm wird die Liste bei jedem Öffnen automatisch aktualisiert. Die lokal definierten Bezeichner sind damit aktuell.

Externe Bezeichner in dem Programm werden beim Öffnen nicht aktualisiert.

Beim Übersetzen des Programms wird auch die geöffnete Querverweisliste automatisch aktualisiert.

Aktualisierung der Querverweisliste bei Auswahl eines Baums (CPU, Projekt, Bibliothek, Programmordner)

Die Querverweislliste wird beim erstmaligen Öffnen automatisch erstellt. Beim erneuten Öffnen erfolgt keine automatische Aktualisierung.

Hinweis

Für eine korrekte, konsistente Anzeige der Referenzdaten ist eine fehlerfreie Übersetzung Voraussetzung. Übersetzen Sie ggf. das Projekt, die CPU, das Programm oder die Bibliothek vorher.

6.6.1.2 Inhalt der Querverweisliste

Die Querverweisliste enthält alle Bezeichner, die dem ausgewählten Element des Projektnavigators zugeordnet sind, sowie deren Verwendungen in Form einer Tabelle:

Die Handhabung der Querverweisliste ist im Abschnitt "Mit der Querverweisliste umgehen (Seite 301)" beschrieben.

Tabelle 6-33 Bedeutung der Spalten und ausgewählter Einträge in der Querverweisliste

Spalte	Eintrag in Spalte	Bedeutung
Name		Name des Bezeichners
Typ		Typ des Bezeichners
	Name	• Datentyp einer Variable (z. B. REAL, INT) • Typ einer POE (z. B. PROGRAM, FUNCTION).
	DERIVED	Abgeleiteter Datentyp
	DERIVED ANY_OBJECT	TO-Datentyp
	ARRAY …	Datentyp ARRAY
	ENUM …	Aufzählungsdatentyp (Enumerator)
	STRUCT …	Datentyp STRUCT
Deklaration		Ort der Deklaration
	Name (Unit)	Deklaration in der Programmquelle *Name*
	Name (LIB)	Deklaration in der Bibliothek *Name*
	Name (TO)	Systemvariable des Technologieobjekts *Name*
	Name (TP)	Deklaration in der angegebenen Standardbibliothek: • Technologiepaket *Name* • std_fct = IEC-Bibliothek • device = gerätespezifische Bibliothek
	Name (DV)	Deklaration am SIMOTION-Gerät *Name* (z. B. I/O-Variable oder geräteglobale Variable)
	_project	Deklaration im Projekt (z. B. Technologieobjekt)
	_device	Interne Variable am SIMOTION-Gerät (z. B. Taskstartinfo)
	_task	Task im Ablaufsystem
Verwendung		Verwendung des Bezeichners
	CALL	Aufruf als Unterprogramm

Spalte	Eintrag in Spalte	Bedeutung
	ENUM *Name*	Als Element bei der Deklaration des Aufzählungsdatentyps *Name*.
	I/O	Deklaration als I/O-Variable
	R	Lesender Zugriff
	R (TYPE)	Als Datentyp in einer Deklaration
	R/W	Lesender und schreibender Zugriff
	STRUCT *Name*	Als Komponente bei der Deklaration der Struktur *Name*.
	TYPE	Deklaration als Datentyp oder POE
	Variablentyp (z. B. VAR, VAR_GLOBAL)	Deklaration als Variable des angegebenen Variablentyps
	W	Schreibender Zugriff
Pfadangabe		Pfadangabe des SIMOTION Geräts bzw. der Programmquelle
	Name	SIMOTION-Gerät *Name*
	Name1 / *Name2*	• Programmquelle *Name2* an SIMOTION Gerät *Name1* • Programmquelle *Name2* in Bibliothek *Name1*
	Name / taskbind.hid	Ablaufsystem des SIMOTION Geräts *Name*
Bereich		Bereich innerhalb des SIMOTION Geräts oder der Programmquelle
	IMPLEMENTATION	Implementationsabschnitt der Programmquelle
	INTERFACE	Inferfaceabschnitt der Programmquelle
	POE-TypName (z. B. FUNCTION *Name*, PROGRAM *Name*)	Programmorganisationseinheit (POE) *Name* innerhalb der Programmquelle. • bei einem MCC-Chart: zusätzlich laufende Nummerierung des Befehls (Blocknummer) • bei einem KOP/FUP-Programm: zusätzlich laufende Nummerierung des Netzwerks
	I/O-Adresse	I/O-Variable
	TASK *Name*	Zuordnung zur Task *Name*
	_device	Geräteglobale Variable
Sprache		Programmiersprache der Programmquelle
Zeile/Block		• Bei einer ST-Quelle: Zeilennummer innerhalb der Programmquelle • Bei einer MCC- bzw. KOP/FUP-Quelle: Relative Zeilennummmer innerhalb des Befehls (Blocks) bzw. Netzwerks. **Hinweis** Die absolute Zeilennumer innerhalb der Programmquelle, die Sie z. B. für die Tracefunktion "Trigger an Codestelle" benötigen, erhalten Sie folgendermaßen: – Drücken Sie den Button **Programmzeile ermitteln**. – Im Dialogfenster drücken Sie den Button **Programmzeile in die Zwischenablage kopieren**.

Hinweis

Diagnosefunktionen Einzelschrittverfolgung und Leuchtspur bei MCC-Programmierung

Bei diesen Diagnosefunktionen werden zusätzliche Variablen und Funktionen angelegt oder verwendet:

- Bei der Diagnosefunktion Einzelschrittverfolgung werden die Variablen TSI#dwuser_1 und TSI#dwuser_2 der Taskstartinfo verwendet.
- Bei der Diagnosefunktion Leuchtspur werden verschiedene interne Funktionen und Variablen, deren Bezeichner mit einem Unterstrich beginnen, automatisch vom Compiler angelegt. Außerdem wird die Variable TSI#currentTaskId der Taskstartinfo verwendet.

Bei eingeschalteter Diagnosefunktion werden diese Variablen und Funktionen für die Steuerung der Diagnosefunktion verwendet. Im Anwenderprogramm dürfen diese Variablen und Funktionen nicht verwendet werden.

6.6.1.3 Mit der Querverweisliste umgehen

Sie können in der Querverweisliste:

- Innerhalb der Spalten alphabetisch sortieren:
  - Klicken Sie hierzu in den Kopf der jeweiligen Spalte.
- Nach einem Bezeichner oder Eintrag suchen:
  - Klicken Sie auf den Button "Suchen" und geben Sie den Suchbegriff ein.
- Die angezeigten Bezeichner und Einträge filtern (Seite 302).
- Den Inhalt in die Zwischenablage kopieren, um Sie z. B. in ein Tabellenkalkulationsprogramm zu übernehmen:
  - Markieren Sie die entsprechenden Zeilen und Spalten.
  - Drücken Sie die Tastenkombination STRG+C.
- Den Inhalt ausdrucken (Menü **Projekt > Drucken**).
- Die referenzierte Programmquelle öffnen und den Cursor auf die zugehörige Zeile der ST-Quelle (bzw. MCC-Befehl oder KOP/FUP-Element) positionieren:
  - Doppelklicken Sie hierzu auf die entsprechende Zeile der Querverweisliste.
    oder
  - Setzen Sie den Cursor in die entsprechende Zeile der Querverweisliste und klicken Sie auf den Button "Gehe zur Verwendung".

Näheres zur Handhabung der Querverweisliste finden Sie in der Online-Hilfe.

6.6.1.4 Querverweisliste filtern

Sie können die Einträge in der Querverweisliste filtern, um nur die Anzeige auf relevante Einträge einzuschränken:

1. Klicken Sie auf den Button "Filtereinstellungen".
   Das Fenster "Filtereinstellungen für Querverweise" wird angezeigt.
2. Aktivieren Sie Checkbox "Filter aktiv".
3. Wenn Sie auch Systemvariablen und Systemfunktionen anzeigen wollen:
   – Deaktivieren Sie die Checkbox "Nur anwenderdefinierte Variablen anzeigen".
4. Setzen Sie für die entsprechenden Spalten das gewünschte Filterkriterium:
   – Wählen Sie den entsprechenden Eintrag aus der Auswahlliste oder geben Sie das Kriterium ein.
   – Wenn Sie nach einer Zeichenfolge innerhalb eines Eintrags suchen wollen: Deaktivieren Sie die Checkbox "Nur ganze Wörter".
5. Bestätigen Sie mit **OK**.

Die Querverweisliste wird gemäß den ausgewählten Filtereinstellungen angezeigt.

Hinweis

Nach Erzeugen der Querverweisliste ist ein Filter standardmäßig aktiviert.

6.6.2 Programmstruktur

In der Programmstruktur sehen Sie alle Funktionsaufrufe und deren Schachtelung innerhalb eines ausgewählten Elements.

Sie können die Programmstruktur selektiv anzeigen für:

- eine einzelne Programmquelle (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle)
- alle Programmquellen eines SIMOTION Geräts
- alle Programmquellen und Bibliotheken des Projekts
- Bibliotheken (alle Bibliotheken, einzelne Bibliothek, einzelne Programmquelle innerhalb einer Bibliothek)

So gehen Sie vor:

1. Markieren Sie im Projektnavigator das entsprechende Element, dessen Programmstruktur Sie anzeigen lassen wollen.
2. Wählen Sie Menü **Bearbeiten > Referenzdaten anzeigen > Programmstruktur**.
   An Stelle des Registers Querverweise wird nun das Register Programmstruktur in der Detailanzeige eingeblendet.

Hinweis

Bei jedem Öffnen der Programmstruktur werden die Anzeigedaten aktualisiert.

Mit der Taste F5 können Sie die Detailanzeige einer geöffneten Programmstruktur aktualisieren.

6.6.2.1 Inhalt der Programmstruktur

Angezeigt werden in einer Baumstruktur:

- als Wurzel jeweils
  - die Programmorganisationseinheiten (Programme, Funktionen, Funktionsbausteine), die in der Programmquelle deklariert sind, oder
  - die verwendeten Tasks des Ablaufsystems
- darunter die in dieser Programmorganisationseinheit bzw. Task referenzierten Unterprogramme.

Zur Struktur der Einträge siehe Tabelle:

Tabelle 6-34 Elemente der Anzeige für die Programmstruktur

Element	Beschreibung
Wurzel (deklarierte POE bzw. verwendete Task))	Durch Komma getrennte Liste • Bezeichner der Programmorganisationseinheit (POE) bzw. der Task • Bezeichner der Programmquelle, in der die POE bzw. die Task deklariert wurde, mit Zusatz [UNIT] • Minimaler und maximaler Stackbedarf (Speicherbedarf der POE bzw. der Task auf dem Lokaldatenstack) in Byte [Min, Max] • Minimaler und maximaler Gesamtstackbedarf (Speicherbedarf der POE bzw. der Task auf dem Lokaldatenstack einschließlich aller gerufenen POE) in Byte [Min, Max]
referenzierte POE	Durch Komma getrennte Liste: • Bezeichner der gerufenen POE • optional: Bezeichner der Programmquelle bzw. des Technologiepakets, in der die POE deklariert wurde: Zusatz (UNIT): anwenderdefinierte Programmquelle Zusatz (LIB): Bibliothek Zusatz (TP): Systemfunktion aus Technologiepaket • nur bei Funktionsbausteinen: Bezeichner der Instanz • nur bei Funktionsbausteinen: Bezeichner der Programmquelle, in der die Instanz deklariert wurde: Zusatz (UNIT): anwenderdefinierte Programmquelle Zusatz (LIB): Bibliothek • Zeile der (generierten) Quelle, in der die POE gerufen wird; mehrere Zeilen werden durch "|" getrennt.

6.6.3 Codeattribute

Unter Codeattribute erhalten Sie Informationen über den Speicherbedarf verschiedener Datenbereiche der Programmquellen.

Sie können die Codeattribute selektiv anzeigen für:

- eine einzelne Programmquelle (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle)
- alle Programmquellen eines SIMOTION Geräts
- alle Programmquellen und Bibliotheken des Projekts
- Bibliotheken (alle Bibliotheken, einzelne Bibliothek, einzelne Programmquelle innerhalb einer Bibliothek)

So gehen Sie vor:

1. Markieren Sie im Projektnavigator das entsprechende Element, dessen Codeattribute Sie anzeigen lassen wollen.
2. Wählen Sie **Menü Bearbeiten > Referenzdaten anzeigen > Codeattribute**.
   An Stelle des Registers **Querverweise** wird nun das Register **Codeattribute** in der Detailanzeige eingeblendet.

Hinweis

Bei jedem Öffnen der Codeattribute werden die Anzeigedaten aktualisiert.

Mit der Taste F5 können Sie die Detailanzeige der geöffneten Codeattribute aktualisieren.

6.6.3.1 Inhalt der Codeattribute

In einer Tabelle werden für alle ausgewählten Programmquellen angezeigt:

- der Bezeichner der Programmquelle,
- der Speicherbedarf in Byte für folgende Datenbereiche der Programmquelle:
  - **Dynamische Daten**: alle Unit-Variablen (remanente und nicht remanente, im Interface- und Implementationsabschnitt),
  - **Retain-Daten**: remanente Unit-Variablen im Interface- und Implementationsabschnitt,
  - **Interfacedaten**: Unit-Variablen (remanente und nicht remanente) im Interfaceabschnitt,
- die **Codegröße** bei der letzten Übersetzung in Byte,
- die **Anzahl der referenzierten Quellen**:
  Es wird die maximale Anzahl der angebundenen Quellen angezeigt (einschließlich Systembibliotheken), unabhängig davon, ob sie später ins Zielsystem geladen werden.

6.6.4 Referenz an Variablen

Wenn Sie im geöffneten Editorfenster zur jeweiligen Programmiersprache den Bezeichner einer Variable ausgewählt haben, können Sie über das Kontextmenü weitere Verwendungsstellen dieser Variablen auflisten oder zu springen.

Den Bezeichner der Variablen wählen Sie aus:

- bei SIMOTION ST: im Editorfenster einer ST-Quelle.
- bei SIMOTION MCC: im Eingabefeld in der Parametermaske eines geöffneten MCC-Befehls innerhalb eines MCC-Charts
- bei SIMOTION KOP/FUP: im Editorfenster eines KOP/FUP-Programms

Unter folgenden Voarussetzungen wird ein Bezeichner als Variable erkannt:

1. Der Bezeichner ist als Variable deklariert. Der Gültigkeitsbereich der Variablen schließt das jeweilige Fenster (ST-Quelle, MCC-Chart, KOP/FUP-Programm) ein.
2. Die Programmquelle ist übersetzt.
3. Die Variable ist folgendermaßen ausgewählt (innerhalb eines MCC-Chart in einer geöffneten Parametermaske) :
   - Der Bezeichner ist vollständig markiert
     oder
   - Der Cursor befindet sich innerhalb des Bezeichners.

Hinweis

Bei Feldern und Strukturen kann nur die Variable, nicht ein einzelnes Element ausgewählt werden.

Über das Kontextmenü **Gehe zu** stehen ihn folgende Möglichkeiten zur Verfügung:

- zur nächsten lokalen Verwendungsstelle springen:
  Wählen Sie das Kontextmenü **Gehe zu > Lokale Verwendung >>**.
  Die nächste Verwendungsstelle der Variablen innerhalb desselben Editorfensters (ST-Quelle, MCC-Chart, KOP/FUP-Programm) wird markiert. Bei einem MCC-Chart wird der entsprechende MCC-Befehl geöffnet.
- zur vorherigen lokalen Verwendungsstelle springen:
  Wählen Sie das Kontextmenü **Gehe zu > Lokale Verwendung <<**.
  Die vorherige Verwendungsstelle der Variablen innerhalb desselben Editorfensters (ST-Quelle, MCC-Chart, KOP/FUP-Programm) wird markiert. Bei einem MCC-Chart wird der entsprechende MCC-Befehl geöffnet.

- zur Deklarationsstelle spingen:
  Wählen Sie das Kontextmenü **Gehe zu > Deklarationsstelle**.
  Die Deklarationstelle der Variablen wird markeiert. Die entsprechende Programmquelle wird gegebenenfalls geöffnet.
- alle Verwendungsstellen auflisten
  Wählen Sie das Kontextmenü **Gehe zu > Verwendungstellen ….**
  In der Detailanzeige werden alle Verwendungsstellen der Variablen innerhalb deren Gültigkeitsbereichs (einschließlich der Deklarationsstelle) aufgelistet Der Aufbau dieser Liste ähnelt der Querverweisliste (Seite 299).
  So springen Sie zur gewünschten Verwendungsstelle:
  - Doppelklicken Sie auf die entsprechende Zeile.
    oder
  - Setzen Sie den Cursor in die entsprechende Zeile und klicken Sie auf den Button "Gehe zur Verwendung".

6.7 Präprozessor und Compiler durch Pragmas steuern

Ein Pragma wird verwendet, um in eine ST-Quelle Text (z. B. Anweisungen) einzufügen, der das Übersetzen der ST-Quelle beeinflusst.

Pragmas sind in geschweiften Klammern { und } eingeschlossen und können enthalten (siehe Bild):

- Präprozessor-Anweisungen zum Steuern des Präprozessors, siehe Präprozessor steuern (Seite 307).
  Die in einer ST-Quelle enthaltenen Pragmas mit Präprozessor-Anweisungen werden vom Präprozessor ausgewertet und als Steuerungsanweisungen interpretiert.
- Attribute für Compileroptionen, um den Compiler zu steuern, (siehe Compiler mit Attributen steuern (Seite 312).
  Die in einer ST-Quelle enthaltenen Pragmas mit Attributen für Compileroptionen werden vom Compiler ausgewertet und als Steuerungsanweisungen interpretiert.

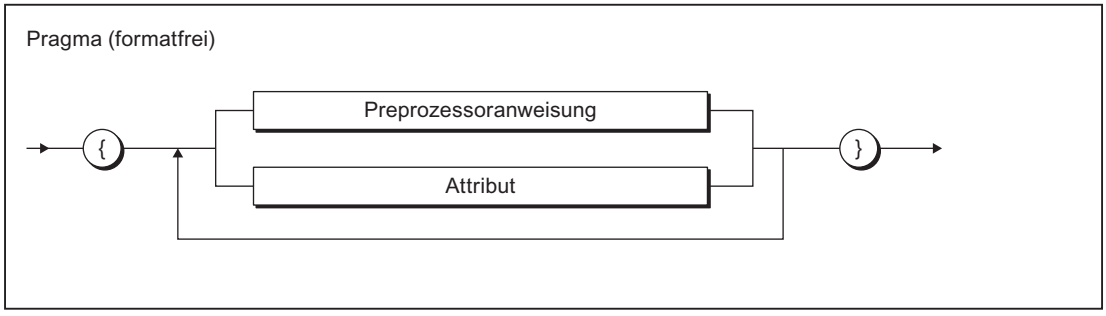


Bild 6-8 Syntax für Pragma

Hinweis

Achten Sie genau auf die Syntax eines Pragmas (z. B. Groß- und Kleinschreibung bei Attributen).

Nicht erkannte Pragmas werden ohne Warnmeldung ignoriert.

6.7.1 Präprozessor steuern

Der Präprozessor bereitet eine ST-Quelle für das Übersetzen vor. Es können z. B. Zeichenfolgen als Ersetzungstexte für Bezeichner definiert werden oder Abschnitte des Quellprogramms für das Compilieren ein- oder ausgeblendet werden.

Standardmäßig ist der Präprozessor abgeschaltet. Sie können ihn folgendermaßen aktivieren:

- Global für alle Programmquellen und Programmiersprachen innerhalb des Projekts, siehe "Globale Einstellungen des Compilers (Seite 64)".
- Lokal für eine Programmquelle, siehe "Lokale Einstellungen des Compilers (Seite 67)".

Während des Übersetzens einer Programmquelle werden Sie über die Arbeit des Präprozessors informiert. Hierzu muss die Anzeige von Warnungen der Warnungsklasse 7

aktiviert sein, siehe Bedeutung der Warnungsklassen (Seite 73). Den Umfang der ausgegebenen Warnungen und Informationen legen Sie fest:

- In den globalen bzw. lokalen Einstellungen des Compilers.
- Mit dem Attribut _U7_PoeBld_CompilerOption := warning:*n*:off bzw. warning:*n*:on innerhalb einer ST-Quelle, siehe "Compiler mit Attributen steuern (Seite 312)".

Die Informationen über die Arbeit des Präprozessors erfolgen wie alle Meldungen des Compilers im Register "Ausgabe übersetzen / prüfen" der Detailanzeige.

Hinweis

Sie können auch den vom Präprozessor modifizierten Text der ST-Quelle anzeigen lassen:

1. Öffnen Sie die ST-Quelle.
2. Wählen Sie Menü **ST-Quelle > Präprozessor ausführen**.

Der modifizierte Quelltext wird im Register "Ausgabe übersetzen / prüfen" der Detailanzeige angezeigt.

6.7.1.1 Präprozessor-Anweisung

Den Präprozessor können Sie über Anweisungen in Pragmas steuern. Es stehen die im folgenden Syntaxdiagramm angegebenen Anweisungen zur Verfügung. Die Anweisungen wirken auf alle folgenden Zeilen der ST-Quelle.

Sie können in ST-Quellen eines SIMOTION Geräts oder einer Bibliothek verwendet werden

Definitionen für den Präprozessor können Sie auch im Eigenschaftsdialog der ST-Quelle vornehmen (siehe Präprozessor-Definitionen vornehmen (Seite 75)). Sie können so den Präprozessor auch bei ST-Quellen mit Know-how-Schutz (siehe Know-how-Schutz für ST-Quellen (Seite 75) steuern.

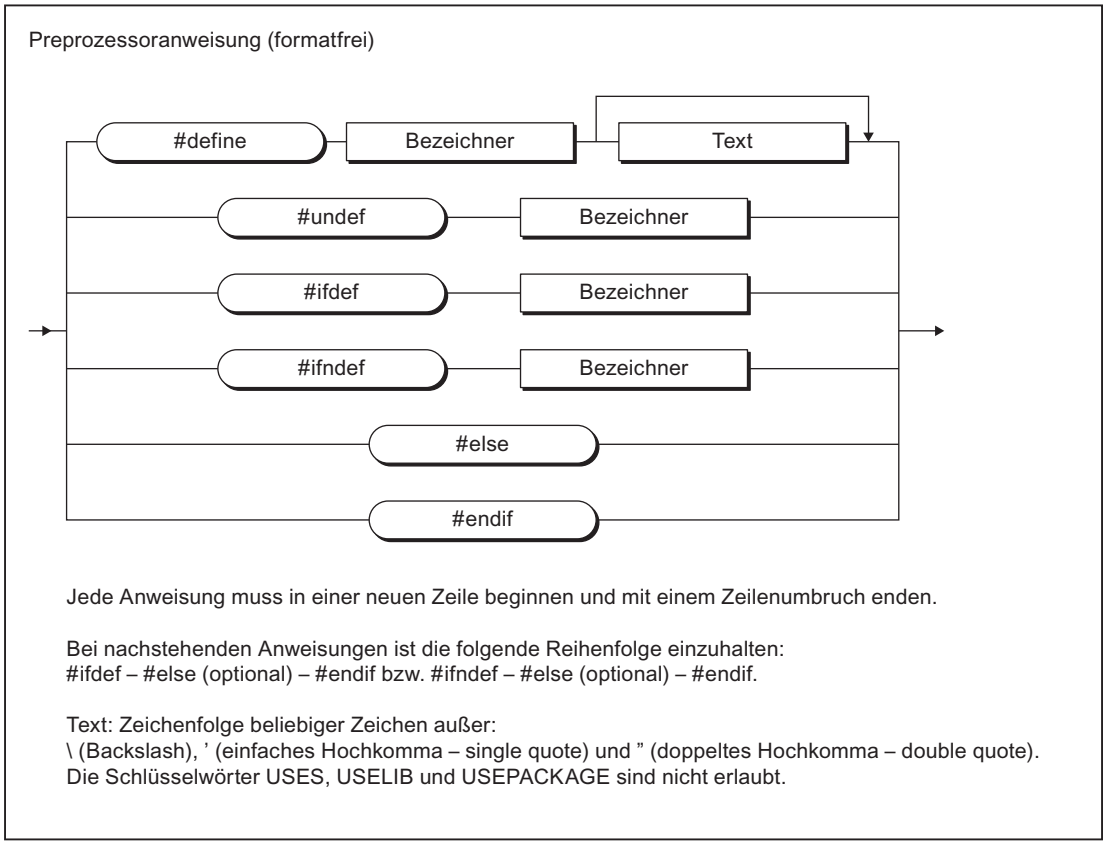


Bild 6-9 Syntax einer Präprozessor-Anweisung

Tabelle 6-35 Präprozessor-Anweisungen

Anweisung	Bedeutung
#define	Der angegebene Bezeichner wird im Folgenden durch den angegebenen Text ersetzt. Erlaubte Zeichen: Siehe Fußteil der Tabelle.
#undef	Die Ersetzungsvorschrift für den Bezeichner wird aufgehoben.
#ifdef	Für Variantenbildung (bedingtes Übersetzen) Wenn der angegebene Bezeichner definiert ist, werden die nachfolgenden Programmzeilen (bis zum nächsten Pragma, das #else oder #endif enthält) vom Compiler übersetzt.
#ifndef	Für Variantenbildung (bedingtes Übersetzen) Wenn der angegebene Bezeichner **nicht** definiert ist, werden die nachfolgenden Programmzeilen (bis zum nächsten Pragma, das #else oder #endif enthält) vom Compiler übersetzt.
#else	Für Variantenbildung (bedingtes Übersetzen) Alternativzweig zu #ifdef oder #ifndef. Die nachfolgenden Programmzeilen (bis zum nächsten Pragma, das #endif enthält) werden vom Compiler übersetzt, wenn die vorhergehende Abfrage mit #ifdef oder #ifndef nicht erfüllt war.

Anweisung	Bedeutung
#endif	Schließt Variantenbildung mit #ifdef oder #ifndef ab.
Erlaubte Zeichen: • Für Bezeichner: Entsprechend den Regeln für Bezeichner (Seite 100). • Für Text: Folge beliebiger Zeichen außer \ (Backslash), ' (einfaches Hochkomma - single quote), " (doppeltes Hochkomma - double quote). Die Schlüsselwörter USES, USELIB und USEPACKAGE sind nicht erlaubt.	

Hinweis

Jede Präprozessor-Anweisung muss in einer neuen Zeile beginnen und mit einem Zeilenumbruch enden. Die das Pragma umschließenden geschweiften Klammern { und } müssen deshalb in separaten Zeilen der ST-Quelle stehen!

Beachten Sie bei Pragmas mit #define-Anweisungen:

- Im Interface-Abschnitt einer ST-Quelle vorhandene Pragmas mit #define-Anweisungen werden exportiert. Die definierten Bezeichner können mit der USES-Anweisung in andere ST-Quellen desselben SIMOTION Geräts oder derselben Bibliothek importiert werden.
- In Pragmas von Bibliotheken definierte Bezeichner können nicht in ST-Quellen eines SIMOTION Geräts importiert werden.
- Das Umdefinieren reservierter Bezeichner ist nicht möglich.

Präprozessor-Definitionen können Sie auch im Eigenschaftsdialog der ST-Quelle vornehmen. Bei unterschiedlichen Definitionen gleicher Bezeichner haben #define-Anweisungen innerhalb der ST-Quelle Vorrang.

6.7.1.2 Beispiel für Präprozessor-Anweisungen

Tabelle 6-36 Beispiel für Präprozessor-Anweisungen

ST-Quelle mit Präprozessoranweisungen

```
INTERFACE
    FUNCTION_BLOCK fb1;
    VAR_GLOBAL
        g_var     : INT;
    END_VAR
// Präprozessor-Definitionen
{
#define my_define g_var
#define my_call f(my_define)
}
//     my_define -> g_var
//     my_call   -> f(g_var)
END_INTERFACE

IMPLEMENTATION
    FUNCTION f : INT
        VAR_INPUT
            i : INT;
        END_VAR
        f := i;
    END_FUNCTION

    FUNCTION_BLOCK fb1
        VAR_INPUT
            i_var : INT;
        END_VAR
        VAR_OUTPUT
            o_var : INT;
        END_VAR
        my_define := i_var;
// Löschen der Präprozessor-Definition
// für my_define
{
#undef my_define
}
        o_var := my_call + 1;
{
#ifdef my_define
}
        my_define := i_var;
{
#endif
}
    END_FUNCTION_BLOCK
END_IMPLEMENTATION
```

Ausgabe des Präprozessors

```
INTERFACE
    FUNCTION_BLOCK fb1;
    VAR_GLOBAL
        g_var     : INT;
    END_VAR

{


}


END_INTERFACE

IMPLEMENTATION
    FUNCTION f : INT
        VAR_INPUT
            i : INT;
        END_VAR
        f := i;
    END_FUNCTION

    FUNCTION_BLOCK fb1
        VAR_INPUT
            i_var : INT;
        END_VAR
        VAR_OUTPUT
            o_var : INT;
        END_VAR
        g_var := i_var;


{

}
        o_var := f(g_var) + 1;
{


}
    END_FUNCTION_BLOCK
END_IMPLEMENTATION
```

6.7.2 Compiler mit Attributen steuern

Mit Attributen innerhalb eines Pragmas können Sie den Compiler steuern.

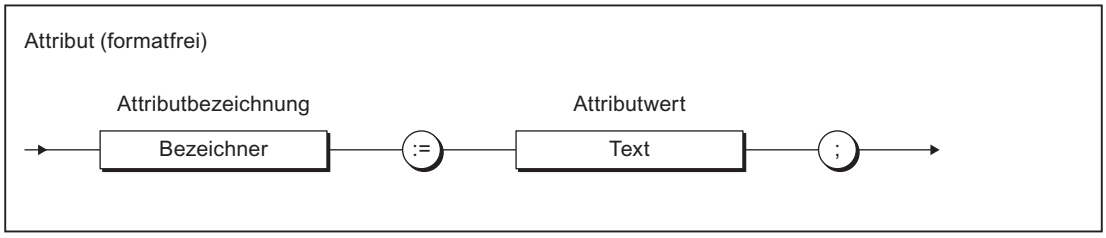


Bild 6-10 Syntax eines Attributs für Compileroptionen

Tabelle 6-37 Zulässige Attribute für Compileroptionen

Attributbezeichnung	Attributwert	Bedeutung
_U7_PoeBld_CompilerOption	Das Attribut beeinflusst die Ausgabe von Warnungen und Informationen des Compilers innerhalb einer ST-Quelle Es wirkt auf alle folgenden Zeilen der ST-Quelle.	
	warning:*n*:err	Ausgabe der durch die Zahl *n* spezifizierten Warnung oder Information als Fehler. Erlaubte Werte für *n*: *n* = 16000 ff: Nummer einer Warnung oder Information.
	warning:*n*:off	Keine Anzeige der durch die Zahl *n* spezifizierten Warnungen oder Informationen. Erlaubte Werte für *n*: *n* = 0..7: Warnungsklasse, siehe Bedeutung der Warnungsklassen (Seite 73). *n* = 16000 ff: Nummer einer Warnung oder Information.
	warning:*n*:on	Anzeige der durch die Zahl *n* spezifizierten Warnungen Erlaubte Werte für *n*: *n* = 0..7: Warnungsklasse, siehe Bedeutung der Warnungsklassen (Seite 73). *n* = 16000 ff: Nummer einer Warnung oder Information.
HMI_Export	Das Attribut ändert die Standardvorgabe, welche Unit-Variablen auf HMI-Geräten zur Verfügung stehen. Es muss direkt nach dem jeweiligen Schlüsselwort folgender Deklarationsblöcke stehen: • VAR_GLOBAL • VAR_GLOBAL RETAIN Es wirkt nur auf die im jeweiligen Deklarationsblock deklarierten Unit-Variablen. Ausführliche Beschreibung des HMI-Exports, insbesondere über die Wirkung des Attributs in Abhängigkeit von der Version des SIMOTION Kernels: siehe Variablen und HMI-Geräte (Seite 256).	
	FALSE	Im Interfaceabschnitt einer ST-Quelle. Die im betreffenden Deklarationsblock deklarierten Unit-Variablen stehen auf HMI-Geräten **nicht** zur Verfügung.
	TRUE	Im Implementationsabschnitt einer ST-Quelle. Die im betreffenden Deklarationsblock deklarierten Unit-Variablen stehen auf HMI-Geräten zur Verfügung.

Attributbezeichnung	Attributwert	Bedeutung
BlockInit_OnChange	Das Attribut ändert die Standardvorgabe, ob ein Download im RUN bei einer Änderung der Versionskennung des jeweiligen Deklarationsblocks möglich ist. Es muss direkt nach dem jeweiligen Schlüsselwort folgender Deklarationsblöcke stehen: • VAR_GLOBAL (im Interface- und Implementationsabschnitt) • VAR_GLOBAL RETAIN (im Interface- und Implementationsabschnitt) • VAR (nur bei Programmen in einer Unit, wenn die Compileroption "Programminstanzdaten nur einmal anlegen" aktiv ist). Es wirkt nur auf die im jeweiligen Deklarationsblock deklarierten Variablen. Siehe auch Versionskennung globaler Variablen und deren Initialisierung beim Download (Seite 254).	
	FALSE	Download im RUN ist bei Änderung der Versionskennung des Deklarationsblocks **nicht** möglich (Voreinstellung).
	TRUE	Download im RUN ist trotz Änderung der Versionskennung des Deklarationsblocks möglich. Die Variablen des Deklarationsblocks werden dabei initialisiert.
BlockInit_OnDeviceRun	Nur wirksam ab Version V4.1 des SIMOTION Kernels. Das Attribut ändert die Standardvorgabe, ob die Variablen des jeweiligen Deklarationsblocks beim Übergang in den Betriebszustand RUN initialisiert werden. Es muss direkt nach dem jeweiligen Schlüsselwort folgender Deklarationsblöcke stehen: • VAR_GLOBAL (im Interface- und Implementationsabschnitt) • VAR (nur bei Programmen in einer Unit, wenn die Compileroption "Programminstanzdaten nur einmal anlegen" aktiv ist). Es wirkt nur auf die im jeweiligen Deklarationsblock deklarierten Variablen. Siehe auch Speicherbereiche der Variablentypen (Seite 240).	
	DISABLE	Die im betreffenden Deklarationsblock deklarierten Variablen werden beim Übergang des Betriebszustands von STOP nach RUN **nicht** initialisiert.
	ALWAYS	Die im betreffenden Deklarationsblock deklarierten Variablen werden beim Übergang des Betriebszustands von STOP nach RUN initialisiert.
	Voreinstellung: Bis Version V4.1 des SIMOTION Kernels: DISABLE. Ab Version V4.2 des SIMOTION Kernels: Gemäß der Einstellung am Gerät	

Hinweis

Achten Sie bei Attributen auf Groß- und Kleinschreibung!

Hinweis

Durch Einfügen, Löschen oder Ändern der Attribute HMI_Export, BlockInit_OnChange bzw. BlockInit_OnDeviceRun in einen Deklarationsblock ändert sich dessen Versionskennung nicht!

Tabelle 6-38 Beispiel für Attribute für Compileroptionen

```
INTERFACE
    VAR_GLOBAL
        { HMI_Export := FALSE;
          BlockInit_OnChange := TRUE; }
        // Kein HMI-Export, Download im RUN möglich
        x : DINT;
    END_VAR
    FUNCTION_BLOCK fb1;
END_INTERFACE

IMPLEMENTATION
    VAR_GLOBAL
        { HMI_Export := TRUE;
          BlockInit_OnDeviceRun := ALWAYS; }
        // HMI-Export, Initialisierung beim Übergang STOP -> RUN
        y : DINT;
    END_VAR
    FUNCTION_BLOCK fb1
        VAR_INPUT
            i_var    : INT;
        END_VAR
        VAR_OUTPUT
            o_var    : INT;
        END_VAR

        { _U7_PoeBld_CompilerOption := warning:2:on; }
        o_var := REAL_TO_INT(1.0);    // Warnung 16004
        { _U7_PoeBld_CompilerOption := warning:2:off; }
        o_var := REAL_TO_INT(1.0);    // keine Warnung 16004
        { _U7_PoeBld_CompilerOption := warning:16004:on; }
        o_var := REAL_TO_INT(1.0);    // Warnung 16004
        { _U7_PoeBld_CompilerOption := warning:16004:off; }
        o_var := REAL_TO_INT(1.0);    // keine Warnung 16004
        { _U7_PoeBld_CompilerOption := warning:2:off;
          _U7_PoeBld_CompilerOption := warning:16004:on; }
        o_var := REAL_TO_INT(1.0);    // Warnung 16004
    END_FUNCTION_BLOCK
END_IMPLEMENTATION
```

6.8 SIMOTION Geräte

6.8.1 Regeln für Bezeichner der SIMOTION-Geräte

Gerätebezeichner allgemein

Bezeichner für SIMOTION Geräte im Projektnavigator müssen nicht den allgemeinen Regeln für Bezeichner (Seite 100) entsprechen.

Für den Bezeichner eines SIMOTION Geräts sind alle darstellbaren Zeichen des erweiterten ASCII-Zeichensatzes erlaubt (ASCII-Code $20 bis $7E, $80 bis $FF), außer folgenden Sonderzeichen:

- " (doppeltes Hochkomma, ASCII-Code $22),
- & (Et-Zeichen, Ampersand, ASCII-Code $26),
- * (Stern, ASCII-Code $2A),
- : (Doppelpunkt, ASCII-Code $3A),
- < (Kleiner-als-Zeichen, ASCII-Code $3C),
- > (Größer-als-Zeichen, ASCII-Code $3E),
- ? (Fragezeichen, ASCII-Code $3F),
- \ (Backslash, ASCII-Code $5C),
- | (Vertikale Linie, ASCII-Code $7C).

Bezeichner für SIMOTION Geräte können nur in SIMOTION SCOUT bzw. HW Konfig festgelegt werden, in den Programmiersprachen können sie ausschließlich verwendet werden.

Beispiele für gültige Gerätebezeichner

- C240.2
- D455-2-DP (1)
- D445-2.PN-1

Gerätebezeichner, die nicht den allgemeinen Regeln für Bezeichner genügen, können für SIMOTION Geräte aller Versionen (bzw. alle Versionen des SIMOTION Kernels) verwendet werden.

Hinweis

Projekte, die Gerätebezeichner enthalten, die nicht den allgemeinen Regeln für Bezeichner (Seite 100) genügen, können nicht im alten Projektformat (bis einschließlich V4.2) gespeichert werden.

Gerätebezeichner für PROFINET IO (Name of Station)

Gerätebezeichner, die als Gerätenamen bei PROFINET IO (NameOfStation) Verwendung finden, müssen den folgenden DNS-Konventionen entsprechen:

- Zulässige Länge: 1 bis 127 Zeichen.
- Gliederung durch Punkte "." in mehrere Labels zulässig; Länge eines Labels 1 bis 63 Zeichen.
- Zulässige Zeichen innerhalb eines Labels:
  - Buchstaben "A" bis "Z" und "a" bis "z".
  - Ziffern "0" bis "9" (nicht am Anfang des Labels).
  - Sonderzeichen Bindestrich "-" (nicht am Anfang und Ende des Labels).
  - Andere Sonderzeichen (z. B. Umlaute, Leerzeichen, Klammern, Unterstrich) sind nicht zulässig.
- Folgende Bezeichner für Gerätenamen sind nicht zulässig:
  - Bezeichner, die mit "port-xyz-" beginnen (x, y, z = 0 … 9),
  - Bezeichner der Form n.n.n.n (n = 0 … 999).

Verwendung von Gerätebezeichnern in SIMOTION SCOUT

Bezeichner für SIMOTION Geräte, die nicht den allgemeinen Regeln für Bezeichner genügen, **müssen** bei Verwendung in SIMOTION SCOUT (z. B in den Programmiersprachen) in doppelte Hochkommata (", ASCII-Code $22) eingeschlossen werden.

Beispiel:

- "D455-2 DP (1)".axis_1. motionStateData.actualVelocity
  Zugriff auf die Systemvariable *motionStateData.actualVelocity* des Technologieobjekts *axis_1* am Gerät *D455-2 DP (1)*.

Hinweis

Auch Gerätebezeichner, die den allgemeinen Regeln für Bezeichner (Seite 100) genügen, **dürfen** in doppelte Hochkommata eingeschlossen werden.

Beispiel: für den Zugriff auf die Systemvariable *motionStateData.motionState* des Technolgieobjekts *axis_2* am Gerät *D435_2* sind folgende Schreibweisen zulässig:

- D435_2.axis_2.motionStateData.motionState
- "D435_2".axis_2.motionStateData.motionState

6.8.2 Einstellungen am Gerät vornehmen (ab Kernel V4.2)

Ab Version V4.2 des SIMOTION Kernels können Sie unter Anderem folgende Einstellungen am SIMOTION Gerät vornehmen:

- Speicherbereich für das Prozessabbild der zyklischen Tasks und das feste Prozessabbild der BackgroundTaks:
  - Separate Speicherbereiche für beide Prozessabbilder - Separates Prozessabbild (Seite 278)
  - Gemeinsames Prozessabbild (Seite 275)
- Initialisierung der nicht remanenten globalen Variablen und Programmdaten beim STOP-RUN-Übergang
- Uhrzeitsynchronisation mit SINAMICS-Antriebsgeräten durchführen
- OPC-XML ermöglichen für geräteglobale Variablen bzw. I/O-Variablen
  Die Symbolinformationen dieser Variablen stehen im SIMOTION Gerät zur Verfügung. Die ist z. B. notwendig für die Watch-Funktion von IT DIAG.
- Bereich für automatische Adressvergabe bei der Telegrammprojektierung einschränken (ab Version V4.3 des SIMOTION Kernels).
  Bei aktiver Checkbox wird der angegebene Adressbereich bei der automatischen Telegrammprojektierung gesperrt. Der Angabe des Sperrbereichs wirkt nicht, wenn die Telegramm bereits projektiert sind.

Eine ausführliche Beschreibung finden Sie in der Online-Hilfe.

Vorgehensweise

So nehmen Sie die Einstellung am Gerät vor:

1. Markieren Sie im Projektnavigator das entsprechende SIMOTION Gerät.
2. Wählen Sie Menü **Bearbeiten > Objekteigenschaften**.
3. Wählen Sie das Register **Einstellungen**.
4. Nehmen Sie die Einstellungen vor.
5. Bestätigen Sie mit **OK**.

6.9 Vorwärtsdeklarationen

Standardmäßig müssen Sie bei der Erstellung der Quelldatei auf die Reihenfolge der Quelldatei-Abschnitte achten. Ein aufzurufender Abschnitt muss immer vor dem aufrufenden Abschnitt stehen, damit er diesem bekannt ist.

Beispielsweise müssen Variablen vor der Verwendung deklariert, Funktionen vor dem Aufruf definiert, Funktionsbausteine vor der Instanzdeklaration definiert werden.

Vorwärtsdeklaration für Programmorganisationseinheiten (POE)

Bei aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben" sind in einer Quelldatei die nachfolgenden Verwendungen von Programmorganisationseinheiten (POE) möglich, bevor die jeweilige POE vollständig definiert wurden:

- Instanzdeklaration eines Funktionsbausteins (Seite 198),
- Aufruf einer Funktion (Seite 197),
- Aufruf eines Programms innerhalb eines Programms (Seite 206), sofern die weiteren Voraussetzungen hierfür erfüllt sind.

Wenn diese Compileroption aktiviert ist, werden folgende Anweisungen innerhalb des Konstrukts TYPE / END_TYPE im Interfaceabschnitt (Seite 212) oder Implementationsabschnitt (Seite 214) einer ST-Quelle als Prototypen interpretiert:

- FUNCTION_BLOCK *fb-name*;
- FUNCTION *fc-name*;
- PROGRAM *prog-name*;

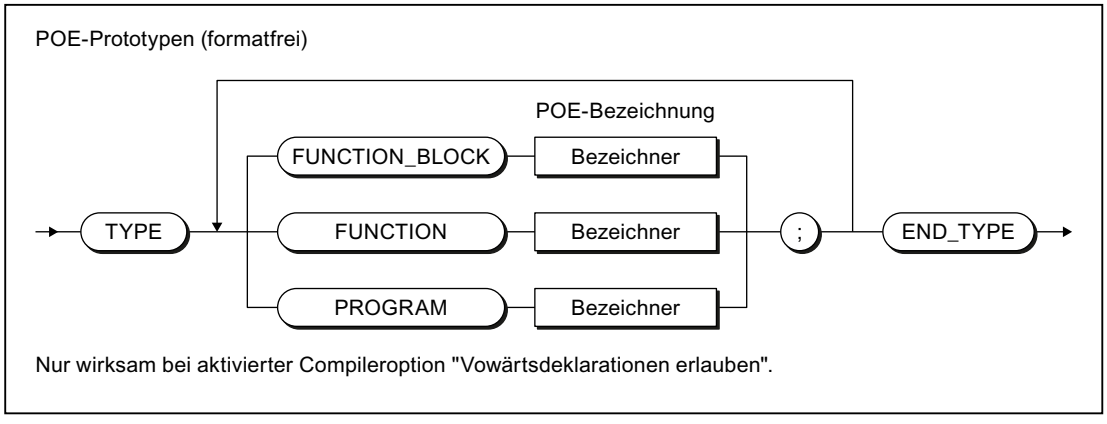


Bild 6-11 Syntax: POE-Prototypen

Nach der Deklaration des Prototyps eines Funktionsbausteins kann eine Instanz des Funktionsbausteins deklariert werden; die Implementierung des Funktionsbaustein erfolgt erst nachfolgend in der ST-Quelle.

Beim Aufruf einer Funktion oder eines Programms vor deren Implementierung ist die Deklaration des Prototyps optional; der Aufruf kann auch ohne Deklaration des Prototyps

erfolgen. Beachten Sie beim Aufruf eines Programms innerhalb eines Programms die weiteren Voraussetzungen.

Hinweis

Wenn die POE-Prototypen innerhalb des Interfaceabschnitts deklariert sind, werden die entsprechenden POE exportiert.

Exportanweisungen für POE innerhalb des Interfaceabschnitts (z. B. FUNCTION_BLOCK *fb-name*;) werden ebenfalls als Prototypen interpretiert.

Bei nicht aktivierter Compileroption (Seite 64) "Vorwärtsdeklarationen erlauben" werden die POE-Prototypen ignoriert. Nur die Prototypen im Interfaceabschnitt werden als Exportanweisungen für die entsprechende POE interpretiert.

Beispiel

Tabelle 6-39 Beispielprogramm mit Vorwärtsdeklaration

```
(*
Nur mit aktivierter Compileroption "Vorwärtsdeklaration erlauben".
*)

(*
Wegen Aufruf eines Programms im Programm ist auch
die Aktivierung folgender Compileroptionen nötig:
"Spracherweiterungen zulassen" und
"Programminstanzdaten nur einmal anlegen".
*)

INTERFACE
    PROGRAM prog_main;
END_INTERFACE

IMPLEMENTATION
    TYPE                        // POE-Prototypen
        FUNCTION_BLOCK fb_1; // Nötig für Instanzdeklaration
        FUNCTION fc_1;       // Optional
        PROGRAM prog_1;      // Optional
    END_TYPE

    // Instanzdeklaration vor Implementierung
    VAR_GLOBAL
        var_fb_1 : fb_1;
    END_VAR

    PROGRAM prog_main
        VAR
            var_1, var_2 : DINT;
            var_3, var_4 : INT;
        END_VAR

        var_fb_1 (x_in := var_3, x_out => var_4);

        // Aufruf vor Implementierung
        var_2 := fc_1 (var_1);
        prog_1();
    END_PROGRAM

    // Implementierungen

    FUNCTION_BLOCK fb_1
        VAR_INPUT
            x_in : INT;
        END_VAR

        VAR_OUTPUT
            x_out : INT;
        END_VAR
```

```
        x_out := x_in;
    END_FUNCTION_BLOCK

    FUNCTION fc_1: DINT
        VAR_INPUT
            x_in : DINT;
        END_VAR

        fc_1 := x_in;
    END_FUNCTION

    PROGRAM prog_1
        VAR
            var_int1, var_int2 : INT;
        END_VAR

        var_int1 := var_int2;
    END_PROGRAM
END_IMPLEMENTATION
```

6.10 Sprunganweisung und -markierung

Als zusätzliche Kontrollanweisung (Seite 167) steht eine Sprunganweisung zur Verfügung.

Eine Sprunganweisung (Seite 178) programmieren Sie mit dem Befehl GOTO unter Angabe der Sprungmarke, zu der Sie springen wollen. Sprünge sind nur innerhalb einer POE zulässig.

Die Sprungmarke geben Sie (durch Doppelpunkt getrennt) vor der Anweisung ein, mit der Sie im Programmablauf fortfahren wollen.

Optional können die Sprungmarken in der POE deklariert werden (durch Struktur LABEL/ END_LABEL in der POE). Im Anweisungsabschnitt können dann nur die deklarierten Sprungmarken verwendet werden.

Syntax von Sprunganweisungen und -marken:

Syntaxbeispiel für Sprunganweisungen

```
FUNCTION func : VOID
    VAR
        x, y, z : BOOL;
    END_VAR
    LABEL
        lab_1, lab_2;      // Deklaration der Sprungmarken
    END_LABEL

    x := y;
  lab_1 : y := z;          // Sprungmarke mit Anweisung
    IF x = y THEN
        GOTO lab_2;        // Sprunganweisung
    END_IF;
    GOTO lab_1;            // Sprunganweisung
  lab_2 : ;                // Sprungsmarke mit Leeranweisung
END_FUNCTION
```

Hinweis

Die GOTO-Anweisung sollte nur in Sonderfällen (z. B. Fehlerbearbeitung) verwendet werden. Nach den Regeln der strukturierten Programmierung sollte sie nicht verwendet werden.

Sprünge sind nur innerhalb einer POE zulässig.

Folgende Sprünge sind nicht zulässig:

- Sprünge in untergeordnete Kontrollstrukturen (WHILE, FOR usw.)
- Sprünge aus einer WAITFORCONDITION-Struktur
- Sprünge innerhalb von CASE-Anweisungen

Sprungmarken können nur in der POE deklariert werden, in der sie verwendet werden. Wenn Sprungmarken deklariert werden, dürfen nur diese deklarierten Sprungmarken verwendet werden.

Fehlerquellen und Programmtest 7

Dieses Kapitel beschreibt mögliche Fehlerquellen beim Programmieren und zeigt Ihnen, wie Sie effizient programmieren können. Sie erfahren auch, welche Möglichkeiten Sie für den Programmtest haben. Alle möglichen Fehlermeldungen bei der Übersetzung, d. h. Compilerfehler siehe Fehlermeldungen des Compilers und deren Behebung (Seite 424). Zu jedem Fehler erhalten Sie Möglichkeiten der Behebung bzw. der Reaktion.

7.1 Hinweise zur Fehlervermeidung und zum effizienten Programmieren

Im Funktionshandbuch SIMOTION *Basisfunktionen* sind einige häufige Fehlerquellen aufgeführt, die das Compilieren oder das ordnungsgemäße Ausführen eines Programms verhindern. Es gibt z. B. Hinweise für:

- Datentypen bei der Zuweisung arithmetischer Ausdrücke
- Start von Funktionen in zyklischen Tasks
- Wartezeiten in zyklischen Tasks
- Fehler beim Download
- CPU geht nicht in RUN
- CPU geht in STOP
- Größe des Lokaldatenstacks
- usw.

Außerdem finden Sie dort einige Hinweise zum effizienten Programmieren, insbesondere für

- Laufzeitorientierte Programmierung
- Änderungsoptimierte Programmierung

7.2 Programmtest

Syntaxfehler werden von ST beim Übersetzungsvorgang erkannt und angezeigt. Laufzeitfehler bei der Ausführung des Programms werden durch Systemalarme angezeigt oder führen zum Betriebszustand STOP. Logische Programmierfehler können Sie mit den Testfunktionen von ST finden, z. B. mit dem Symbol-Browser, Status Programm, Trace.

Wenn Sie die gleichen Ergebnisse der Testfunktionen wie nachfolgend abgebildet erhalten wollen, ist es empfehlenswert, das Beispielprogramm aus Ein Beispielprogramm erstellen (Seite 85) heranzuziehen.

7.2.1 Betriebsmodi für Programmtest

7.2.1.1 Betriebsmodi der SIMOTION-Geräte

Für den Programmtest stehen verschiedene Betriebsmodi der SIMOTION-Geräte zur Verfügung.

Tabelle 7-1 Betriebsmodi eines SIMOTION Geräts

Betriebsmodus	Bedeutung
Prozessbetrieb	Der Programmablauf im SIMOTION-Gerät ist optimiert für maximale System-Performance. Folgende Diagnosefunktionen stehen zur Verfügung, sie sind aber teilweise wegen der Optimierung auf maximale System-Performance nur eingeschränkt nutzbar: • Variablen im Symbol-Browser bzw. einer Watchtabelle beobachten • Status Programm (nur eingeschränkt): – Eingeschränkte Beobachtung von Variablen (z. B. Variablen in Schleifen, Rückgabewerte von Systemfunktionen). – Maximal 1 Programmquelle (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle)[1] kann beobachtet werden. • Trace-Tool (nur eingeschränkt) mit Messfunktionen für Antriebe und Funktionsgenerator, siehe Online-Hilfe: – Maximal **1** Trace auf jedem SIMOTION-Gerät.
Testbetrieb	Die Diagnosefunktionen des Prozessbetriebs sind im vollen Umfang verfügbar: • Variablen im Symbol-Browser bzw. einer Watchtabelle beobachten • Status Programm: – Beobachtung aller Variablen möglich. – Ab Version V4.0 des SIMOTION Kernels: Mehrere Programmquellen (z. B. ST-Quellen, MCC-Quellen, KOP/FUP-Quellen)[1] pro Task können beobachtet werden. – Bei Version V3.2 des SIMOTION Kernels: Maximal 1 Programmquelle (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle)[1] pro Task kann beobachtet werden. • Trace-Tool mit Messfunktionen für Antriebe und Funktionsgenerator, siehe Online-Hilfe: – Maximal **4** Trace auf jedem SIMOTION Gerät. Zusätzlich ist folgende Diagnosefunktion verfügbar: • Leuchtspur zur Beobachtung des Programmablaufs in zyklisch durchlaufenen Programmzweigen (nur bei Programmiersprache MCC und ab Version V4.2 des SIMOTION Kernels). **Hinweis** Mit zunehmender Nutzung von Diagnosefunktionen erhöht sich die Laufzeit und Speicherbelastung.

Betriebsmodus	Bedeutung
Debug-Modus	Neben den Diagnosefunktionen des Testbetriebs können Sie folgende Funktion nutzen: • Haltepunkte Innerhalb einer Programmquelle können Sie Haltepunkte (Seite 346) setzen. Bei Erreichen eines aktivierten Haltepunktes werden ausgewählte Tasks angehalten. • MotionTasks steuern Im Register "Task Manager" der Gerätediagnose können Sie Tasksteuerbefehle auf MotionTasks anwenden, siehe Funktionshandbuch SIMOTION Basisfunktionen. Maximal 1 SIMOTION-Gerät des Projekts kann in den Debug-Modus geschaltet werden. SIMOTION SCOUT ist im Online-Modus, d. h., mit dem Zielsystem verbunden. Beachten Sie die den nachfolgenden Abschnitt: Wichtige Hinweise zur Lebenszeichenüberwachung (Seite 328).

[1] Jeweils 1 MCC-Chart bzw. 1 KOP/FUP-Programm in einer Programmquelle.

Betriebsmodus wählen

So wählen Sie den Betriebsmodus eines SIMOTION-Geräts:

1. Stellen Sie sicher, dass eine Verbindung mit dem Zielsystem besteht (Online-Modus).
2. Markieren Sie im Projektnavigator das SIMOTION-Gerät.
3. Wählen Sie das Kontextmenü "Betriebsmodus".
4. Wählen Sie den gewünschten Betriebsmodus (siehe oben stehende Tabelle).
   Wenn Sie "Debug-Modus" ausgewählt haben:
   – Akzeptieren Sie die Sicherheitshinweise.
   – Parametrieren Sie die Lebenszeichenüberwachung.

   Beachten Sie den nachfolgenden Abschnitt: Wichtige Hinweise zur Lebenszeichenüberwachung (Seite 328).
5. Bestätigen Sie mit **OK**.
   Das SIMOTION-Gerät wechselt in den ausgewählten Betriebsmodus (außer beim Debug-Modus, siehe nachfolgende Ausführungen).

Besonderheiten beim Debug-Modus

Nur für ein SIMOTION-Gerät kann der Debug-Modus ausgewählt werden.

Wenn Sie den Debug-Modus ausgewählt haben, wechselt nur SIMOTION SCOUT in den Debug-Modus, das SIMOTION-Gerät befindet sich im Testbetrieb.

- Der aktivierte Debug-Modus des SIMOTION SCOUT wird im Projektnavigator durch ein Symbol vor dem SIMOTION-Gerät angezeigt.
- Die Funktionsleiste Haltepunkte wird eingeblendet.

Der Debug-Modus des SIMOTION-Geräts wird erst eingeschaltet, wenn mindestens ein gesetzter Haltepunkt aktiviert ist. Wenn alle Haltepunkte deaktiviert sind, wird der Debug-Modus des SIMOTION-Geräts aufgehoben.

Der eingeschaltete Debug-Modus des SIMOTION-Geräts wird in der Statuszeile angezeigt.

7.2.1.2 Wichtige Hinweise zur Lebenszeichenüberwachung

⚠ WARNUNG

Gefährliche Anlagenzustände möglich

Bei Problemen mit der Kommunikation zwischen PC und dem SIMOTION-Gerät kann es zu gefährlichen Anlagenzustände (z. B. unkontrollierbaren Achsbewegungen) kommen.

Verwenden Sie den deshalb Debug-Modus oder eine Steuertafel nur mit aktivierter Lebenszeichenüberwachung mit einer angemessen kurzen Überwachungszeit!

Beachten Sie unbedingt die einschlägigen Sicherheitsvorschriften.

Die Funktion ist ausschließlich zu Inbetriebnahme-, Diagnose- und Servicezwecken freigegeben. Die Funktion ist generell nur von befugtem Fachpersonal zu benutzen. Die Sicherheitsabschaltungen der übergeordneten Steuerung sind unwirksam.

Deshalb ist auf eine hardwaremäßige Ausführung des NOT-AUS-Kreises zu achten. Hierzu sind vom Anwender die erforderlichen Maßnahmen zu treffen.

In folgenden Fällen tauschen das SIMOTION-Gerät und SIMOTION SCOUT regelmäßig Lebenszeichen aus, um eine ordnungsgemäße Verbindung sicherzustellen:

- Im Debug-Modus bei aktivierten Haltepunkten.
- Beim Steuern einer Achse oder eines Antriebs mit der Steuertafel (Steuerungshoheit beim PC).

Wenn der Austausch der Lebenszeichen länger unterbrochen ist als die eingestellte Überwachungszeit, werden folgende Reaktionen ausgelöst:

- Im Debug-Modus bei aktivierten Haltepunkten:
  - Das SIMOTION-Gerät geht in den Betriebszustand STOP.
  - Die Ausgänge werden deaktiviert (ODIS).
- Beim Steuern einer Achse oder eines Antriebs mit der Steuertafel (Steuerungshoheit beim PC):
  - Die Achse wird stillgesetzt.
  - Die Freigaben werden zurückgesetzt.

Sicherheitshinweise akzeptieren

Nach Auswahl des Debug-Modus oder einer Steuertafel müssen Sie die Sicherheitshinweise akzeptieren. Außerdem können Sie die Lebenszeichenüberwachung parametrieren.

Gehen Sie wie folgt vor:

1. Klicken Sie auf den Button **Einstellungen**.
   Das Fenster "Parameter Lebenszeichenüberwachung" öffnet.
2. Lesen Sie dort, wie nachfolgend beschrieben, die Sicherheitshinweise und parametrieren Sie die Lebenszeichenüberwachung.

Lebenszeichenüberwachung parametrieren

Im Fenster "Parameter Lebenszeichenüberwachung" gehen Sie weiter wie folgt vor:

1. Lesen Sie den Warnhinweis!
2. Klicken Sie auf den Button **Sicherheitshinweise**, um das Fenster mit den ausführlichen Sicherheitshinweisen zu öffnen.
3. Lassen Sie die Voreinstellungen für die Lebenszeichenüberwachung unverändert. Änderungen sollten Sie nur in Ausnahmefällen und unter Beachtung aller Gefahrenhinweise vornehmen.
4. Bestätigen Sie mit **Akzeptieren**, dass Sie die Sicherheitshinweise gelesen haben und die Lebenszeichenüberwachung richtig parametriert haben.

Hinweis

Die Lebenszeichenüberwachung spricht unter anderem in folgenden Fällen an:

- Drücken der Leertaste.
- Wechsel in eine andere Windows-Anwendung.
- Zu hohe Kommunikationslast zwischen SIMOTION-Gerät und SIMOTION SCOUT (z. B. durch Hochladen von Tasktrace-Daten).

Es werden folgende Reaktionen ausgelöst:

- Im Debug-Modus bei aktivierten Haltepunkten:
  - Das SIMOTION-Gerät geht in den Betriebszustand STOP.
  - Die Ausgänge werden deaktiviert (ODIS).
- Beim Steuern einer Achse oder eines Antriebs mit der Steuertafel (Steuerungshoheit beim PC):
  - Die Achse bzw. der Antrieb wird stillgesetzt.
  - Die Freigaben werden zurückgesetzt.

⚠ WARNUNG

Gefährliche Anlagenzustände möglich

Diese Funktion wird nicht in allen Betriebszuständen garantiert.

Deshalb ist auf eine hardwaremäßige Ausführung des NOT-AUS-Kreises zu achten. Hierzu sind vom Anwender die erforderlichen Maßnahmen zu treffen.

7.2.1.3 Parameter Lebenszeichenüberwachung

Tabelle 7-2 Parameterbeschreibung Lebenszeichenüberwachung

Feld	Beschreibung
Lebenszeichenüberwachung	Das SIMOTION-Gerät und SIMOTION SCOUT tauschen regelmäßig Lebenszeichen aus, um eine ordnungsgemäße Verbindung sicherzustellen. Wenn der Austausch der Lebenszeichen länger unterbrochen ist als die eingestellte Überwachungszeit, werden folgende Reaktionen ausgelöst: • Im Debug-Modus bei aktivierten Haltepunkten: – Das SIMOTION-Gerät geht in den Betriebszustand STOP. – Die Ausgänge werden deaktiviert (ODIS). • Beim Steuern einer Achse oder eines Antriebs mit der Steuertafel (Steuerungshoheit beim PC): – Die Achse wird stillgesetzt. – Die Freigaben werden zurückgesetzt. Folgende Parametrierungen sind möglich: • Checkbox **Aktiv**: Bei aktivierter Checkbox ist die Lebenszeichenüberwachung aktiv. Das Deaktivieren der Lebenszeichenüberwachung ist nicht immer möglich. • Überwachungszeit: Geben Sie die Überwachungszeit ein. **Vorsicht** Lassen Sie die Voreinstellungen für die Lebenszeichenüberwachung möglichst unverändert. Änderungen sollten Sie nur in Ausnahmefällen und unter Beachtung aller Gefahrenhinweise vornehmen.
Sicherheitshinweise	**Beachten Sie unbedingt den Warnhinweis!** Klicken Sie auf die Schaltfläche, um weitere Sicherheitshinweise zu erhalten. Siehe: Wichtige Hinweise zur Lebenszeichenüberwachung (Seite 328)

7.2.2 Editieren der Programmquellen im Online-Modus

Online-Editieren im Prozess- oder Testbetrieb

Wenn SIMOTION SCOUT mit dem Zielsystem verbunden ist, das sich im Betriebsmodus "Prozessbetrieb" bzw. "Testbetrieb" befindet, können Programmquellen (z. B. ST-Quellen, MCC-Quellen mit den MCC-Charts) grundsätzlich editiert , übersetzt und im Betriebszustand STOP ins Zielsystem geladen werden. Zum Download im Betriebszustand RUN siehe den entsprechenden Abschnitt im Funktionshandbuch "SIMOTION Basisfunktionen".

Die Testfunktionen "Status Programm", "Beobachten des Programmablaufs" (nur bei MCC) und Leuchtspur (nur bei MCC) können Sie für eine Programmquelle oder eine Programmorganisationseinheit (POE) allerdings nur einschalten, wenn folgende Bedingungen erfüllt sind:

1. Diese Programmquelle oder eine beliebige POE dieser Quelle (z. B. MCC-Chart) enthält keine nicht gespeicherten Änderungen.
2. Die Programmquelle (Unit) im SCOUT ist konsistent mit dem Zielsystem.

Hinweis

Bei aktivierter Testfunktion "Status Programm" ist das Editieren der entsprechenden Programmquelle oder einer ihrer POE gesperrt.

Beim Ändern einer MCC-Quelle oder eines MCC-Charts, bei denen Testfunktionen "Beobachten des Programmablaufs" oder Leuchtspur aktiv sind, werden die Testfunktionen abgebrochen.

Online-Editieren im Debug-Modus

Wenn SIMOTION SCOUT sich im Betriebsmodus Debug-Modus befindet, ist das Editieren möglich, solange sich das SIMOTION-Gerät nicht im Debug-Modus befindet, also keine Haltepunkte aktiviert sind.

Haltepunkte können Sie nur aktivieren und dadurch das SIMOTION-Gerät in den Debug-Modus schalten, wenn die entsprechende Programmquelle und alle ihrer POE gespeichert, aktuell übersetzt und konsistent mit dem Zielsystem sind.

Beim Versuch, eine Programmquelle oder POE zu editieren, wenn sich das SIMOTION-Gerät im Debug-Modus befindet, werden Sie aufgefordert, alle Haltepunkte zu deaktivieren und somit das SIMOTION-Gerät aus dem Debug-Modus zu schalten.

Hinweis

Wenn Haltepunkte aktiviert sind und das SIMOTION-Gerät sich im Debug-Modus befindet:

Die Eingabe eines Leerzeichens bewirkt, dass das SIMOTION-Gerät in den Betriebszustand STOP geht und alle Ausgänge deaktiviert werden (ODIS).

7.2.3 Symbol-Browser

7.2.3.1 Eigenschaften des Symbol-Browsers

Im Symbol-Browser können Sie Name, Datentyp und Werte von Variablen betrachten und gegebenenfalls die Werte ändern. Im Einzelnen können Sie: folgende Variablen sehen:

- Unit-Variablen sowie statische Variablen eines Programms oder Funktionsbausteins
- Systemvariablen eines SIMOTION Geräts oder eines Technologieobjekts
- I/O-Variablen oder geräteglobale Variablen.

Für diese Variablen können sie:

- eine Momentaufnahme der Variablenwerte betrachten,
- die laufende Veränderung der Variablenwerte beobachten,
- Variablenwerte ändern (steuern).

Der Symbol-Browser kann die Variablenwerte jedoch nur dann anzeigen/ändern, wenn das Projekt in das Zielsystem geladen ist und eine Verbindung zum Zielsystem besteht.

7.2.3.2 Symbol-Browser einsetzen

Voraussetzungen

- Stellen Sie sicher, dass eine Verbindung mit dem Zielsystem besteht und dass ein Projekt ins Zielsystem geladen ist. Zum Laden des Projekts mit dem Beispielprogramm siehe "Beispielprogramm ausführen (Seite 91)".
- Das Anwenderprogramm kann, muss aber nicht ausgeführt werden. Wenn es nicht ausgeführt wird, sehen Sie nur die Anfangswerte der Variablen.

Die Vorgehensweise ist abhängig vom Speicherbereich, in dem die zu beobachtenden Variablen abgelegt sind.

Vorgehensweise

So gehen Sie vor:

1. Markieren Sie im Projektnavigator das entsprechende Element gemäß der nachfolgenden Tabelle.
2. Klicken Sie in der Detailanzeige auf das Register **Symbolbrowser**.
   Sie sehen im Symbol-Browser die entsprechenden Variablen.
3. Wählen Sie für jede Variable in der Spalte "Anzeigeformat", wie diese Variable angezeigt werden soll.

Tabelle 7-3 Elemente im Projektnavigator und im Symbolbrowser zu beobachtende Variablen

Im Symbolbrowser zu beobachtende Variablen	Zu markierendes Element im Projektnavigator
Variablen im Anwenderspeicher der Unit oder im remanenten Speicher, siehe Speicherbereiche der Variablentypen (Seite 240): • Remanente und nicht remanente Unit-Variablen des Interfaceabschnitts einer Programmquelle (Unit), • Remanente und nicht remanente Unit-Variablen des Implementationsabschnitts einer Programmquelle (Unit), • Statische Variablen von Funktionsbausteinen, deren Instanzen als Unit-Variablen deklariert sind. • Außerdem, wenn die Programmquelle (Unit) mit der Compileroption (Seite 64) "Programminstanzdaten nur einmal anlegen" übersetzt wurde: – Statische Variablen der Programme. – Statische Variablen von Funktionsbausteinen, deren Instanzen als statische Variablen von Programmen deklariert sind.	Programmquelle (Unit)
Variablen im Anwenderspeicher der Task, siehe Speicherbereiche der Variablentypen (Seite 240): Wenn die Programmquelle (Unit) ohne die Compileroption (Seite 64) "Programminstanzdaten nur einmal anlegen" (Standard) übersetzt wurde, befinden sich folgende Variablen im Anwenderspeicher derjenigen Task, der das Programm zugeordnet wurde: • Statische Variablen der Programme. • Statische Variablen von Funktionsbausteinen, deren Instanzen als statische Variablen von Programmen deklariert sind.	ABLAUFSYSTEM
Systemvariablen eines SIMOTION Geräts	SIMOTION Gerät
Systemvariablen eines Technologieobjekts	Instanz des Technologieobjekts
Geräteglobale Variablen	GERÄTEGLOBALE VARIABLEN
I/O-Variablen (im Register Adressliste der Detailanzeige). Das Register Adressliste der Detailanzeige öffnen Sie mit einem Doppelklick auf das Element ADRESSLISTE im Projektnavigator.	ADRESSLISTE

Hinweis

Temporäre Variablen können Sie (zusammen mit Unit-Variablen und statischen Variablen) mit **Status Programm** (siehe Eigenschaften von Status Programm (Seite 340)) beobachten.

Hinweis

Diagnosefunktion Leuchtspur bei MCC-Programmierung

Bei der Diagnosefunktion Leuchtspur werden verschiedene interne Variablen, deren Bezeichner mit einem Unterstrich beginnen, automatisch vom Compiler angelegt. Diese Variablen sind im Symbol-Browser sichtbar.

Bei eingeschalteter Diagnosefunktion werden diese Variablen für die Steuerung der Diagnosefunktion verwendet. Im Anwenderprogramm dürfen diese Variablen nicht verwendet werden.

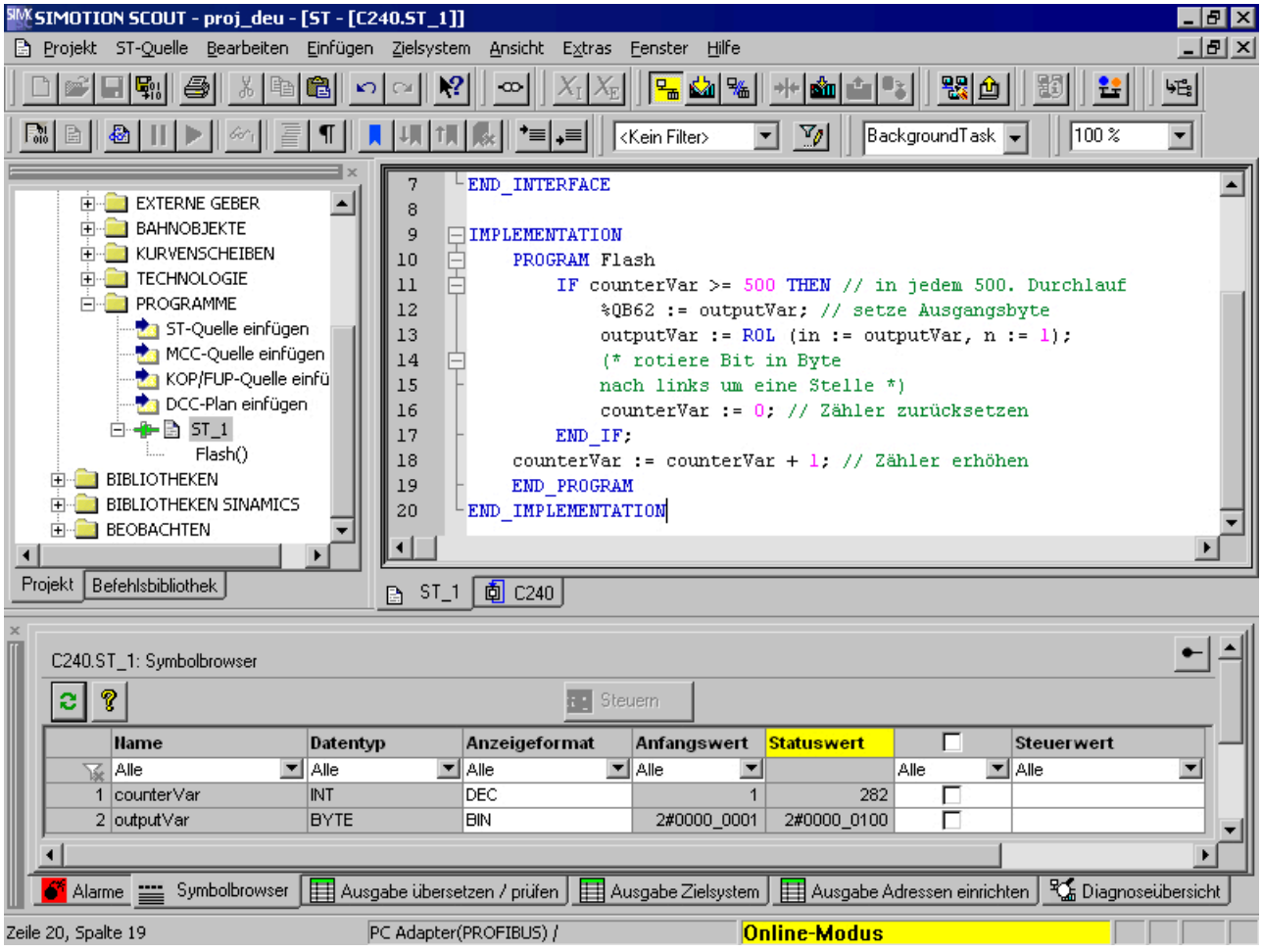


Bild 7-1 Beobachten von Variablen im Symbol-Browser

Status und Steuern von Variablen

In der Spalte **Statuswert** wird der aktuelle Wert der Variablen angezeigt und periodisch aktualisiert.

Sie können den Wert einer oder mehrerer Variablen ändern. Gehen Sie bei den zu ändernden Variablen wie folgt vor:

1. Geben Sie in der Spalte **Steuerwert** einen Wert ein.
2. Aktivieren Sie die Checkbox neben dieser Spalte
3. Klicken Sie auf die Schaltfläche **Steuern**.

Die von Ihnen eingegebenen Werte werden in die ausgewählten Variablen geschrieben.

⚠ WARNUNG

Gefährliche Anlagenzustände möglich

Beim Steuern weisen Sie den Variablen die eingetragenen Werte sofort zu. Damit können gefährliche Anlagenzustände ausgelöst werden z. B. unvorhersehbare Achsbewegungen.

Hinweis

Beachten Sie, wenn Sie die Werte mehrerer Variablen ändern:

Die Werte werden sequentiell in die Variablen geschrieben. Es können mehrere Millisekunden vergehen, bis der nächste Wert geschrieben wird. Die Änderung der Variablen erfolgt im Symbol-Browser von oben nach unten. Die Konsistenz ist deshalb nicht gewährleistet.

Umgang mit dem Symbolbrowser

Die Funktionen des Symbolbrowsers und wie Sie mit ihm umgehen, ist ausführlich in der Online-Hilfe beschrieben.

Anzeige ungültiger Gleitpunktzahlen

Ungültige Gleitpunktzahlen werden im Symbol-Browser (unabhängig vom SIMOTION Gerät) wie folgt angezeigt:

Tabelle 7-4 Anzeige ungültiger Gleitpunktzahlen

Anzeige	Bedeutung
1.#QNAN -1.#QNAN	Ungültiges Bitmuster nach IEEE 754, (NaN – Not a Number). Zwischen signalisierenden NaN (NaNs) und stillen NAN (NaNq) wird nicht unterschieden.
1.#INF -1.#INF	Bitmuster für + Unendlich (+ infinity) nach IEEE 754 Bitmuster für – Unendlich (– infinity) nach IEEE 754
-1.#IND	Bitmuster für Unbestimmt (indeterminate)

7.2.4 Variablen in Watchtabelle beobachten

7.2.4.1 Variablen in der Watchtabelle

Mit dem Symbol-Browser sehen Sie nur die Variablen eines Objekts innerhalb des Projekts. Mit Programm Status sehen Sie nur die Variablen einer ST-Quelle innerhalb eines frei wählbaren Beobachtungsbereichs.

Mit Watchtabellen hingegen können Sie ausgewählte Variablen unterschiedlicher Herkunft (z. B. Programmquellen, Technologieobjekte, SINAMICS Antriebe - auch auf verschiedenen Geräten) als Gruppe beobachten.

Im Offline-Modus können Sie den Datentyp der Variablen sehen. Im Online-Modus sehen Sie den Wert der Variablen und können ihn auch steuern.

7.2.4.2 Watchtabelle einsetzen

Sie können Variablen verschiedener Programmquellen, Technologieobjekte, SIMOTION Geräte usw. (auch auf unterschiedlichen Geräten) in einer Watchtabelle zusammenfassen, gemeinsam beobachten und gegebenenfalls ändern.

Watchtabelle anlegen

So legen Sie eine Watchtabelle an und ordnen ihr Variablen zu:

1. Markieren Sie im Projektnavigator den Ordner **BEOBACHTEN**.
2. Wählen Sie **Einfügen > Watchtabelle**, um eine Watchtabelle zu erzeugen, und geben Sie den Namen der Watchtabelle ein. Eine Watchtabelle mit diesem Namen wird im Ordner **BEOBACHTEN** angelegt.
3. Klicken Sie auf das Objekt im Projektnavigator, von dem Sie Variablen in die Watchtabelle verschieben möchten.
4. Wählen Sie im Symbol-Browser die entsprechende Variablenzeile aus, indem Sie in der linken Spalte auf deren Nummer klicken.
5. Wählen Sie im Kontextmenü den Eintrag **In Watchtabelle aufnehmen** und die entsprechende Watchtabelle, z. B. **Watchtabelle_1**.
6. Wenn Sie auf die Watchtabelle klicken, sehen Sie im Register **Watchtabelle** der Detailansicht, dass sich die ausgewählte Variable in der Watchtabelle befindet.
7. Wiederholen Sie die Schritte 3 bis 6, um Variablen verschiedener Objekte beobachten zu können.

Status und Steuern von Variablen

Wenn Sie mit dem Zielsystem verbunden sind, können Sie die Variableninhalte beobachten.

In der Spalte **Statuswert** wird der aktuelle Wert der Variablen angezeigt und periodisch aktualisiert.

Sie können den Wert einer oder mehrerer Variablen ändern. Gehen Sie bei den zu ändernden Variablen wie folgt vor:

1. Geben Sie in der Spalte **Steuerwert** einen Wert ein.
2. Aktivieren Sie die Checkbox in dieser Spalte
3. Klicken Sie auf die Schaltfläche **Steuern sofort**.

Die von Ihnen eingegebenen Werte werden in die ausgewählten Variablen geschrieben.

⚠ WARNUNG

Gefährliche Anlagenzustände möglich

Beim Steuern weisen Sie den Variablen die eingetragenen Werte sofort zu. Damit können gefährliche Anlagenzustände ausgelöst werden z. B. unvorhersehbare Achsbewegungen.

Hinweis

Beachten Sie, wenn Sie die Werte mehrerer Variablen ändern:

Die Werte werden sequentiell in die Variablen geschrieben. Es können mehrere Millisekunden vergehen, bis der nächste Wert geschrieben wird. Die Änderung der Variablen erfolgt in der Watchtabelle von oben nach unten. Die Konsistenz ist deshalb nicht gewährleistet.

Umgang mit der Watchtabelle

Die Funktionen des Symbolbrowsers und wie Sie mit ihm umgehen, ist ausführlich in der Online-Hilfe beschrieben.

Anzeige ungültiger Gleitpunktzahlen

Ungültige Gleitpunktzahlen werden in der Watchtabelle (unabhängig vom SIMOTION Gerät) wie folgt angezeigt:

Tabelle 7-5 Anzeige ungültiger Gleitpunktzahlen

Anzeige	Bedeutung
1.#QNAN -1.#QNAN	Ungültiges Bitmuster nach IEEE 754, (NaN – Not a Number). Zwischen signalisierenden NaN (NaNs) und stillen NAN (NaNq) wird nicht unterschieden.
1.#INF -1.#INF	Bitmuster für + Unendlich (+ infinity) nach IEEE 754 Bitmuster für – Unendlich (– infinity) nach IEEE 754
-1.#IND	Bitmuster für Unbestimmt (indeterminate)

7.2.5 Status Variable

Mit "Status Variable" können Sie in einer geöffneten Programmquelle bzw. Programmorganisationseinheit (z. B. ST-Quelle, MCC-Chart, KOP-Programm) den aktuellen Wert einer einzelnen Variablen beobachten, die Sie mit dem Mauszeiger anwählen.

Voraussetzungen

- Stellen Sie sicher, dass eine Verbindung mit dem Zielsystem besteht und dass ein Projekt ins Zielsystem geladen ist. Zum Laden eines Projekts siehe "Beispielprogramm ausführen (Seite 91)".
- Die Programmquelle, in der sich Programmorganisationseinheit (POE) befindet, deren Variablen Sie beobachten wollen, ist mit dem Zielsystem konsistent.
- Die zugehörige Quelle (z. B. ST-Quelle, MCC-Chart, KOP-Programm) ist geöffnet.
- Nur bei der Programmiersprache MCC: Die Parametermaske des Befehls, in dem die Variable verwendet wird, die Sie beobachten wollen, ist geöffnet.
- Das Anwenderprogramm kann, muss aber nicht ausgeführt werden. Wenn es nicht ausgeführt wird, sehen Sie nur die Anfangswerte der Variablen.

Vorgehensweise

So beobachten Sie eine einzelne Variable mit Status Variable:

1. Bewegen Sie den Mauszeiger zum Bezeichner einer Variablen
   - Bei der Programmiersprache ST: in der geöffneten ST-Quelle
   - Bei der Programmiersprache ST: innerhalb eines Eingabefeldes in der geöffneten Parametermaske
   - Bei der Programmiersprache KOP/FUP: Innerhalb eines Netzwerks des KOP/FUP-Programms
2. Verweilen Sie mit dem Mauszeiger kurze Zeit über dem Bezeichner.

Der aktuelle Wert der Variablen wird im Tooltip angezeigt. Bei längerem Verweilen des Mauszeigers über dem Bezeichner wird der Wert laufend aktualisiert.

Hinweis

Mit "Status Variable" wird der aktuelle Wert der Variablen unabhängig von der gewählten Verwendungsstelle angezeigt.

Mit der Funktion "Status Variable" können Sie alle Variablen beobachten, die Sie auch im Symbolbrowser (Seite 332) oder in der Adressliste beobachten können. Dies sind:

- Systemvariablen der SIMOTION Geräte
- Systemvariablen der Technologieobjekte
- Geräteglobale Variablen
- Remanente und nicht remanente Unit-Variablen des Interfaceabschnitts einer Programmquelle (Unit)
- Remanente und nicht remanente Unit-Variablen des Implementationsabschnitts einer Programmquelle (Unit)
- Statische Variablen der Programme
- Statische Variablen von Funktionsbausteinen, deren Instanzen als Unit-Variablen deklariert sind

- Statische Variablen von Funktionsbausteinen, deren Instanzen als statische Variablen von Programmen deklariert sind
- I/O-Variablen

7.2.6 Programm-Durchlauf

7.2.6.1 Programm-Durchlauf: Anzeige von Codestelle und Aufrufpfad

Sie können sich die Codestelle (z. B. Zeile einer ST-Quelle), die eine MotionTask aktuell durchläuft, sowie deren Aufrufpfad anzeigen lassen.

So gehen Sie vor:

1. Klicken Sie in der Funktionsleiste Programm-Durchlauf auf die Schaltfläche **Programm-Durchlauf anzeigen**.
   Das Fenster "Callstack Programm-Durchlauf (Seite 339)" öffnet.
2. Wählen Sie die gewünschte MotionTask aus.
3. Klicken Sie auf die Schaltfläche **Aktualisieren**.

Im Fenster wird angezeigt:

- die zu diesem Zeitpunkt durchlaufene Codestelle (z. B. Zeile der ST-Quelle) mit Angabe der Programmquelle und der POE.
- rekursiv die Codestellen anderer POE, welche die durchlaufene Codestelle aufrufen.

Für Programmquellen des SIMOTION RT werden nachstehende Namen angezeigt:

Tabelle 7-6 Programmquellen des SIMOTION RT

Name	Bedeutung
taskbind.hid	Ablaufsystem
stdfunc.pck	IEC-Bibliothek
device.pck	Gerätespezifische Bibliothek
tp-name.pck	Bibliothek des Technologiepakets *tp-name*, z. B. cam.pck für die Bibliothek des Technologiepakets CAM.

7.2.6.2 Parameter Programm-Durchlauf

Sie können sich für alle konfigurierten Tasks anzeigen lassen:

- die aktuell durchlaufene Stelle im Programmcode (z. B. Zeile einer ST-Quelle)
- den Aufrufpfad dieser Codestelle

Tabelle 7-7 Parameterbeschreibung Programm-Durchlauf

Feld	Beschreibung
Selektierte CPU	Das ausgewählte SIMOTION-Gerät wird angezeigt.
Aktualisieren	Durch Klicken auf die Schaltfläche werden die aktuellen Codestellen aus dem SIMOTION-Gerät gelesen und im geöffneten Fenster angezeigt.
Aufrufende Task	Wählen Sie die Task aus, für die Sie die aktuell durchlaufene Codestelle ermitteln wollen. Zur Auswahl stehen alle konfigurierten Tasks des Ablaufsystems.
Aktuelle Codestelle	Die durchlaufene Stelle des Programmcodes (z. B. Zeile einer ST-Quelle) wird angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE).
wird aufgerufen von	Die Codestellen, welche die durchlaufene Codestelle innerhalb der ausgewählten Task aufrufen, werden rekursiv angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE, ggf. Name der Instanz eines Funktionsbausteins).

Für Namen der Programmquellen des SIMOTION RT siehe Tabelle in Programm-Durchlauf (Seite 339).

7.2.6.3 Funktionsleiste Programm-Durchlauf

Mit dieser Funktionsleiste können Sie sich die Codestelle (z. B. Zeile einer ST-Quelle), die eine MotionTask aktuell durchläuft, sowie deren Aufrufpfad anzeigen lassen.

Tabelle 7-8 Funktionsleiste Programm-Durchlauf

Symbol	Bedeutung
	Programm-Durchlauf anzeigen Klicken Sie auf dieses Symbol, um das Fenster Callstack Programm-Durchlauf zu öffnen. In diesem Fenster können Sie die gerade aktive Codestelle mit ihrem Aufrufpfad anzeigen lassen Siehe: Programm-Durchlauf: Anzeige von Codestelle und Aufrufpfad (Seite 339)

7.2.7 Status Programm

7.2.7.1 Eigenschaften von Status Programm

Status Programm ermöglicht eine zyklusgenaue Beobachtung der Variablenwerte während der Programmbearbeitung.

Sie können einen Beobachtungsbereich in der ST-Quelle auswählen und darin neben globalen und statischen lokalen Variablen auch temporäre lokale Variablen (z. B. innerhalb einer Funktion) beobachten.

Die Werte folgender Variablen werden angezeigt:

- Variablen mit einfachem Datentyp (INT, REAL usw.),
- einzelne Elemente einer Struktur, sofern eine Zuweisung erfolgt,
- einzelne Elemente eines Feldes (Arrays), sofern eine Zuweisung erfolgt,
- Variablen mit Aufzählungsdatentypen.

Hinweis

Die Werte von Konstanten werden nicht angezeigt.

Auf Grund der beschränkten Puffer-Kapazitäten und der Forderung nach minimaler Laufzeitverfälschung können folgende Variablen nicht angezeigt werden:

- komplette Felder
- komplette Strukturen

Einzelne Arrayelemente oder einzelne Strukturelemente werden jedoch angezeigt, sofern eine Zuweisung in der ST-Quelle erfolgt.

Funktionsweise von Status Programm

Auf dem SIMOTION Gerät:

- Während des Durchlaufs des markierten Beobachtungsbereichs in der ST-Quelle wird der Puffer für die zu beobachtenden Variablen mit den entsprechenden Werten gefüllt. Die Aufzeichnung der Werte erfolgt abhängig vom Betriebsmodus (Seite 326) (Prozessbetrieb bzw. Testbetrieb), in dem sich das SIMOTION Gerät befindet, siehe nachstehende Tabelle.
- Erst nach Verlassen des markierten Beobachtungsbereichs bzw. nach Abbruch der Aufzeichnung wird der Puffer zur Anzeige im SIMOTION SCOUT bereitgestellt.

SIMOTION SCOUT ruft die bereitgestellten Werte in regelmäßigen Abständen ab und zeigt sie in dem Format an, das Sie bei den Einstellungen des ST-Editors (Seite 35) als "Format für Statusanzeige" gewählt haben.

Bei Funktionen und Funktionsbausteinen können Sie eine Stelle in einer ST-Quelle wählen, an der die Funktion bzw. die Instanz eines Funktionsbausteins gerufen wird (Aufrufpfad). Sie können damit die Variablenwerte gezielt für diesen Aufruf betrachten.

Tabelle 7-9 Unterschiede zwischen Prozessbetrieb und Testbetrieb bei Status Programm

	Prozessbetrieb	Testbetrieb
Optimierung des Programmablaufs	Für maximale System-Performance, nur eingeschränkte Diagnose möglich.	Für volle Diagnosemöglichkeit
Maximale Anzahl der beobachtbaren Programmquellen (z. B. ST-Quellen, MCC-Quellen, KOP/FUP-Quellen)	Maximal 1 Programmquelle[1]	• Ab Version V4.0 des SIMOTION Kernels: Mehrere Programmquellen[1] pro Task • Bei Version V3.2 des SIMOTION Kernels: Maximal 1 Programmquelle[1] pro Task

	Prozessbetrieb	Testbetrieb
Schleifen (z. B. WHILE, REPEAT, FOR)	Die Aufzeichnung wird beim ersten Rücksprung der Schleife abgebrochen. Die Werte werden bereitgestellt. Bei vollständig markierten Schleifen gilt deshalb: • Nach dem ersten Durchlauf der Schleife werden die entsprechenden Werte angezeigt. • Änderungen der Werte während weiterer Schleifendurchläufe werden nicht angezeigt.	Die Aufzeichnung bei den Rücksprüngen erfolgt korrekt. Bei vollständig markierten Schleifen gilt deshalb: • Solange die Schleife durchlaufen wird, werden keine Werte angezeigt. • Erst nach Verlassen des Beobachtungsbereichs werden die Werte beim letzten Durchlauf der Schleife angezeigt.
Systemfunktionen, die intern Schleifen enthalten (z. B. Funktionen zur Bearbeitung von Strings)	Die Aufzeichnung wird während des Ausführens der Systemfunktion abgebrochen. Werte werden unter Umständen nicht korrekt angezeigt.	Die Aufzeichnung während des Aufrufs der Systemfunktion erfolgt korrekt. Werte werden korrekt angezeigt.

[1] Jeweils 1 MCC-Chart bzw. 1 KOP/FUP-Programm in einer Programmquelle

Hinweis

Status Programm benötigt zusätzliche CPU-Ressourcen.

Beachten Sie, wenn Sie mehrere Programme gleichzeitig mit Status Programm beobachten wollen:

- Testbetrieb muss aktiviert sein (siehe Betriebsmodi der SIMOTION Geräte (Seite 326)).
- Bei Version V3.2 des SIMOTION Kernels müssen die Programme verschiedenen Tasks zugeordnet sein.

7.2.7.2 Status Programm einsetzen

Bevor Sie mit Status Programm arbeiten können, müssen Sie dem System mitteilen, dass es in einem besonderen Modus ablaufen soll:

1. Stellen Sie sicher, dass beim Compilieren der ST-Quelle der zusätzliche Debugcode erzeugt wird:
   - Markieren Sie im Projektnavigator die ST-Quelle und wählen Sie das Menü **Bearbeiten > Objekteigenschaften**.
   - Wählen Sie das Register **Compiler**, um die lokalen Einstellungen des Compilers (Seite 67) zu ändern.
   - Stellen Sie sicher, dass die Checkbox **Status Programm ermöglichen** markiert ist, und bestätigen Sie mit **OK**.
     Diese Compileroption können Sie auch in den globalen Einstellungen des Compilers (Seite 64) ändern.
2. Öffnen Sie die ST-Quelle und übersetzen Sie diese neu mit **ST-Quelle > Übernehmen und Übersetzen**.

3. Laden und starten Sie das Programm wie gewohnt.
4. Klicken Sie in der Funktionsleiste ST-Editor (Seite 55) auf die Schaltfläche für **Status Programm**, um diesen Testmodus zu starten.

Das Fenster des ST-Editors erscheint nun vertikal geteilt:

- In der linken Fensterhälfte sehen Sie die ST-Quelle. Dort können Sie einen Bereich markieren.
- In der rechten Fensterhälfte werden die Variablen des ausgewählten Bereichs und deren Werte angezeigt.

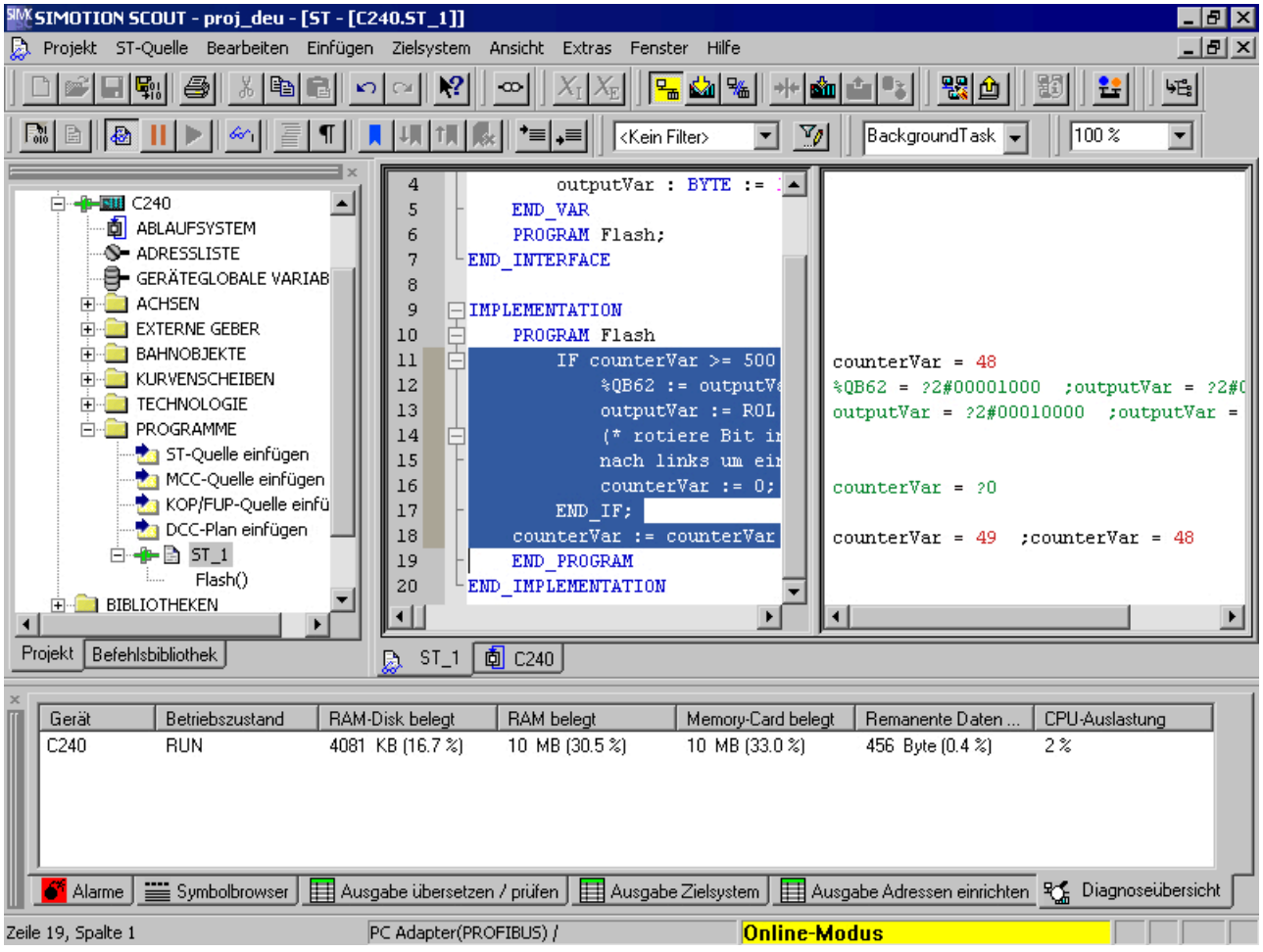


Bild 7-2 Teil eines ST-Programms im Testmodus Status Programm

So testen Sie mit Status Programm:

1. Markieren Sie in der linken Fensterhälfte den Abschnitt der ST-Quelle, den Sie testen wollen.
2. Wenn Sie einen Abschnitt einer POE markiert haben, der von mehreren Stellen einer Programmquelle oder in mehreren Tasks aufgerufen wird:
   Geben Sie den Aufrufpfad für Status Programm (Seite 344) an.

In der rechten Fensterhälfte werden die Variablen des ausgewählten Bereichs und deren Werte angezeigt. Die Anzeige der Variablenwerte wird zyklisch aktualisiert. Bei Variablen mit Bitdatentyp erfolgt sie in dem Format, das Sie bei den Einstellungen des ST-Editors (Seite 35) als "Format für Statusanzeige" gewählt haben:

- Werte, die sich im aktuellen Durchlauf geändert haben, werden in **roter** Schrift dargestellt.
- Werte, die sich nicht geändert haben, werden in **schwarzer** Schrift dargestellt.
- Variablen ohne Werte, z. B. Variablen in einem nicht durchlaufenen IF-Zweig, werden mit Fragezeichen versehen und in **grüner** Schrift angezeigt.

Wenn Ihnen die Anzeige der Variablenwerte zu schnell wechselt:

- Klicken Sie in der Funktionsleiste ST-Editor (Seite 55) auf die Schaltfläche  für **Beobachten der Programmvariablen anhalten**, um die Anzeige zu stoppen.
- Klicken Sie in der Funktionsleiste ST-Editor (Seite 55) auf die Schaltfläche  für **Beobachten der Programmvariablen fortfahren**, um die Anzeige weiterlaufen zu lassen.

Sie können die Aktualisierung der angezeigten Werte erzwingen:

- Klicken Sie in der Funktionsleiste ST-Editor (Seite 55) auf die Schaltfläche  für **Aktualisieren**.
  Der Puffer des SIMOTION Geräts wird ausgelesen, auch wenn der markierte Beobachtungsbereich noch nicht vollständig durchlaufen ist und die Werte unvollständig sind. Dies kann z. B. sinnvoll sein, wenn das Programm an einer WAITFORCONDITION-Anweisung wartet.
  Das Beobachten der Programmvariablen muss eingeschaltet sein.

7.2.7.3 Aufrufpfad für Status Programm

Beim Beobachten der Variablenwerte von Funktionen und Funktionsbausteinen können Sie den Aufrufpfad spezifizieren. Sie können damit die Variablenwerte gezielt für diesen Aufruf betrachten.

Hierzu öffnet sich in folgenden Fällen das Fenster **Aufrufpfad** automatisch:

- Sie haben einen Abschnitt einer Funktion markiert:
  Die Funktion wird an verschiedenen Stellen in den Programmquellen (z. B. ST-Quellen) des SIMOTION Geräts aufgerufen.
- Sie haben einen Abschnitt eines Funktionsbaustein markiert:
  Es gibt mehrere Instanzen des Funktionsbaustein oder die Instanz wird an verschiedenen Stellen in den Programmquellen (z. B. ST-Quellen) des SIMOTION Geräts aufgerufen.
- Sie haben einen Abschnitt eines Programms markiert:
  Das Programm ist mehreren Tasks zugeordnet.

So wählen Sie den Aufrufpfad aus:

Im Fenster **Aufrufpfad Status Programm** wird der markierte Abschnitt der POE (Codestelle) angezeigt (mit Name der ST-Quelle, Zeilennummer, Name der POE).

1. Wenn die Codestelle in mehreren Tasks aufgerufen wird:
   - Wählen Sie die Task aus.
2. Wählen Sie die aufrufende Codestelle (in der aufrufenden POE) aus.
   Zur Auswahl werden angeboten:
   - die aufrufenden Codestellen innerhalb der ausgewählten Task (mit Name der Programmquelle, Zeilennummer, Name der POE)
     Wenn die ausgewählte aufrufende Codestelle wiederum von mehreren Codestellen aufgerufen wird, werden weitere Zeilen eingeblendet, in denen Sie ähnlich vorgehen.
   - **alle**:
     Alle angezeigten Codestellen werden ausgewählt. Darüber hinaus werden alle Codestellen (bis zur höchsten Hierarchieebene) ausgewählt, von denen die angezeigten Codestellen aufrufen werden.

7.2.7.4 Parameter Aufrufpfad Status Programm

Tabelle 7-10 Parameterbeschreibung Aufrufpfad Status Programm

Feld	Beschreibung
Aufrufende Task	Wählen Sie die Task aus. Zur Auswahl stehen alle Tasks, in denen die markierte Codestelle aufgerufen wird.
Aktuelle Codestelle	Der markierte Abschnitt der POE (Codestelle) wird angezeigt (mit Name der ST-Quelle, Zeilennummer, Name der POE)
wird aufgerufen von	Wählen Sie die aufrufende Codestelle aus. Zur Auswahl stehen: • die aufrufenden Codestellen innerhalb der ausgewählten Task (mit Name der Programmquelle, Zeilennummer, Name der POE) Wenn die ausgewählte aufrufende Codestelle wiederum von mehreren Codestellen aufgerufen wird, werden weitere Zeilen eingeblendet, in denen Sie ähnlich vorgehen. • **alle**: Alle angezeigten Codestellen werden ausgewählt. Darüber hinaus werden alle Codestellen (bis zur höchsten Hierarchieebene) ausgewählt, von denen die angezeigten Codestellen aufrufen werden.

7.2.8 Haltepunkte

7.2.8.1 Allgemeine Vorgehensweise zum Setzen von Haltepunkten

Sie können innerhalb einer POE einer Programmquelle (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) Haltepunkte setzen. Bei Erreichen eines aktivierten Haltepunktes wird die Task, in der die POE mit dem Haltepunkt aufgerufen wird, angehalten. Wenn sich der Haltepunkt, der das Anhalten der Tasks ausgelöst hat, in einem Programm oder Funktionsbaustein befindet, werden die Werte der statischen Variablen dieser POE im Register "Status Variablen" der Detailanzeige angezeigt. Temporäre Variablen (auch Durchgangsparameter bei Funktionsbausteinen) werden nicht angezeigt. Statische Variablen anderer POE oder Unit-Variablen können Sie im Symbol-Browser beobachten.

Voraussetzung:

- Die Programmquelle mit der POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) ist geöffnet.

Vorgehensweise

So gehen Sie vor:

1. Wählen Sie für das betreffende SIMOTION-Gerät den Betriebsmodus "Debug-Modus", siehe Debug-Modus einstellen (Seite 347).
2. Legen Sie die anzuhaltenden Tasks fest, siehe Debug-Taskgruppe festlegen (Seite 348).
3. Setzen Sie Haltepunkte, siehe Haltepunkte setzen (Seite 351).
4. Legen Sie den Aufrufpfad fest, siehe Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354).
5. Aktivieren Sie die Haltepunkte, siehe Haltepunkte aktivieren (Seite 361).

7.2.8.2 Debug-Modus einstellen

⚠ WARNUNG

Gefährliche Anlagenzustände möglich

Bei Problemen mit der Kommunikation zwischen PC und dem SIMOTION-Gerät kann es zu gefährlichen Anlagenzuständen (z. B. unkontrollierbaren Achsbewegungen) kommen.

Verwenden Sie deshalb den Debug-Modus nur mit aktivierter Lebenszeichenüberwachung (Seite 328) mit einer angemessen kurzen Überwachungszeit!

Beachten Sie unbedingt die einschlägigen Sicherheitsvorschriften.

Die Funktion ist ausschließlich zu Inbetriebnahme-, Diagnose- und Servicezwecken freigegeben. Die Funktion ist generell nur von befugtem Fachpersonal zu benutzen. Die Sicherheitsabschaltungen der übergeordneten Steuerung sind unwirksam!

Deshalb ist auf eine hardwaremäßige Ausführung des NOT-AUS-Kreises zu achten. Hierzu sind vom Anwender die erforderlichen Maßnahmen zu treffen.

Voraussetzung

1. Eine Verbindung mit dem Zielsystem muss bestehen (Online-Modus)
2. Der Debug-Modus ist für kein SIMOTION-Gerät angewählt.

Vorgehensweise

So stellen Sie den Debug-Modus ein:

1. Markieren Sie im Projektnavigator das SIMOTION-Gerät.
2. Wählen Sie das Kontextmenü **Betriebsmodus**.
3. Wählen Sie den Betriebsmodus (Seite 326) **Debug-Modus**.
4. Akzeptieren Sie die Sicherheitshinweise
5. Parametrieren Sie die Lebenszeichenüberwachung.
   Beachten Sie den Abschnitt: Wichtige Hinweise zur Lebenszeichenüberwachung (Seite 328).
6. Bestätigen Sie mit **OK**.

SIMOTION SCOUT wechselt für dieses Gerät in den Debug-Modus; das SIMOTION-Gerät selbst verbleibt im Betriebsmodus "Testbetrieb", solange bis mindestens ein Haltepunkt aktiviert wird:

Der aktivierte Debug-Modus des SIMOTION SCOUT wird Projektnavigator durch ein Symbol vor dem SIMOTION-Gerät angezeigt.

Die Funktionsleiste Haltepunkte (Seite 353) wird eingeblendet.

Solange keine Haltepunkte aktiviert sind, können im Sie im Debug-Modus Programmquellen editieren (Seite 330).

Der Debug-Modus des SIMOTION-Geräts wird erst eingeschaltet, wenn mindestens ein gesetzter Haltepunkt aktiviert ist. Wenn alle Haltepunkte deaktiviert sind, wird der Debug-Modus des SIMOTION-Geräts aufgehoben. Der eingeschaltete Debug-Modus des SIMOTION-Geräts wird in der Statuszeile angezeigt.

Hinweis

Das Drücken der Leertaste oder der Wechsel in eine andere Windows-Anwendung bewirken, wenn sich das SIMOTION-Gerät im Debug-Modus befindet (Haltepunkte aktiviert):

- Das SIMOTION-Gerät geht in den Betriebszustand STOP.
- Die Ausgänge werden deaktiviert (ODIS).

⚠ WARNUNG

Gefährliche Anlagenzustände möglich

Diese Funktion wird nicht in allen Betriebszuständen garantiert.

Deshalb ist auf eine hardwaremäßige Ausführung des NOT-AUS-Kreises zu achten. Hierzu sind vom Anwender die erforderlichen Maßnahmen zu treffen.

7.2.8.3 Debug-Taskgruppe festlegen

Bei Erreichen eines aktivierten Haltepunktes werden alle Tasks angehalten, die der Debug-Taskgruppe zugeordnet sind.

Voraussetzung

1. Eine Verbindung mit dem Zielsystem muss bestehen (Online-Modus).
2. SIMOTION SCOUT befindet sich für das betreffende SIMOTION-Gerät im Debug-Modus, siehe Debug-Modus einstellen (Seite 347).

Vorgehensweise

So ordnen Sie eine Task der Debug-Taskgruppe zu:

1. Markieren Sie im Projektnavigator das betreffende SIMOTION-Gerät.
2. Wählen Sie Kontextmenü **Debug-Taskgruppe.**
   Das Fenster Debug-Taskgruppe öffnet.
3. Wählen Sie die Tasks aus, die bei Erreichen eines Haltepunktes angehalten werden sollen:
   - Wenn Sie nur einzelne Tasks anhalten wollen (im Betriebszustand RUN): Aktivieren Sie die Auswahlmöglichkeit **Debug-Taskgruppe.**
     Ordnen Sie alle Tasks, die bei Erreichen eines Haltepunktes angehalten werden sollen, der Liste **Anzuhaltende Tasks** zu.
   - Wenn Sie alle Anwender-Tasks anhalten wollen (über Betriebszustand HALT): Aktivieren Sie die Auswahlmöglichkeit **Alle Tasks.**
     Wählen Sie in diesem Fall auch, ob nach dem Fortsetzen der Programmbearbeitung die Ausgänge und Technologieobjekte wieder freigegeben werden sollen.

Hinweis

Beachten Sie das unterschiedliche Verhalten bei Erreichen eines aktivierten Haltepunkts, siehe nachfolgende Tabelle.

Tabelle 7-11 Verhalten am Haltepunkt in Abhängigkeit von den anzuhaltenden Tasks in der Debug-Taskgruppe

Eigenschaft		Anzuhaltende Tasks	
		Einzelne ausgewählte Tasks (Debug-Taskgruppe)	**Alle Tasks**
Verhalten bei Erreichen des Haltepunkts			
	Betriebszustand	RUN	HALT
	Angehaltene Tasks	Nur Tasks in der Debug-Taskgruppe	Alle Tasks
	Ausgänge	Aktiv	Abgeschaltet (ODIS aktiviert)
	Technologie	Regelung aktiv	Keine Regelung (ODIS aktiviert)
	Laufzeitmessung der Tasks	Aktiv für alle Tasks	Abgeschaltet für alle Task
	Zeitüberwachung der Tasks	Abgeschaltet für Tasks in der Debug-Taskgruppe	Abgeschaltet für alle Tasks
	Echtzeituhr	Läuft weiter	Läuft weiter
Verhalten beim Fortsetzen der Programmbearbeitung			
	Betriebszustand	RUN	RUN
	Gestartete Tasks	Alle Tasks in der Debug-Taskgruppe	Alle Tasks
	Ausgänge	Aktiv	Das Verhalten der Ausgänge und Technologieobjekte ist abhängig von der Checkbox **Mit 'Fortsetzen' werden die Ausgänge aktiviert (ODIS deaktiviert).** • Aktiv: ODIS wird deaktiviert. Alle Ausgänge und Technologieobjekte werden freigegeben. • Inaktiv: ODIS bleibt aktiviert. Alle Ausgänge und Technologieobjekte werden nur einen STOP-RUN-Übergang freigegeben.
	Technologie	Regelung aktiv	

Hinweis

Änderungen an der Debug-Taskgruppe können Sie nur vornehmen, wenn keine Haltepunkte aktiviert sind.

Die Einstellungen der Debug-Taskgruppe bleiben auch nach Beenden des Betriebsmodus "Debug-Modus" erhalten.

Weiteres Vorgehen:

1. Setzen Sie Haltepunkte (siehe Haltepunkte setzen (Seite 351)).
2. Legen Sie den Aufrufpfad fest (siehe Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354)).
3. Aktivieren Sie die Haltepunkte (siehe Haltepunkte aktivieren (Seite 361)).

7.2.8.4 Parameter Debug-Taskgruppe

In diesem Fenster legen Sie die Debug-Taskgruppe fest. Bei Erreichen eines aktivierten Haltepunktes werden alle Tasks angehalten, die der Debug-Taskgruppe zugeordnet sind.

Voraussetzung ist, dass sich das betreffende SIMOTION Gerät im Debug-Modus befindet, siehe Betriebsmodi der SIMOTION-Geräte (Seite 326).

Tabelle 7-12 Parameterbeschreibung Debug-Einstellungen

Feld	Beschreibung
Debug-Taskgruppe	Wählen Sie diese Auswahlmöglichkeit, wenn Sie nur einzelne Tasks anhalten wollen. Das SIMOTION Gerät bleibt nach Erreichen eines aktivierten Haltepunktes im Betriebszustand RUN. Ausgänge und Technologieobjekte bleiben aktiviert. Ordnen Sie alle Tasks, die bei Erreichen eines Haltepunktes angehalten werden sollen, der Liste **Anzuhaltende Tasks** zu.
Alle Tasks	Wählen Sie diese Auswahlmöglichkeit, wenn Sie alle Anwender-Tasks anhalten wollen. Das SIMOTION Gerät geht nach Erreichen eines aktivierten Haltepunktes in den Betriebszustand HALT, alle Ausgänge und Technologieobjekte werden deaktiviert (ODIS aktiviert). Wählen Sie in diesem Fall auch, ob nach dem Fortsetzen der Programmbearbeitung die Ausgänge und Technologieobjekte wieder freigegeben werden sollen.
Mit 'Fortsetzen' werden die Ausgänge aktiviert (ODIS deaktiviert).	Nur wenn **Alle Tasks** ausgewählt. Aktivieren Sie die Checkbox, um nach dem Fortsetzen der Programmbearbeitung die Ausgänge und Technologieobjekte wieder freizugeben. Bei deaktivierter Checkbox können die Ausgänge und Technologieobjekte nur durch einen Download des Projekts freigegeben werden.

Hinweis

Beachten Sie das unterschiedliche Verhalten am aktivierten Haltepunkt in Abhängigkeit von den anzuhaltenden Tasks, siehe Tabelle in Debug-Taskgruppe festlegen (Seite 348).

Änderungen an der Debug-Taskgruppe können Sie nur vornehmen, wenn keine Haltepunkte aktiviert sind.

7.2.8.5 Parameter Debug-Tabelle

In der Debug-Tabelle werden alle Haltepunkte in den Programmquellen eines SIMOTION Geräts angezeigt.

Tabelle 7-13 Parameterbeschreibung Debug-Tabelle

Feld		Beschreibung
Debug-Punkte (Tabelle)		
	aktiv	Der Aktivierungszustand des jeweiligen Haltepunkts wird angezeigt und kann geändert werden: **Aktiv**: Der Haltepunkt ist aktiviert. **Inaktiv**: Der Haltepunkt ist deaktiviert. Siehe: Haltepunkte aktivieren (Seite 361).
	Quelle, Zeile (POE)	Die Codestelle mit dem gesetzten Haltepunkt wird angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE)
	Aufrufpfad	Klicken Sie auf die Schaltfläche, um den Aufrufpfad für den Haltepunkt festzulegen. Siehe: Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354).
Alle Haltepunkte …		
	Aktivieren	Klicken Sie auf die Schaltfläche, um alle Haltepunkte (in allen Programmquellen) des SIMOTION Geräts zu aktivieren. Siehe: Haltepunkte aktivieren (Seite 361).
	deaktivieren	Klicken Sie auf die Schaltfläche, um alle Haltepunkte (in allen Programmquellen) des SIMOTION Geräts zu deaktivieren. Siehe: Haltepunkte aktivieren (Seite 361).
	Löschen	Klicken Sie auf die Schaltfläche, um alle Haltepunkte (in allen Programmquellen) des SIMOTION Geräts zu löschen. Siehe: Haltepunkte setzen (Seite 351).

7.2.8.6 Haltepunkte setzen

Voraussetzungen:

1. Die Programmquelle mit der POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) ist geöffnet.
2. Eine Verbindung mit dem Zielsystem muss bestehen (Online-Modus).
3. SIMOTION SCOUT befindet sich für das betreffende SIMOTION-Gerät im Debug-Modus, siehe Debug-Modus einstellen (Seite 347).
4. Die anzuhaltenden Tasks sind festgelegt, siehe Debug-Taskgruppe festlegen (Seite 348).

Vorgehensweise

So setzen Sie einen Haltepunkt:

1. Wählen Sie eine Codestelle aus, an der kein Haltepunkt gesetzt ist:
   - SIMOTION ST: Setzen Sie in der ST-Quelle den Cursor in eine Zeile, die eine Anweisung enthält.
   - SIMOTION MCC: Markieren Sie im MCC-Chart einen MCC-Befehl (außer Modul oder Kommentarblock).
   - SIMOTION KOP/FUP: Setzen Sie den Cursor in ein Netzwerk des KOP/FUP-Programms.
2. Führen Sie folgende Aktion aus (Alternativen):
   - Wählen Sie Menü **Debug > Haltepunkt setzen / entfernen** (Tastenkürzel F9).
   - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

So entfernen Sie einen Haltepunkt:

1. Wählen Sie die Codestelle mit dem Haltepunkt aus.
2. Führen Sie folgende Aktion aus (Alternativen):
   - Wählen Sie Menü **Debug > Haltepunkt setzen / entfernen** (Tastenkürzel F9).
   - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

So entfernen Sie alle Haltepunkte (in allen Programmquellen) des SIMOTION-Geräts:

- Führen Sie folgende Aktion aus (Alternativen):
  - Wählen Sie Menü **Debug > Alle Haltepunkte entfernen** (Tastenkürzel STRG+F5).
  - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

Hinweis

Haltepunkte können Sie nicht setzen:

- Bei SIMOTION ST: in Zeilen, die ausschließlich Kommentar enthalten.
- Bei SIMOTION MCC: auf die Befehle Modul oder Kommentarblock.
- Bei SIMOTION KOP/FUP: innerhalb eines Netzwerks.
- An Codestellen, in denen bereits weitere Debug-Punkte (z. B. Triggerpunkte) gesetzt sind.

Die Debug-Punkte in allen Programmquellen des SIMOTION Geräts können Sie in der Debug-Tabelle auflisten:

- Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  für "Debug-Tabelle".

In der Debug-Tabelle können Sie auch alle Haltepunkte (in allen Programmquellen) des SIMOTION-Geräts entfernen:

- Klicken Sie in der auf die Schaltfläche für "Alle Haltepunkte löschen".

Gesetzte Haltepunkte bleiben auch nach Verlassen des Betriebsmodus "Debug-Modus" gespeichert, sie werden nur im Debug-Modus angezeigt.

Sie können die Diagnosefunktionen Status Programm (Seite 342) und Haltepunkte gemeinsam in einer Programmquelle bzw. POE verwenden. Es gelten jedoch folgende Einschränkungen in Abhängigkeit von den Programmiersprachen:

- SIMOTION ST: Bei Version V3.2 des SIMOTION Kernels dürfen die mit Status Programm zu testenden (markierten) Zeilen der ST-Quelle keinen Haltepunkt enthalten.
- SIMOTION MCC und KOP/FUP: Die mit Status Programm zu testenden Befehle des MCC-Charts (bzw. Netzwerke des KOP/FUP-Programms) dürfen keinen Haltepunkt enthalten.

Weiteres Vorgehen

1. Legen Sie den Aufrufpfad fest, siehe Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354).
2. Aktivieren Sie die Haltepunkte, siehe Haltepunkte aktivieren (Seite 361).

7.2.8.7 Funktionsleiste Haltepunkte

Diese Funktionsleiste enthält wichtige Bedienhandlungen zum Setzen und Aktivieren von Haltepunkten:

Tabelle 7-14 Funktionsleiste Haltepunkte

Symbol	Bedeutung
	Haltepunkt setzen / entfernen Klicken Sie auf dieses Symbol, um an der ausgewählten Codestelle einen Haltepunkt zu setzen bzw. einen vorhandenen Haltepunkt zu entfernen. Siehe: Haltepunkte setzen (Seite 351).
	Haltepunkt aktivieren / deaktivieren Klicken Sie auf dieses Symbol, um den Haltepunkt an der ausgewählten Codestelle zu aktivieren bzw. zu deaktivieren. Siehe: Haltepunkte aktivieren (Seite 361).
	Aufrufpfad bearbeiten Klicken Sie auf dieses Symbol, um den Aufrufpfad für Haltepunkte festzulegen: • Wenn eine Codestelle **mit** Haltepunkt ausgewählt ist: den Aufrufpfad für diesen Haltepunkt. • Wenn eine Codestelle **ohne** Haltepunkt ausgewählt wird, den Aufrufpfad für alle Haltepunkte der POE. Siehe: Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354), Aufrufpfad für alle Haltepunkte festlegen (Seite 358).
	Alle Haltepunkte der aktiven POE aktivieren Klicken Sie auf dieses Symbol, um in der aktiven Programmquelle bzw. POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) alle Haltepunkte zu aktivieren. Siehe: Haltepunkte aktivieren (Seite 361).
	Alle Haltepunkte der aktiven POE deaktivieren Klicken Sie auf dieses Symbol, um in der aktiven Programmquelle bzw. POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) alle Haltepunkte zu deaktivieren. Siehe: Haltepunkte aktivieren (Seite 361).

Symbol	Bedeutung
	Alle Haltepunkte der aktiven POE entfernen Klicken Sie auf dieses Symbol, um in der aktiven Programmquelle bzw. POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) alle Haltepunkte zu entfernen. Siehe: Haltepunkte setzen (Seite 351).
	Debug-Tabelle Klicken Sie auf dieses Symbol, um die Debug-Tabelle anzuzeigen. Siehe: Parameter Debug-Tabelle (Seite 351).
	Call Stack anzeigen Klicken Sie auf dieses Symbol, um nach Erreichen eines aktivierten Haltepunktes: • den Aufrufpfad am aktuellen Haltepunkt ansehen, • die Codestellen ansehen, an denen die anderen Tasks der Debug-Taskgruppe angehalten wurden, sowie deren Aufrufpfad. Siehe: Call-Stack anzeigen (Seite 363).
	Fortsetzen Klicken Sie auf dieses Symbol, um nach Erreichen eines aktivierten Haltepunktes den Programmablauf fortzusetzen. Siehe: Programmablauf fortsetzen (Seite 365), Call-Stack anzeigen (Seite 363).
	Nächster Schritt (ab Version V4.4 des SIMOTION Kernels) Nur verfügbar bei den Programmiersprachen MCC und KOP/FUP. Klicken Sie auf dieses Symbol, um den Programmablauf fortzusetzen, bis der nächste MCC-Befehl bzw. KOP/FUP-Netzwerk erreicht ist. Siehe: Programmablauf schrittweise fortsetzen.
	Unterprogramm durchsteppen (ab Version V4.4 des SIMOTION Kernels) Nur verfügbar bei der Programmiersprache MCC. Klicken Sie auf dieses Symbol, um in das aufgerufene Unterprogramm zu springen und dort am ersten Befehl anzuhalten. Das Unterprogramm muss in den Programmiersprachen MCC oder KOP/FUP erstellt sein. Siehe: Programmablauf schrittweise fortsetzen.

7.2.8.8 Aufrufpfad für einen einzelnen Haltepunkt festlegen

Voraussetzungen:

1. Die Programmquelle mit der POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) ist geöffnet.
2. Eine Verbindung mit dem Zielsystem muss bestehen (Online-Modus).
3. SIMOTION SCOUT befindet sich für das betreffende SIMOTION-Gerät im Debug-Modus, siehe Debug-Modus einstellen (Seite 347).
4. Die anzuhaltenden Tasks sind festgelegt, siehe Debug-Taskgruppe festlegen (Seite 348).
5. Haltepunkt ist gesetzt, siehe Haltepunkte setzen (Seite 351).

Vorgehensweise

So legen Sie den Aufrufpfad für einen einzelnen Haltepunkt fest:

1. Wählen Sie eine Codestelle aus, an der bereits ein Haltepunkt gesetzt ist:
   - SIMOTION ST: Setzen Sie den Cursor in eine entsprechende Zeile der ST-Quelle.
   - SIMOTION MCC: Markieren Sie einen entsprechenden Befehl im MCC-Chart.
   - SIMOTION KOP/FUP: Setzen Sie den Cursor in ein entsprechendes Netzwerk des KOP/FUP-Programms.
2. Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  für "Aufrufpfad bearbeiten".
   Im Fenster "Aufrufpfad/Taskauswahl Haltepunkt" wird die markierte Codestelle angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE).
3. Wählen Sie die Task aus, in der das Anwenderprogramm (d. h. alle Tasks in der Debug-Taskgruppe) angehalten wird, wenn der ausgewählte Haltepunkt erreicht wird.
   Zur Auswahl stehen:
   - **alle Aufrufstellen ab dieser Aufrufebene**
     Das Anwenderprogramm wird immer angehalten, wenn der aktivierte Haltepunkt in einer beliebigen Task der Debug-Taskgruppe erreicht wird.
   - die einzelnen Tasks, von denen der ausgewählte Haltepunkt erreicht werden kann.
     Das Anwenderprogramm wird nur dann angehalten, wenn der Haltepunkt in der ausgewählten Task erreicht wird. Die Task muss sich in der Debug-Taskgruppe befinden.
     Die Angabe eines Aufrufpfads ist möglich.
4. Nur bei Funktionen und Funktionsbausteinen: Wählen Sie den Aufrufpfad aus, d. h. die aufrufende Codestelle (in der aufrufenden POE).
   Zur Auswahl stehen:
   - **alle Aufrufstellen ab dieser Aufrufebene**
     Es wird **kein** Aufrufpfad angegeben. Das Anwenderprogramm wird immer am aktivierten Haltepunkt angehalten, wenn die POE in den ausgewählten Tasks aufgerufen wird.
   - Nur wenn eine einzelne Task ausgewählt ist: die aufrufenden Codestellen innerhalb der ausgewählten Task (mit Name der Programmquelle, Zeilennummer, Name der POE).
     Der Aufrufpfad wird angegeben. Das Anwenderprogramm wird nur dann am aktivierten Haltepunkt angehalten, wenn die POE von der ausgewählten Codestelle aufgerufen wird.
     Wenn die POE der ausgewählten aufrufenden Codestelle wiederum von anderen Codestellen aufgerufen wird, werden sukzessive weitere Zeilen eingeblendet, in denen Sie ähnlich vorgehen.
5. Wenn der Haltepunkt erst nach mehrmaligem Erreichen der Codestelle aktiviert werden soll, wählen Sie die entsprechende Anzahl.

Hinweis

Den Aufrufpfad zu den einzelnen Haltepunkten können Sie auch in der Debug-Tabelle festlegen:

1. Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  für "Debug-Tabelle". Das Fenster "Debug-Tabelle" öffnet.
2. Klicken Sie auf die entsprechende Schaltfläche in der Spalte "Aufrufpfad".
3. Gehen Sie weiter wie oben beschrieben vor:
   - Legen Sie die Task fest
   - Legen Sie den Aufrufpfad fest (nur bei Funktionen und Funktionsbausteinen)
   - Legen Sie die Anzahl der Durchläufe fest, nach denen der Haltepunkt aktiviert werden soll.

Weiteres Vorgehen:

- Aktivieren Sie die Haltepunkte, siehe Haltepunkte aktivieren (Seite 361).

Hinweis

Den Aufrufpfad an einem aktuellen Haltepunkt sowie die Codestellen, an denen die anderen Tasks der Debug-Taskgruppe angehalten wurden, können Sie mit der Funktion "Call Stack anzeigen (Seite 363)" einsehen.

Siehe auch

Aufrufpfad für alle Haltepunkte festlegen (Seite 358)

7.2.8.9 Parameter Aufrufpfad/Taskauswahl Haltepunkt

Tabelle 7-15 Parameterbeschreibung Aufrufpfad/Taskauswahl Haltepunkt

Feld	Beschreibung
Selektierte CPU	Das ausgewählte SIMOTION Gerät wird angezeigt.
Aufrufende Task	Wählen Sie die Task aus, in der das Anwenderprogramm (d. h. alle Tasks in der Debug-Taskgruppe) angehalten wird, wenn der ausgewählte Haltepunkt erreicht wird. Zur Auswahl stehen: • **alle Aufrufstellen ab dieser Aufrufebene** Das Anwenderprogramm wird immer angehalten, wenn der aktivierte Haltepunkt in einer beliebigen Task der Debug-Taskgruppe erreicht wird. • die einzelnen Tasks, von denen die POE mit dem ausgewählten Haltepunkt erreicht werden kann. Das Anwenderprogramm wird nur dann angehalten, wenn der Haltepunkt in der ausgewählten Task erreicht wird. Die Task muss sich in der Debug-Taskgruppe befinden. Die Angabe eines Aufrufpfads ist möglich.
Aktuelle Codestelle	Die Codestelle mit dem gesetzten Haltepunkt wird angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE)
wird aufgerufen von	Nur bei Funktionen und Funktionsbausteinen. Wählen Sie den Aufrufpfad der POE aus, d. h. die aufrufende Codestelle (in der aufrufenden POE). Zur Auswahl stehen: • **alle Aufrufstellen ab dieser Aufrufebene** Es wird **kein** Aufrufpfad angegeben. Das Anwenderprogramm wird immer am aktivierten Haltepunkt angehalten, wenn die POE in den ausgewählten Tasks erreicht wird. • Nur wenn eine einzelne Task ausgewählt ist: Die aufrufenden Codestellen innerhalb der ausgewählten Task (mit Name der Programmquelle, Zeilennummer, Name der POE). Der Aufrufpfad wird angegeben. Das Anwenderprogramm wird nur dann am aktivierten Haltepunkt angehalten, wenn die POE von der ausgewählten Codestelle aufgerufen wird. Wenn die POE der ausgewählten aufrufenden Codestelle wiederum von anderen Codestellen aufgerufen wird, werden sukzessive weitere Zeilen eingeblendet, in denen Sie ähnlich vorgehen.
Der Haltepunkt wird jeweils zum …ten Durchlauf aktiviert	Wählen Sie die entsprechende Anzahl, wenn der Haltepunkt erst nach mehrmaligem Erreichen der Codestelle aktiviert werden soll.

Hinweis

Änderungen an der Debug-Taskgruppe können Sie nur vornehmen, wenn keine Haltepunkte aktiviert sind.

7.2.8.10 Aufrufpfad für alle Haltepunkte festlegen

Mit diesem Vorgehen können Sie:

- für alle zukünftigen Haltepunkte in einer POE (z. B. MCC-Chart, KOP/FUP-Programm oder POE in einer ST-Quelle) eine Defaulteinstellung wählen.
- für alle bisher gesetzten Haltepunkte in dieser POE den Aufrufpfad übernehmen und angleichen.

Voraussetzungen

1. Die Programmquelle mit der POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) ist geöffnet.
2. Eine Verbindung mit dem Zielsystem muss bestehen (Online-Modus).
3. SIMOTION SCOUT befindet sich für das betreffende SIMOTION-Gerät im Debug-Modus, siehe Debug-Modus einstellen (Seite 347).
4. Die anzuhaltenden Tasks sind festgelegt, siehe Debug-Taskgruppe festlegen (Seite 348).

Vorgehensweise

So legen Sie den Aufrufpfad für alle zukünftigen Haltepunkte in einer POE fest:

1. Wählen Sie eine Codestelle aus, an der **kein** Haltepunkt gesetzt ist:
   - SIMOTION ST: Setzen Sie den Cursor in eine entsprechende Zeile der ST-Quelle.
   - SIMOTION MCC: Markieren Sie einen entsprechenden Befehl im MCC-Chart.
   - SIMOTION KOP/FUP: Setzen Sie den Cursor in ein entsprechendes Netzwerk des KOP/FUP-Programms
2. Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  für "Aufrufpfad bearbeiten".
   Im Fenster "Aufrufpfad/Taskauswahl alle Haltepunkte je POE" wird die markierte Codestelle angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE.
3. Wählen Sie die Task aus, in der das Anwenderprogramm (d. h. alle Tasks in der Debug-Taskgruppe) angehalten wird, wenn ein Haltepunkt in dieser POE erreicht wird.
   Zur Auswahl stehen:
   - **alle Aufrufstellen ab dieser Aufrufebene**
     Das Anwenderprogramm wird immer angehalten, wenn ein aktivierter Haltepunkt der POE in einer beliebigen Task der Debug-Taskgruppe erreicht wird.
   - die einzelnen Tasks, von denen der ausgewählte Haltepunkt erreicht werden kann.
     Das Anwenderprogramm wird nur dann angehalten, wenn ein Haltepunkt in der ausgewählten Task erreicht wird. Die Task muss sich in der Debug-Taskgruppe befinden.
     Die Angabe eines Aufrufpfads ist möglich.

4. Nur bei Funktionen und Funktionsbausteinen: Wählen Sie den Aufrufpfad aus, d. h. die aufrufende Codestelle (in der aufrufenden POE).
   Zur Auswahl stehen:
   - **alle Aufrufstellen ab dieser Aufrufebene**
     Es wird kein Aufrufpfad angegeben. Das Anwenderprogramm wird immer an einem aktivierten Haltepunkt angehalten, wenn die POE in den ausgewählten Tasks aufgerufen wird.
   - Nur wenn eine einzelne Task ausgewählt ist: die aufrufenden Codestellen innerhalb der ausgewählten Task (mit Name der Programmquelle, Zeilennummer, Name der POE). Der Aufrufpfad wird angegeben. Das Anwenderprogramm wird nur dann an einem aktivierten Haltepunkt angehalten, wenn die POE von der ausgewählten Codestelle aufgerufen wird.
     Wenn die ausgewählte aufrufende Codestelle wiederum von anderen Codestellen aufgerufen wird, werden sukzessive weitere Zeilen eingeblendet, in denen Sie ähnlich vorgehen.
5. Wenn ein Haltepunkt erst nach mehrmaligem Erreichen der Codestelle aktiviert werden soll, wählen Sie die entsprechende Anzahl der Durchläufe.
6. Wenn Sie diesen Aufrufpfad für alle bisher gesetzten Haltepunkte in dieser POE übernehmen und angleichen wollen:
   - Klicken Sie auf **Übernehmen**.

Weiteres Vorgehen:

- Aktivieren Sie die Haltepunkte, siehe Haltepunkte aktivieren (Seite 361).

Hinweis

Den Aufrufpfad an einem aktuellen Haltepunkt sowie die Codestellen, an denen die anderen Tasks der Debug-Taskgruppe angehalten wurden, können Sie mit der Funktion "Call Stack anzeigen (Seite 363)" einsehen.

Siehe auch

Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354)

7.2.8.11 Parameter Aufrufpfad/Taskauswahl alle Haltepunkte je POE

Hier legen Sie eine Voreinstellung für den Aufrufpfad aller zukünftig zu setzenden Haltepunkte in einer POE fest. Außerdem können Sie diese Einstellung auch für alle bisher gesetzten Haltepunkte dieser POE übernehmen.

Tabelle 7-16 Parameterbeschreibung Aufrufpfad/Taskauswahl alle Haltepunkte je POE

Feld	Beschreibung
Selektierte CPU	Das ausgewählte SIMOTION Gerät wird angezeigt.
Aufrufende Task	Wählen Sie die Task aus, in der das Anwenderprogramm (d. h. alle Tasks in der Debug-Taskgruppe) angehalten wird, wenn ein Haltepunkt in dieser POE erreicht wird. Zur Auswahl stehen: • **alle Aufrufstellen ab dieser Aufrufebene** Das Anwenderprogramm wird immer angehalten, wenn ein aktivierter Haltepunkt der POE in einer beliebigen Task der Debug-Taskgruppe erreicht wird. • die einzelnen Tasks, von denen die POE erreicht werden kann. Das Anwenderprogramm wird nur dann angehalten, wenn ein aktivierter Haltepunkt in der ausgewählten Task erreicht wird. Die Task muss sich in der Debug-Taskgruppe befinden. Die Angabe eines Aufrufpfads ist möglich.
Aktuelle POE	Die POE, in der sich Cursor befindet, wird angezeigt (mit Name der Programmquelle, Name der POE)
wird aufgerufen von	Nur bei Funktionen und Funktionsbausteinen. Wählen Sie den Aufrufpfad der POE aus, d. h. die aufrufende Codestelle (in der aufrufenden POE). Zur Auswahl stehen: • **alle Aufrufstellen ab dieser Aufrufebene** Es wird **kein** Aufrufpfad angegeben. Das Anwenderprogramm wird immer an einem aktivierten Haltepunkt angehalten, wenn die POE in den ausgewählten Tasks aufgerufen wird. • Nur wenn eine einzelne Task ausgewählt ist: Die aufrufenden Codestellen innerhalb der ausgewählten Task (mit Name der Programmquelle, Zeilennummer, Name der POE). Der Aufrufpfad wird angegeben. Das Anwenderprogramm wird nur dann an einem aktivierten Haltepunkt angehalten, wenn die POE von der ausgewählten Codestelle aufgerufen wird. Wenn die POE der ausgewählten aufrufenden Codestelle wiederum von anderen Codestellen aufgerufen wird, werden sukzessive weitere Zeilen eingeblendet, in denen Sie ähnlich vorgehen.
Der Haltepunkt wird jeweils zum …ten Durchlauf aktiviert	Wählen Sie die entsprechende Anzahl, wenn der Haltepunkt erst nach mehrmaligem Erreichen der Codestelle aktiviert werden soll.
Diesen Aufrufpfad für alle bisherigen Haltepunkte dieser POE übernehmen	Klicken Sie auf die Schaltfläche **Übernehmen**, wenn Sie den Aufrufpfad für alle bisher gesetzten Haltepunkte der aktuellen POE übernehmen wollen. Vorhandene Einstellungen werden überschrieben.

7.2.8.12 Haltepunkte aktivieren

Sie müssen die Haltepunkte aktivieren, damit sie auf den Programmablauf wirken.

Voraussetzungen

1. Die Programmquelle mit der POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) ist geöffnet.
2. Eine Verbindung mit dem Zielsystem muss bestehen (Online-Modus).
3. SIMOTION SCOUT befindet sich für das betreffende SIMOTION-Gerät im Debug-Modus, siehe Debug-Modus einstellen (Seite 347).
4. Die anzuhaltenden Tasks sind festgelegt, siehe Debug-Taskgruppe festlegen (Seite 348).
5. Haltepunkte sind gesetzt, siehe Haltepunkte setzen (Seite 351).
6. Aufrufpfade sind festgelegt, siehe Aufrufpfad für einen einzelnen Haltepunkt festlegen (Seite 354).

Haltepunkte aktivieren

So aktivieren Sie einen einzelnen Haltepunkt:

1. Wählen Sie eine Codestelle aus, an der bereits ein Haltepunkt gesetzt ist:
   - SIMOTION ST: Setzen Sie den Cursor in eine entsprechende Zeile der ST-Quelle.
   - SIMOTION MCC: Markieren Sie einen entsprechenden Befehl im MCC-Chart.
   - SIMOTION KOP/FUP: Setzen Sie den Cursor in ein entsprechendes Netzwerk des KOP/FUP-Programms.
2. Führen Sie folgende Aktion aus (Alternativen):
   - Wählen Sie Menü **Debug > Haltepunkt aktivieren / deaktivieren** (Tastenkürzel F12).
   - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

So aktivieren Sie alle Haltepunkte (in allen Programmquellen) des SIMOTION Geräts:

- Führen Sie folgende Aktion aus (Alternativen):
  - Wählen Sie Menü **Debug > Alle Haltepunkte aktivieren**.
  - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

Nach Aktivieren des ersten Haltepunkts schaltet das SIMOTION-Gerät in den Debug-Modus. Dort verbleibt es bis zum Deaktivieren des letzten Haltepunkts.

In der Funktionsleiste Taskstatus (Seite 365) werden die Tasks mit aktivierten Haltepunkten grau ( ) hinterlegt angezeigt.

Hinweis

Haltepunkte aller Programmquellen des SIMOTION-Geräts können Sie in der Debug-Tabelle aktivieren und deaktivieren:

1. Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche für "Debug-Tabelle". Das Fenster "Debug-Tabelle" öffnet.
2. Führen Sie nachstehende Aktion aus, abhängig davon, welche Haltepunkte Sie aktivieren oder deaktivieren wollen:
   - Einzelne Haltepunkte: Aktivieren oder deaktivieren Sie die entsprechende Checkbox.
   - Alle Haltepunkte (in allen Programmquellen): Klicken Sie auf die entsprechende Schaltfläche.

Bis Version V4.3 des SIMOTION Kernels gilt:

- Bei aktivierten Haltepunkten kann die Testfunktion "Einzelschritt" der Programmiersprache SIMOTION MCC nicht verwendet werden.

Ab Version V4.4 des SIMOTION Kernels gilt:

- Die Testfunktion "Einzelschritt" der Programmiersprache SIMOTION MCC steht im Debug-Modus nicht zur Verfügung.

Haltepunkte können Sie nicht aktivieren, wenn die Steuerungshoheit bei der Achssteuertafel liegt. Umgekehrt können Sie die Steuerungshoheit nicht für die Achssteuertafel holen, wenn ein Haltepunkt aktiviert ist.

Verhalten am aktivierten Haltepunkt

Bei Erreichen eines aktivierten Haltepunkts (gegebenenfalls über den ausgewählten Aufrufpfad (Seite 354)) werden die Tasks angehalten, die der Debug-Taskgruppe zugeordnet sind. Das Verhalten ist abhängig von den Tasks in der Debug-Taskgruppe und ist in "Debug-Taskgruppe festlegen (Seite 348)" beschrieben. Der Haltepunkt wird besonders markiert.

In der Funktionsleiste Taskstatus (Seite 365) wird die Task, in der der Haltepunkt erreicht wurde, rot ( ) hinterlegt angezeigt.

Für die Programmiersprachen MCC bzw. KOP/FUP gilt: Wird die Debug-Taskgruppe duch einen Haltepunkt gestoppt, so hat der Anwender die Möglichkeit, in der Combo-Box auf eine andere, zur Debug-Taskgruppe gehörende Task zu wechseln. Es wird immer die Haltestelle der aktuell ausgewählten Task visualisiert.

Wenn sich der Haltepunkt, der das Anhalten der Tasks ausgelöst hat, in einem Programm oder Funktionsbaustein befindet, werden die Werte der statischen Variablen dieser POE im Register "Status Variablen" der Detailanzeige angezeigt. Temporäre Variablen (auch Durchgangsparameter bei Funktionsbausteinen) werden nicht angezeigt. Statische Variablen anderer POE oder Unit-Variablen können Sie im Symbol-Browser (Seite 332) beobachten.

Mit der Funktion "Call-Stack anzeigen (Seite 363)" können Sie:

- den Aufrufpfad am aktuellen Haltepunkt ansehen,
- die Codestellen mit Aufrufpfad ansehen, an denen die anderen Tasks der Debug-Taskgruppe angehalten wurden.

Programmablauf fortsetzen

Die Ausführung der angehaltenen Tasks können Sie fortsetzen, siehe "Programmablauf fortsetzen" (Seite 365).

Ab Version V4.4 des SIMOTION Kernels können Sie in den Programmiersprachen MCC und KOP/FUP die Task, die am aktivierten Haltepunkt angehalten wurde, schrittweise fortsetzen, siehe Programm schrittweise fortsetzen.

Haltepunkte deaktivieren

So deaktivieren Sie einen einzelnen Haltepunkt:

1. Wählen Sie die Codestelle mit dem aktivierten Haltepunkt aus.
2. Führen Sie folgende Aktion aus (Alternativen):
   - Wählen Sie Menü **Debug > Haltepunkt aktivieren / deaktivieren** (Tastenkürzel F12).
   - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

So deaktivieren Sie alle Haltepunkte (in allen Programmquellen) des SIMOTION Geräts:

- Führen Sie folgende Aktion aus (Alternativen):
  - Wählen Sie Menü **Debug > Alle Haltepunkte deaktivieren**.
  - Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  .

Nach dem Deaktivieren des letzten Haltepunkts schaltet das SIMOTION-Gerät in den Betriebsmodus "Testbetrieb"; SIMOTION SCOUT bleibt weiter im Debug-Modus.

7.2.8.13 Call-Stack anzeigen

Mit der Funktion "Call-Stack anzeigen" können Sie:

- den Aufrufpfad am aktuellen Haltepunkt ansehen,
- die Codestellen mit Aufrufpfad ansehen, an denen die anderen Tasks der Debug-Taskgruppe angehalten wurden.

Voraussetzung

Das Anwenderprogramm steht an einem aktivierten Haltepunkt, d. h., die Tasks der Debug-Taskgruppe (Seite 348) sind angehalten.

Vorgehensweise

So rufen Sie die Funktion "Call-Stack anzeigen" auf:

- Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche  für "Call-Stack anzeigen".
  Es öffnet sich das Dialogfenster "Call-Stack Haltepunkt". Der aktuelle Aufrufpfad (einschließlich der aufrufenden Task und Anzahl der eingestellten Durchläufe) wird angezeigt.
  Der Aufrufpfad kann nicht verändert werden.

So wenden Sie die Funktion "Call-Stack anzeigen" an:

1. Lassen Sie das Dialogfenster "Call-Stack Haltepunkt" weiter geöffnet.
2. So zeigen Sie die Codestelle an, bei der eine andere Task angehalten wurde:
   – Wählen Sie die entsprechende Task. Zur Auswahl stehen alle Tasks der Debug-Taskgruppe.

   Die Codestelle einschließlich Aufrufpfad wird angezeigt. Falls die Codestelle in einem Anwenderprogramm enthalten ist, wird die betreffende Programmquelle mit der POE (z. B. ST-Quelle, MCC-Chart, KOP/FUP-Programm) geöffnet und die Codestelle markiert.
3. So setzen Sie den Programmablauf wieder fort:
   – Klicken Sie in der Funktionsleiste Haltepunkte auf die Schaltfläche für "Fortsetzen" (Tastenkürzel STRG+F8).

   Beim Erreichen des nächsten aktivierten Haltepunkts werden die Tasks der Debug-Taskgruppe erneut angehalten. Der aktuelle Aufrufpfad einschließlich der aufrufenden Task wird angezeigt.
4. Das Dialogfenster "Call-Stack Haltepunkt" schließen Sie mit **OK**.

Für Namen der Programmquellen des SIMOTION RT siehe Tabelle in "Programm-Durchlauf (Seite 339)".

7.2.8.14 Parameter Callstack Haltepunkte

Wenn ein aktivierter Haltepunkt (Seite 361) erreicht ist, können Sie sich für jede Task in der Debug-Taskgruppe (Seite 348) anzeigen lassen:

- die Stelle im Programmcode (z. B. Zeile einer ST-Quelle) an der die Task angehalten wurde,
- den Aufrufpfad dieser Codestelle.

Tabelle 7-17 Parameterbeschreibung Aufrufpfad Haltepunkt

Feld	Beschreibung
Selektierte CPU	Das ausgewählte SIMOTION Gerät wird angezeigt.
Aufrufende Task	Wählen Sie die Task aus, für die Sie die Codestelle ansehen wollen, an der die Task angehalten wurde. Zur Auswahl stehen alle Tasks des Debug-Taskgruppe.
Aktuelle Codestelle	Die Stelle des Programmcodes (z. B. Zeile einer ST-Quelle), an der die ausgewählte Task angehalten wurde, wird angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE).
wird aufgerufen von	Die Codestellen, welche die aktuelle Codestelle innerhalb der ausgewählten Task aufrufen, werden rekursiv angezeigt (mit Name der Programmquelle, Zeilennummer, Name der POE, ggf. Name der Instanz eines Funktionsbausteins).

Für Namen der Programmquellen des SIMOTION RT siehe Tabelle in "Programm-Durchlauf (Seite 339)".

7.2.8.15 Programmablauf fortsetzen

So setzen Sie den Programmablauf wieder fort:

- Führen Sie folgende Aktion aus (Alternativen):
  - Wählen Sie Menü **Debug > Fortsetzen** (Tastenkürzel STRG+F8).
  - Klicken Sie in der Funktionsleiste Haltepunkte (Seite 353) auf die Schaltfläche  für "Fortsetzen".

Die angehalteten Tasks werden fortgesetzt, bis der nächste aktive Haltepunkt erreicht ist

7.2.9 Funktionsleiste Taskstatus

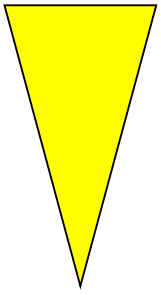
Die Funktionsleiste Taskstatus zeigt in einer Combo-Box alle Tasks des aktiven SIMOTION Geräts an, denen ein Programm zugeordnet ist.

Sie wird unter folgenden Bedingungen angezeigt:

1. SIMOTION SCOUT ist im Online-Modus.
2. Das betreffende SIMOTION Geräts ist aktiv, z. B.
   - Im Projektnavigator ist das SIMOTION Gerät oder ein Element in dessen Teilbaum (z. B. Programmquelle, Technologieobjekt) ausgewählt.
   - Im Arbeitsbereich ist ein geöffnetes Fenster aktiv, das zu einem Element im Teilbaum des SIMOTION Geräts gehört.
3. Das SIMOTION Gerät ist konsistent.

Eine Hintergrundfärbung zeigt das Eintreten bestimmter Ereignisse in der betreffenden Task an, siehe nachfolgende Tabelle. Die betreffende Task wird gemäß der Rangfolge des Ereignisses in der Combo-Box der Funktionsleiste angezeigt.

Tabelle 7-18 Bedeutung der Hintergrundfarben in der Funktionsleiste Taskstatus

Hintergrundfarbe	Bedeutung	Rangfolge
Cyan	Die betreffende Tasks wartet bei der Testfunktion "Einzelschritt" an einem Befehl (nur bei Programmiersprache SIMOTION MCC).	Höchste  Niederste
Rot	Die betreffende Task steht an einem Haltepunkt (Seite 361).	
Blau	In der betreffenden Task ist die Testfunktion "Einzelschritt" eingeschaltet (nur bei Programmiersprache SIMOTION MCC)	
Grau	In der betreffenden Task ist mindestens 1 Haltepunkt (Seite 361) aktiviert.	
Gelb	In der betreffenden Task ist die Testfunktion "Beobachten" eingeschaltet (nur Programmiersprache SIMOTION MCC).	
Weiß	In der betreffenden Task ist keine der oben genannten Testfunktionen eingeschaltet.	

Hinweis

Eine Auswahl einer Task in der Combo-Box ist nur nur möglkich:

- bei folgenden Testfunktionen der Programmiersprache SIMOTION MCC möglich:
  - Beobachten
  - Einzelschritt
  - Leuchtspur
- bei aktivierten Haltepunkten (Seite 361) in den Programmiersprachen MCC oder KOP/FUP.

7.2.10 Trace

Mit dem **Trace-Tool** können Sie zeitliche Verläufe von Variablenwerten (z. B. Unit-Variablen, lokale Variablen, Systemvariablen, I/O-Variablen) aufzeichnen und speichern. Damit können Sie die Optimierung, beispielsweise von Achsen, dokumentieren.

Sie können die Aufzeichnungsdauer einstellen, bis vier Kanäle anzeigen lassen, Triggerbedingungen wählen, zeitliche Verschiebungen parametrieren, zwischen verschiedenen Kurvendarstellungen und Skalierungen wählen usw.

Neben der taktsynchronen Aufzeichnung können Sie auch **Aufzeichnung an Codestelle** wählen. Damit können Sie die Werte von Variablen immer dann aufzeichnen, wenn das Programm eine bestimmte Stelle der ST-Quelle durchläuft.

Das Trace-Tool ist ausführlich in der Online-Hilfe beschrieben.

7.2.11 Projektvergleich

Für SIMOTION SCOUT steht die Funktionalität **Projektvergleich** (Start über den Button **Starte Objektvergleich** ) zur Verfügung, um Objekte innerhalb desselben Projektes bzw. mit Objekten aus anderen Projekten (online oder offline) miteinander zu vergleichen.

Der Projektvergleich bietet Ihnen die Möglichkeit, die Unterschiede zu ermitteln und bei Bedarf eine Datenübernahme durchzuführen, um die Unterschiede zu beheben.

Objekte sind Geräte und deren Unterobjekte, sowie Programme, Technologieobjekte (TOs) oder Drive-Objekte (DOs) und Bibliotheken. Durch den Projektvergleich werden Sie im Service-Fall an der Anlage unterstützt.

Weitere Informationen zum Projekt- und Detailvergleich finden Sie im Funktionshandbuch SIMOTION Projektvergleich.

Anhang

A

A.1 Formale Sprachbeschreibung

In diesem Kapitel finden Sie Übersichten zu den Grundelementen von ST und eine vollständige Zusammenstellung aller Syntaxdiagramme mit den Sprachelementen. Dieser Anhang stellt eine Kurzfassung der Grundlagen der Sprache ST dar.

A.1.1 Hilfen für die Sprachbeschreibung

Basis für die Sprachbeschreibung in den einzelnen Kapiteln sind Syntaxdiagramme. Sie geben Ihnen einen guten Einblick in den syntaktischen (d. h. grammatikalischen) Aufbau von ST.

Wie Sie Syntaxdiagramme verwenden, haben Sie in *Hilfen für die Sprachbeschreibung* erfahren. Nachfolgend die für den fortgeschrittenen Anwender interessante Unterscheidung zwischen formatpflichtigen und formatfreien Regeln.

A.1.1.1 Formatpflichtige Regeln (lexikalische Regeln)

Die lexikalischen Regeln beschreiben die Struktur der Elemente, die bei der Lexikalanalyse des Compilers bearbeitet werden. Daher ist die Schreibweise nicht formatfrei und die Regeln sind streng einzuhalten. Das bedeutet insbesondere:

- Einfügen von Formatierungszeichen ist nicht erlaubt.
- Block- und Zeilenkommentare können nicht eingefügt werden.
- Attribute zu Bezeichnern können nicht eingefügt werden.

Das folgende Bild zeigt Ihnen eine lexikalische Regel, hier die der erlaubten Bezeichner.

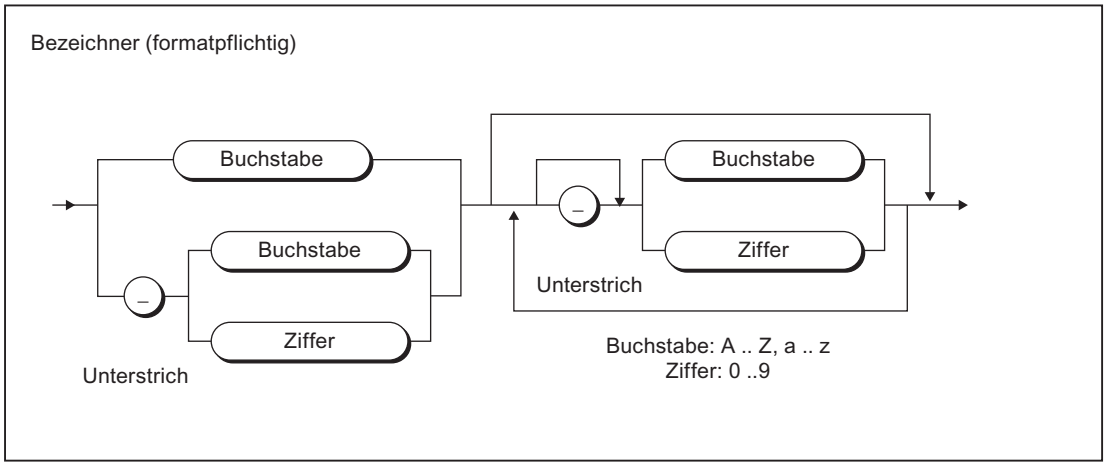


Bild A-1 Beispiel für eine lexikalische Regel

Gültige Beispiele nach dieser dargestellten Regel wären:

```
R_REGLER3
_A_FELD
_100_3_3_10
```

A.1.1.2 Formatfreie Regeln (syntaktische Regeln)

Aufbauend auf den lexikalischen Regeln wird in den syntaktischen Regeln die Struktur von ST beschrieben. Im Rahmen dieser Regeln können Sie Ihr ST-Programm formatfrei erstellen.

Die Eigenschaft formatfrei bedeutet für Sie:

- Einfügen von Formatierungszeichen ist überall möglich.
- Block- und Zeilenkommentare können eingefügt werden.

Das folgende Beispiel zeigt Ihnen eine syntaktische Regel, hier die der Wertzuweisung, in einer Anweisung.

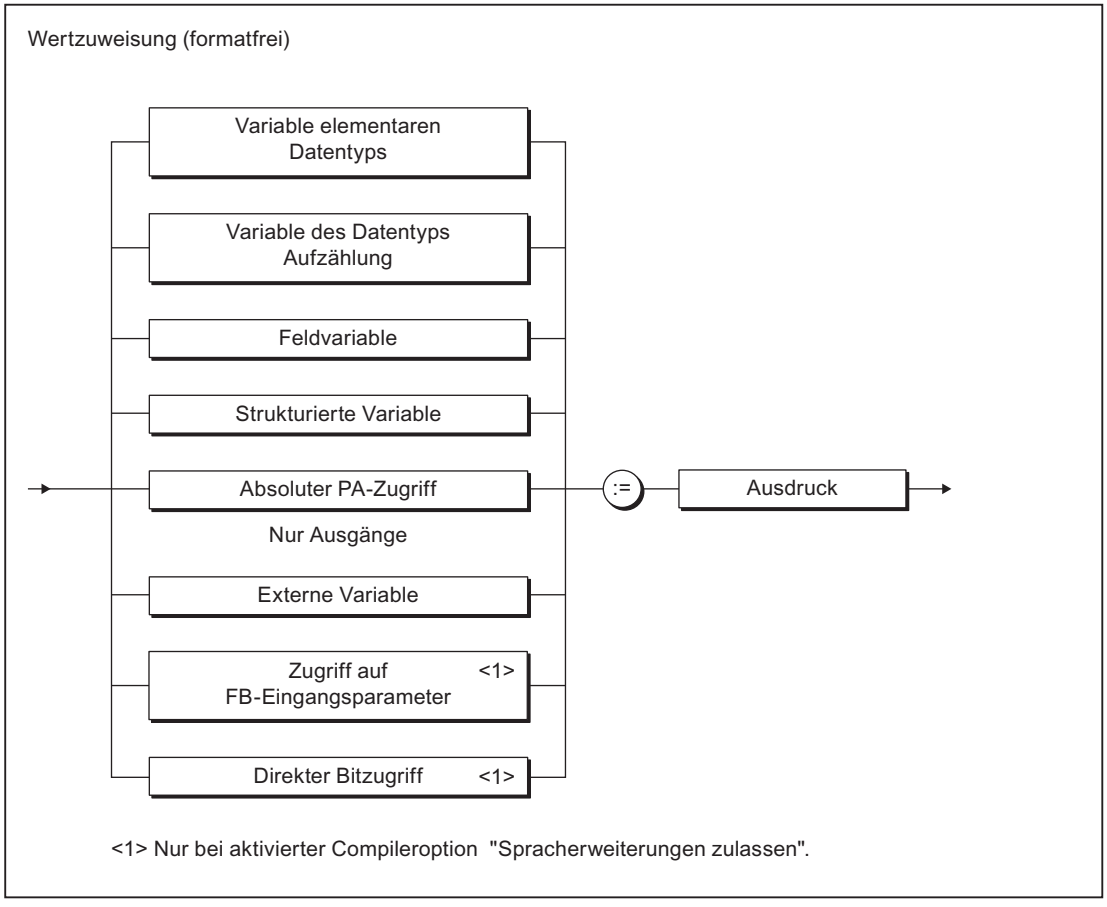


Bild A-2 Beispiel für eine syntaktische Regel

Gültige Beispiele nach dieser dargestellten Regel wären:

```
VARIABLE_1 := 100; SWITCH := FALSE;
//Dies ist ein Kommentar
VARIABLE_2 := 3.2 + VARIABLE_1;
```

A.1.2 Grundelemente (Terminale)

Ein Terminal ist ein Grundelement, das nicht durch eine weitere Regel, sondern verbal erklärt wird. In den Syntaxdiagrammen wird es als Oval oder Kreis dargestellt.

A.1.2.1 Buchstaben, Ziffern und sonstige Zeichen

Buchstaben und Ziffern sind die hauptsächlich verwendeten Zeichen. Der *Bezeichner* besteht z. B. aus Buchstaben, Ziffern und Unterstrich. Letzteres gehört zu den sonstigen Zeichen.

Tabelle A-1 Buchstaben und Ziffern

Zeichen	Untergruppe	Zeichensatzelemente
Buchstabe	Großbuchstabe	A .. Z
	Kleinbuchstabe	a .. z
Ziffer	Dezimalziffer	0 .. 9
Oktalziffer	Oktalziffer	0 .. 7
Hexadezimalziffer	Hexadezimalziffer	0 .. 9, A .. F, a .. f
Bit	Binärziffer	0, 1

Den vollständigen, erweiterten ASCII Zeichensatz können Sie in Kommentaren benutzen. Ab Dezimaläquivalent 32 (Leerzeichen) können Sie alle druckbaren Zeichen des ASCII-Code verwenden.

Für Sprachbefehle, Bezeichner, Konstanten, Ausdrücke und Operatoren können Sie sonstige Zeichen, d. h. Zeichen außer Buchstaben und Ziffern, nur nach bestimmten Regeln verwenden.

A.1.2.2 Formatierungs- und Trennzeichen in den Regeln

Formatierungs- und Trennzeichen werden in den formatpflichtigen (lexikalischen) und in den formatfreien (syntaktischen) Regeln unterschiedlich gebraucht. Die Unterschiede zwischen syntaktischen und lexikalischen Regeln haben Sie in Hilfen für die Sprachbeschreibung (Seite 367) erfahren.

In folgenden Tabellen finden Sie die Formatierungs- und Trennzeichen der lexikalischen und syntaktischen Regeln. Dazu erhalten Sie eine Beschreibung sowie alle Regeln, in denen die Formatierungs- und Trennzeichen als Terminale verwendet werden (siehe Regeln (Seite 381)).

Tabelle A-2 Formatierungs- und Trennzeichen in den lexikalischen Regeln

Zeichen	Beschreibung	Lexikalische Regel
:	Trennzeichen zwischen Stunden, Minuten und Sekunden	Tageszeitangabe
.	Trennzeichen für Gleitpunktzahl-Darstellung, Zeitintervalldarstellung, Absolute Adressierung	Gleitpunktzahl-Darstellung, Tageszeitangabe, Dezimaldarstellung, Zugriff auf lokale oder globale Instanz
_ Unterstrich	Trennzeichen für Bezeichner, Trennzeichen für Zahlenwerte in Konstanten	Bezeichner, Dezimalziffernfolge, Binärziffernfolge, Oktalziffernfolge, Hexadezimalziffernfolge, Stufendarstellung
%	Einleitung für Direktbezeichner beim CPU-Speicherzugriff	Einfacher Speicherzugriff
//	Kommentar	Zeilenkommentar
(**)	Kommentar	Blockkommentar

Tabelle A-3 Formatierungs- und Trennzeichen in den syntaktischen Regeln

Zeichen	Beschreibung	Syntaktische Regel
:	Trennzeichen zur Typangabe	Funktion, Variablendeklaration, Komponentendeklaration, CASE-Anweisung, Instanzdeklaration
;	Abschluss einer Vereinbarung oder Anweisung	Konstantenblock, Anweisung, Variablendeklaration, Instanzdeklaration, Komponentendeklaration, Anweisungsteil
,	Trennzeichen für Listen	Variablendeklaration, Feld-Initialisierungsliste, Instanzdeklaration, ARRAY-Datentyp Spezifikation, FB-Parameter, FC-Parameter, Wertliste
..	Bereichsangabe	Array-Datentyp Spezifikation, Wertliste
.	Strukturzugriff	strukturierte Variable
( )	Initialisierungsliste für Arrays, Klammerung in Ausdrücken, Aufruf Funktion und Funktionsbaustein	Feld-Initialisierungsliste, Ausdruck, Einfache Multiplikation, Operand, Exponent, FB-Aufruf, Funktionsaufruf
[ ]	Array-Vereinbarung, strukturierte Variable Teil Array	ARRAY-Datentyp Spezifikation

Siehe auch

Hilfen für die Sprachbeschreibung (Seite 98)

A.1.2.3 Formatierungs- und Trennzeichen für Konstanten

Nachfolgend sehen Sie alle Formatierungs- und Trennzeichen für Konstanten mit Angabe der lexikalischen Regel, in denen sie benutzt werden.

Tabelle A-4 Formatierungs- und Trennzeichen für Konstanten

Zeichen	Kennzeichen für	Lexikalische Regel
2#	Ganzzahl-Konstante	Binärziffernfolge
8#	Ganzzahl-Konstante	Oktalziffernfolge
16#	Ganzzahl-Konstante	Hexadezimalziffernfolge
E	Trennzeichen für Gleitpunktzahl-Konstante	Exponent
e	Trennzeichen für Gleitpunktzahl-Konstante	Exponent
D#	Zeitangabe	Datum
DATE#	Zeitangabe	Datum
DATE_AND_TIME#	Zeitangabe	Datum und Zeit
DT#	Zeitangabe	Datum und Zeit
T#	Zeitangabe	Zeitdauer
TIME#	Zeitangabe	Zeitdauer
TIME_OF_DAY#	Zeitangabe	Tageszeit
TOD#	Zeitangabe	Tageszeit
d	Trennzeichen für Zeitintervall (Day)	Tage (Regel Stufendarstellung)
h	Trennzeichen für Zeitintervall (Hours)	Stunden (Regel: Stufendarstellung)
m	Trennzeichen für Zeitintervall (Minutes)	Minuten (Regel: Stufendarstellung)
ms	Trennzeichen für Zeitintervall (Milliseconds)	Millisekunden (Regel: Stufendarstellung)
s	Trennzeichen für Zeitintervall (Seconds)	Sekunden (Regel: Stufendarstellung)

A.1.2.4 Vordefinierte Bezeichner für den Prozessabbild-Zugriff

Nachfolgend sehen Sie alle vordefinierten Variablen im ST, mit denen Sie auf CPU-Speicherbereiche zugreifen können (Absolutbezeichner). Beachten Sie, dass Sie Ausgänge lesen und beschreiben, Eingänge jedoch nur lesen dürfen.

Tabelle A-5 Absolutbezeichner

Bezeichner	Beschreibung	Lexikalische Regel
%In.x oder **%IXn.x**	CPU-Eingangsbereich mit Byte und Bitadresse	absoluter PA-Zugriff
%IBn	CPU-Eingangsbereich mit Byteadresse	absoluter PA-Zugriff

Bezeichner	Beschreibung	Lexikalische Regel
%IWn	CPU-Eingangsbereich mit Wortadresse	absoluter PA-Zugriff
%IDn	CPU-Eingangsbereich mit Doppelwortadresse	absoluter PA-Zugriff
%Qn.x oder %QXn.x	CPU-Ausgangsbereich mit Byte und Bitadresse	absoluter PA-Zugriff
%QBn	CPU-Ausgangsbereich mit Byteadresse	absoluter PA-Zugriff
%QWn	CPU-Ausgangsbereich mit Wortadresse	absoluter PA-Zugriff
%QDn	CPU-Ausgangsbereich mit Doppelwortadresse	absoluter PA-Zugriff

A.1.2.5 Bezeichner der Taskstartinfo

Die folgenden Bezeichner sind für die Taskstartinfo definiert:

Tabelle A-6 Bezeichner der Taskstartinfo

Bezeichner	Datentyp	Beschreibung
TSI#alarmNumber	DINT	Abfrage der Alarmnummer
TSI#commandId.high	UDINT	Abfrage der CommandId (höherwertiges Wort)
TSI#commandId.low	UDINT	Abfrage der CommandId (niederwertiges Wort)
TSI#currentTaskId	StructTaskId	Abfrage der TaskId der eigenen Task
TSI#cycleTime	TIME	Abfrage der projektierten Zykluszeit der eigenen Task
TSI#details	DWORD	Abfrage der Detailinformation
TSI#executionFaultType	UDINT	Abfrage der Art des Verarbeitungsfehlers
TSI#interruptId	UDINT	Abfrage des auslösenden Ereignisses
TSI#logBaseAdrIn	DINT	Abfrage der logischen Basisadresse
TSI#logBaseAdrOut	DINT	Abfrage der logischen Basisadresse
TSI#logDiagAddr	DINT	Abfrage der logischen Diagnose-Adresse
TSI#shutDownInitiator	UDINT	Abfrage der Veranlassung für Übergang in STOP
TSI#startTime	DT	Abfrage der Startzeit
TSI#taskId	StructTaskId	Abfrage der TaskId der auslösenden Task
TSI#toInst	ANYOBJECT	Abfrage der TO-Instanz

A.1.2.6 Operatoren

Nachfolgend sehen Sie alle ST-Operatoren mit Angabe der syntaktischen Regeln, in denen Sie benutzt werden.

Tabelle A-7 ST-Operatoren

Operator	Beschreibung	Regel
:=	Zuweisungsoperator (auch für Initialisierungswerte))	Wertzuweisung, Eingangszuweisung, Durchgangszuweisung, Variablendeklaration, Konstantendeklaration, Anwenderdefinierte Datentypen, Komponentendeklaration
+, –	Arithmetische Operatoren: unäre Operatoren, Vorzeichen	Ausdruck, Exponent
+, –, *, / MOD	Arithmetische Basisoperatoren	Ausdruck, arithmetischer Basisoperator
**	Arithmetische Operatoren: Potenzoperator	Ausdruck
NOT	Logische Operatoren: Negation	Ausdruck, Operand
AND, &, OR, XOR	Logische Basisoperatoren	Logischer Basisoperator
<, >, <=, >=, =, <>	Vergleichsoperator	Vergleichsoperator
=>	Zuweisungsoperator	Ausgangszuweisung

A.1.2.7 Reservierte Wörter

Nachfolgend finden Sie Schlüsselwörter, vordefinierte Bezeichner und Standardfunktionen des ST-Grundsystems alphabetisch aufgelistet. Dazu erhalten Sie eine Beschreibung sowie die syntaktische Regel aus *Regeln*, in der sie als Terminale verwendet werden. Eine Ausnahme stellen Standardfunktionen dar, die nur implizit in der syntaktischen Regel *Funktionsaufruf* als Standardfunktionsname enthalten sind.

Hinweis

Variablen dürfen nicht wie Schlüsselworte und vordefinierte Bezeichner heißen. Näheres zu Bezeichnern siehe *Bezeichner im ST*. Eine Übersicht der Bezeichner, die von Technologischen Objekten abhängig sind, und weitere reservierte Bezeichner finden Sie in *Reservierte Bezeichner*.

Tabelle A-8 ST-Schlüsselwörter und vordefinierte Bezeichner im ST-Grundsystem

Schlüsselwörter / Bezeichner	Beschreibung	Regel
ABS	Numerische Standardfunktion	Funktionsaufruf
ACOS	Numerische Standardfunktion	Funktionsaufruf
AND	Logischer Operator	Logischer Basisoperator
ANYOBJECT	Allgemeiner Datentyp für Technologieobjekte	TO-Datentyp

Schlüsselwörter / Bezeichner	Beschreibung	Regel
ANYOBJECT_TO_OBJECT	Standardfunktion für Typkonvertierung (Technologieobjekte)	Funktionsaufruf
ANYTYPE_TO_BIGBYTEARRAY	Standardfunktion (Marshalling)	Funktionsaufruf
ANYTYPE_TO_LITTLEBYTEARRAY	Standardfunktion (Marshalling)	Funktionsaufruf
ARRAY	Einleitung der Spezifikation eines Arrays, danach folgt zwischen **[** und **]** die Indexliste	ARRAY-Datentyp-Spezifikation
AS	Einleitung für Namensraum	–
ASIN	Numerische Standardfunktion	Funktionsaufruf
AT	Schlüsselwort für Adressangabe	Komponentendeklaration mit relativer Adresse, symbolischer PA-Zugriff
ATAN	Numerische Standardfunktion	Funktionsaufruf
BIGBYTEARRAY_TOANYTYPE	Standardfunktion (Marshalling)	Funktionsaufruf
BOOL	Elementarer Datentyp für binäre Daten	Bitdatentyp
BOOL_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BOOL_VALUE_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BY	Einleitung der Schrittweite	FOR-Anweisung (Seite 421)
BYTE	Elementarer Datentyp	Bitdatentyp
BYTE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_VALUE_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
BYTE_VALUE_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
CASE	Einleitung Kontrollanweisung zur Selektion	CASE-Anweisung
CONCAT	Standardfunktion für Stringbearbeitung	Funktionsaufruf
CONCAT_DATE_TOD	Standardfunktion für Typkonvertierung	Funktionsaufruf
CONSTANT	Einleitung für Definition von Konstanten	Konstantenblock
COS	Numerische Standardfunktion	Funktionsaufruf
CTD	Abwärtszähler	Funktionsbaustein-Aufruf

Schlüsselwörter / Bezeichner	Beschreibung	Regel
CTD_DINT	Abwärtszähler	Funktionsbaustein-Aufruf
CTD_UDINT	Abwärtszähler	Funktionsbaustein-Aufruf
CTU	Aufwärtszähler	Funktionsbaustein-Aufruf
CTU_DINT	Aufwärtszähler	Funktionsbaustein-Aufruf
CTU_UDINT	Aufwärtszähler	Funktionsbaustein-Aufruf
CTUD	Auf- Abwärtszähler	Funktionsbaustein-Aufruf
CTUD_DINT	Auf- Abwärtszähler	Funktionsbaustein-Aufruf
CTUD_UDINT	Auf- Abwärtszähler	Funktionsbaustein-Aufruf
DATE	Elementarer Datentyp für Datum	Zeitdatentyp
DATE_AND_TIME	Elementarer Datentyp für Datum und Uhrzeit	Zeitdatentyp
DATE_AND_TIME_TO_DATE	Standardfunktion für Typkonvertierung	Funktionsaufruf
DATE_AND_TIME_TO_TIME_OF_DAY	Standardfunktion für Typkonvertierung	Funktionsaufruf
DELETE	Standardfunktion für Stringbearbeitung	Funktionsaufruf
DINT	Elementarer Datentyp für Ganzzahl doppelter Genauigkeit (double integer) im Wertebereich -2**31 .. 2**31-1	Numerischer Datentyp
DINT_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_STRING	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
DINT_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
DO	Einleitung des Anweisungsabschnitts bei FOR-Anweisung, WHILE-Anweisung oder WAITFOR-CONDITION-Anweisung	FOR-Anweisung (Seite 421), WHILE-Anweisung (Seite 421), WAITFORCONDITION-Anweisung (Seite 422)
DT	Kurzschreibweise für DATE_AND_TIME	Zeitdatentyp
DT_TO_DATE	Standardfunktion für Typkonvertierung	Funktionsaufruf
DT_TO_TOD	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD	Elementarer Datentyp Doppelwort	Bitdatentyp
DWORD_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf

Schlüsselwörter / Bezeichner	Beschreibung	Regel
DWORD_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_VALUE_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
DWORD_VALUE_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
ELSE	Einleitung des Falls, wenn keine Bedingung erfüllt ist	IF-Anweisung, CASE-Anweisung
ELSIF	Einleitung der Alternativ-Bedingung	IF-Anweisung
END_CASE	Abschluss der CASE-Anweisung	CASE-Anweisung
END_EXPRESSION	Abschluss der EXPRESSION-Anweisung	Expression
END_FOR	Abschluss der FOR-Anweisung	FOR-Anweisung (Seite 421)
END_FUNCTION	Abschluss der Funktion	Funktion
END_FUNCTION_BLOCK	Abschluss des Funktionsbausteins	Funktionsbaustein
END_IF	Abschluss der IF-Anweisung	IF-Anweisung
END_IMPLEMENTATION	Abschluss des Implementation-Teils	Implementation-Teil
END_INTERFACE	Abschluss des Interface-Teils	Interface-Teil
END_LABEL	Abschluss der LABEL-Anweisung	Sprungmarkendeklaration (Seite 400)
END_PROGRAM	Abschluss des Programms	Programm
END_REPEAT	Abschluss der REPEAT-Anweisung	REPEAT-Anweisung (Seite 421)
END_STRUCT	Abschluss der Spezifikation einer Struktur	STRUCT-Datentyp-Spezifikation
END_TYPE	Abschluss der UDT-Definition	Anwenderdefinierter Datentyp
END_VAR	Abschluss eines Deklarationsblocks	Variablenblock, Parameterblock, Konstantenblock
END_WAITFORCONDITION	Abschluss der Kontrollanweisung für eine auf ein programmierbares Ereignis wartende Task	WAITFORCONDITION-Anweisung (Seite 422)
END_WHILE	Abschluss der WHILE-Anweisung	WHILE-Anweisung (Seite 421)
ENUM_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
EXIT	Direkter Ausstieg aus der Schleifenbearbeitung	EXIT-Anweisung (Seite 422)
EXP	Numerische Standardfunktion	Funktionsaufruf
EXPD	Numerische Standardfunktion	Funktionsaufruf
EXPRESSION	Programmierbares Ereignis für wartende Task	Expression
EXPT	Numerische Standardfunktion	Funktionsaufruf
F_TRIG	Erkennung Fallende Flanke	Funktionsbaustein-Aufruf
FALSE	Vordefinierte boolesche Konstante: Logische Bedingung nicht erfüllt, Wert gleich 0	–
FIND	Standardfunktion für Stringbearbeitung	Funktionsaufruf
FOR	Einleitung der Kontrollanweisung zur Schleifenbearbeitung	FOR-Anweisung (Seite 421)
FUNCTION	Einleitung der Funktion	Funktion
FUNCTION_BLOCK	Einleitung des Funktionsbausteins	Funktionsbaustein

Schlüsselwörter / Bezeichner	Beschreibung	Regel
GOTO	Sprung	GOTO-Anweisung (Seite 423)
IF	Einleitung Kontrollanweisung für Selektion	IF-Anweisung
IMPLEMENTATION	Einleitung für IMPLEMENTATION-Teil	IMPLEMENTATION-Teil
INSERT	Standardfunktion für Stringbearbeitung	Funktionsaufruf
INT	Elementarer Datentyp für Ganzzahl einfacher Genauigkeit (integer) im Wertebereich -2**15 .. 2**15-1	Numerischer Datentyp
INT_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_TIME	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
INT_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
INTERFACE	Einleitung des Interface-Teils	Interface-Teil
LABEL	Sprungmarkendefinition	Sprungmarkendeklaration (Seite 400)
LEFT	Standardfunktion für Stringbearbeitung	Funktionsaufruf
LEN	Standardfunktion für Stringbearbeitung	Funktionsaufruf
LIMIT	Standardfunktion für Auswahl	Funktionsaufruf
LITTLEBYTEARRAY_TOANYTYPE	Standardfunktion (Marshalling)	Funktionsaufruf
LN	Numerische Standardfunktion	Funktionsaufruf
LOG	Numerische Standardfunktion	Funktionsaufruf
LREAL	Elementarer Datentyp für Gleitpunktzahl doppelter Genauigkeit (long real) mit einer Bitbreite von 64	Numerischer Datentyp
LREAL_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_STRING	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_VALUE_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_VALUE_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
LREAL_VALUE_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf

Schlüsselwörter / Bezeichner	Beschreibung	Regel
MAX	Standardfunktion für Auswahl	Funktionsaufruf
MID	Standardfunktion für Stringbearbeitung	Funktionsaufruf
MIN	Standardfunktion für Auswahl	Funktionsaufruf
MOD	Arithmetischer Operator für Divisionsrest	Arithmetischer Basisoperator
MUX	Standardfunktion für Auswahl	Funktionsaufruf
NOT	Logischer Operator, gehört zu den Unären Operatoren	Ausdruck, Operand
OF	Schlüsselwort	ARRAY-Datentyp-Spezifikation, CASE-Anweisung
OR	Logischer Operator	Logischer Basisoperator
OVERLAP	Einleitung für Struktur mit überlappenden Adressbereichen	STRUCT-Datentyp-Spezifikation
PROGRAM	Einleitung des Programms	Programm
R_TRIG	Erkennung Steigende Flanke	Funktionsbaustein-Aufruf
REAL	Elementarer Datentyp für Gleitpunktzahl einfacher Genauigkeit (real) mit einer Bitbreite von 32	Numerischer Datentyp
REAL_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_STRING	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_TIME	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_VALUE_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_VALUE_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
REAL_VALUE_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
REPEAT	Einleitung der Kontrollanweisung zur Schleifenbearbeitung	REPEAT-Anweisung (Seite 421)
REPLACE	Standardfunktion für Stringbearbeitung	Funktionsaufruf
RETAIN	Deklaration gepufferter Variablen	Remanenter Variablenblock
RETURN	Steueranweisung zur Rückkehr von Unterprogramm	RETURN-Anweisung (Seite 422)
RIGHT	Standardfunktion für Stringbearbeitung	Funktionsaufruf
ROL	Bitstring-Standardfunktionen	Funktionsaufruf
ROR	Bitstring-Standardfunktionen	Funktionsaufruf
RS	Bistabiler Funktionsbaustein (vorrangig rücksetzen)	Funktionsbaustein-Aufruf
RTC	Echtzeituhr	Funktionsbaustein-Aufruf
SEL	Standardfunktion für Auswahl	Funktionsaufruf
SHL	Bitstring-Standardfunktionen	Funktionsaufruf

Schlüsselwörter / Bezeichner	Beschreibung	Regel
SHR	Bitstring-Standardfunktionen	Funktionsaufruf
SIN	Numerische Standardfunktion	Funktionsaufruf
SINT	Elementarer Datentyp für kurze Ganzzahl (short integer) im Wertebereich -128 .. 127	Numerischer Datentyp
SINT_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
SINT_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
SQRT	Numerische Standardfunktion	Funktionsaufruf
SR	Bistabiler Funktionsbaustein (vorrangig setzen)	Funktionsbaustein-Aufruf
STRING	Elementarer Datentyp für Zeichenketten	String-Datentyp
STRING_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
STRING_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
STRING_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
STRING_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
STRUCT	Einleitung der Spezifikation einer Struktur, danach folgt Liste der Komponenten	STRUCT-Datentyp-Spezifikation
StructAlarmId	Datentyp für AlarmId	–
StructAlarmId_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
StructTaskId	Datentyp für TaskId	–
TAN	Numerische Standardfunktion	Funktionsaufruf
THEN	Einleitung der Folgeaktionen, wenn Bedingung erfüllt	IF-Anweisung
TIME	Elementarer Datentyp für Zeitangaben	Zeitdatentyp
TIME_OF_DAY	Elementarer Datentyp für Tageszeit	Zeitdatentyp
TIME_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
TIME_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
TO	Einleitung des Endwerts	FOR-Anweisung (Seite 421)
TOD	Kurzschreibweise für TIME_OF_DAY	Zeitdatentyp
TOF	Ausschaltverzögerung	Funktionsbaustein-Aufruf
TON	Einschaltverzögerung	Funktionsbaustein-Aufruf
TP	Puls	Funktionsbaustein-Aufruf
TRUE	Vordefinierte Boolesche Konstante: Logische Bedingung erfüllt, Wert ungleich 0	–
TRUNC	Numerische Standardfunktion	Funktionsaufruf
TYPE	Einleitung der UDT-Definition	Anwenderdefinierter Datentyp

Schlüsselwörter / Bezeichner	Beschreibung	Regel
UDINT	Elementarer Datentyp für Ganzzahl doppelter Genauigkeit (double integer) ohne Vorzeichen (unsigned) von 0 .. 2**32-1	Numerischer Datentyp
UDINT_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_STRING	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
UDINT_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT	Elementarer Datentyp für Ganzzahl einfacher Genauigkeit (Integer) ohne Vorzeichen (unsigned) von 0 .. 2**16-1	Numerischer Datentyp
UINT_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
UINT_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
UNIT	Einleitung für UNIT-Teil	UNIT-Teil
UNTIL	Einleitung der Abbruchbedingung für REPEAT-Anweisung	REPEAT-Anweisung (Seite 421)
USELIB	Einleitung des Bibliotheksnamens	–
USEPACKAGE	Einleitung des Paketnamens	–
USES	Einleitung für Referenz auf andere Units	–
USINT	Elementarer Datentyp für kurze Ganzzahl (short integer) ohne Vorzeichen (unsigned) im Wertebereich 0 .. 255	Numerischer Datentyp
USINT_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf

Schlüsselwörter / Bezeichner	Beschreibung	Regel
USINT_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_TO_WORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
USINT_VALUE_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
VAR	Einleitung eines Deklarationsblocks lokaler Variablen oder Konstanten	Statischer Variablenblock, temporärer Variablenblock FC, Konstantenblock
VAR_GLOBAL	Einleitung eines Deklarationsblocks für Unit-Variablen (globale Variablen) oder Unit-Konstanten	Unit-Variablen, Unit-Konstanten
VAR_IN_OUT	Einleitung eines Deklarationsblocks	Parameterblock
VAR_INPUT	Einleitung eines Deklarationsblocks	Parameterblock
VAR_OUTPUT	Einleitung eines Deklarationsblocks	Parameterblock
VAR_TEMP	Einleitung eines Deklarationsblocks	Temporärer Variablenblock FB/Programm
VOID	Kein Rückgabewert bei der Funktion	Funktion
WAITFORCONDITION	Einleitung der Kontrollanweisung für eine auf ein programmierbares Ereignis wartende Task	WAITFORCONDITION-Anweisung (Seite 422)
WHILE	Einleitung der Kontrollanweisung zur Schleifenbearbeitung	WHILE-Anweisung (Seite 421)
WITH	Zusatz zur Kontrollanweisung WAITFORCONDITION	WAITFORCONDITION-Anweisung (Seite 422)
WORD	Elementarer Datentyp Wort	Bitdatentyp
WORD_TO_BOOL	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_BYTE	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_DINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_DWORD	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_INT	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_SINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_UDINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_UINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_TO_USINT	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_VALUE_TO_LREAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
WORD_VALUE_TO_REAL	Standardfunktion für Typkonvertierung	Funktionsaufruf
XOR	Logischer Operator	Logischer Basisoperator

A.1.3 Regeln

Nachfolgende Syntaxregeln der Sprache ST sind eingeteilt in Regeln mit formatpflichtiger (lexikalische Regeln) und formatfreier Schreibweise (syntaktische Regeln). Die Unterschiede zwischen syntaktischen und lexikalischen Regeln haben Sie in *Hilfen für die Sprachbeschreibung* erfahren.

A.1.3.1 Bezeichnungen

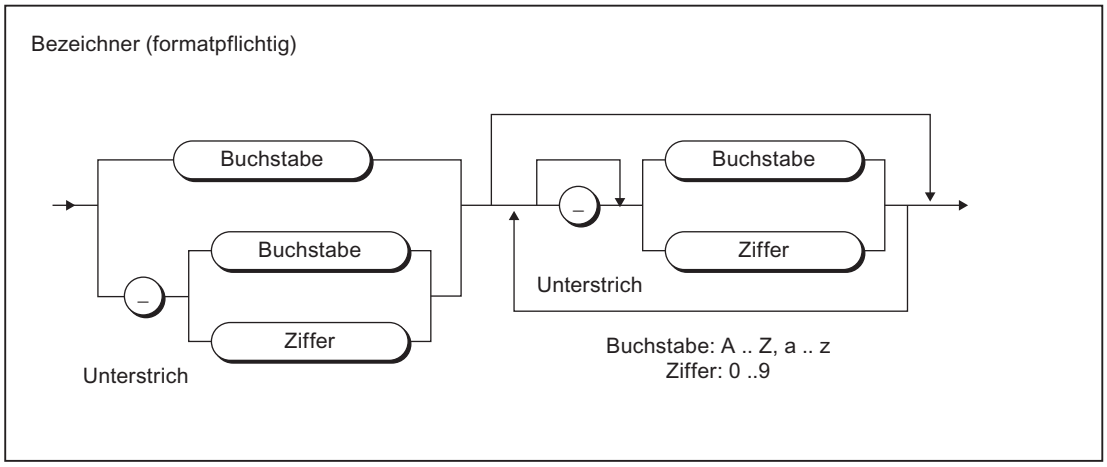


Bild A-3 Bezeichner

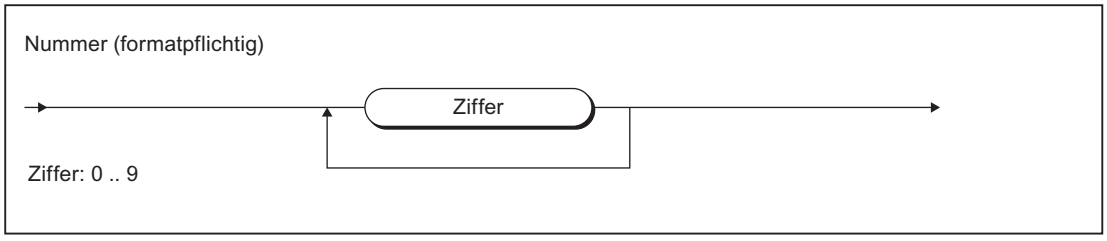


Bild A-4 Nummer

A.1.3.2 Schreibweise von Konstanten (Literale)

Literale

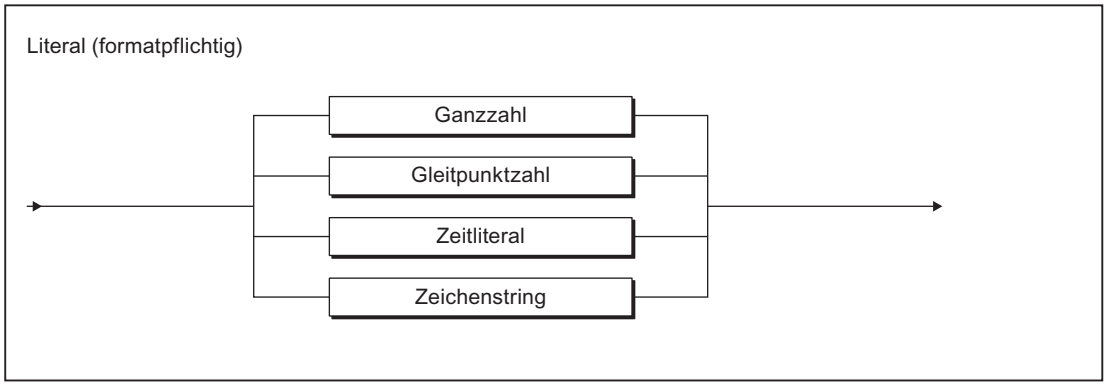


Bild A-5 Literal

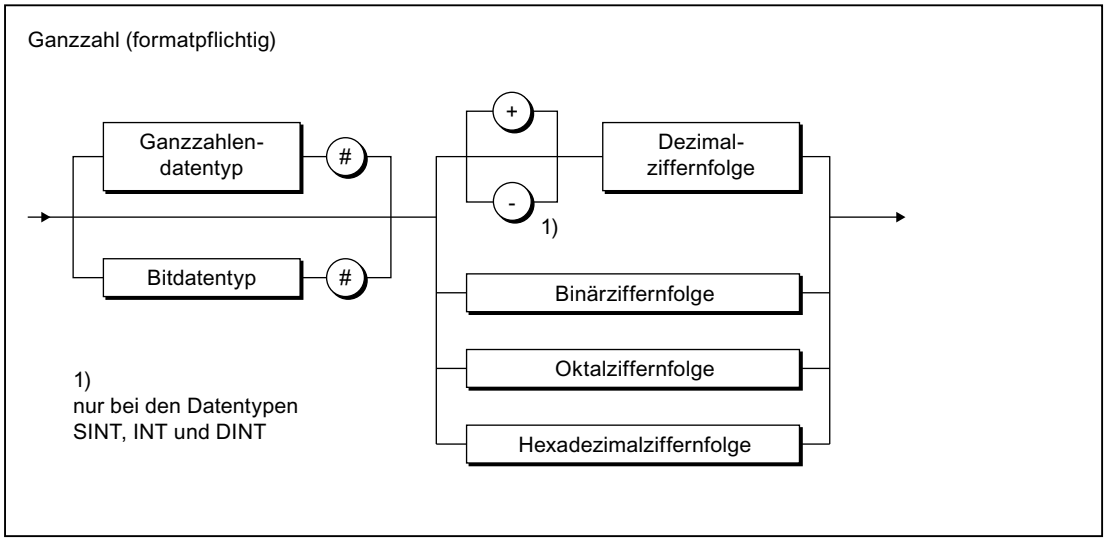


Bild A-6 Ganzzahl

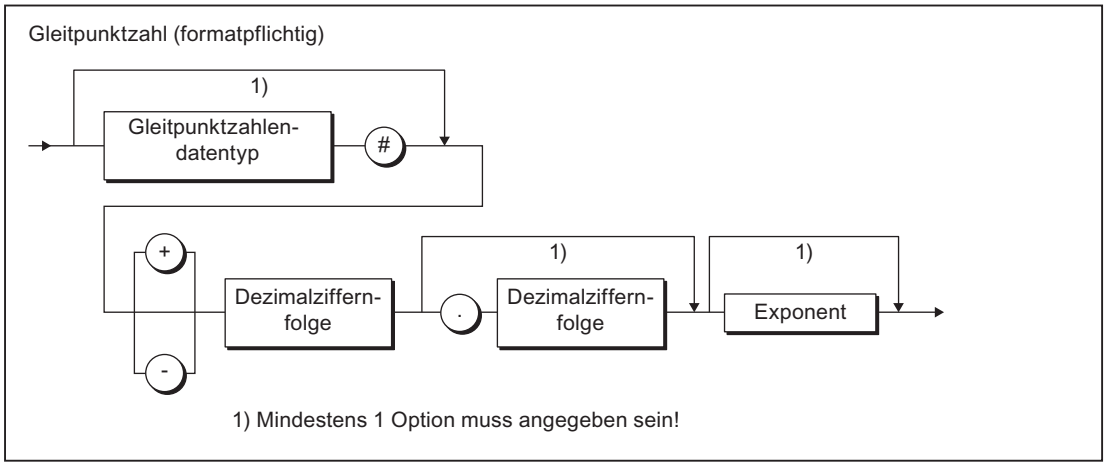


Bild A-7 Gleitpunktzahl

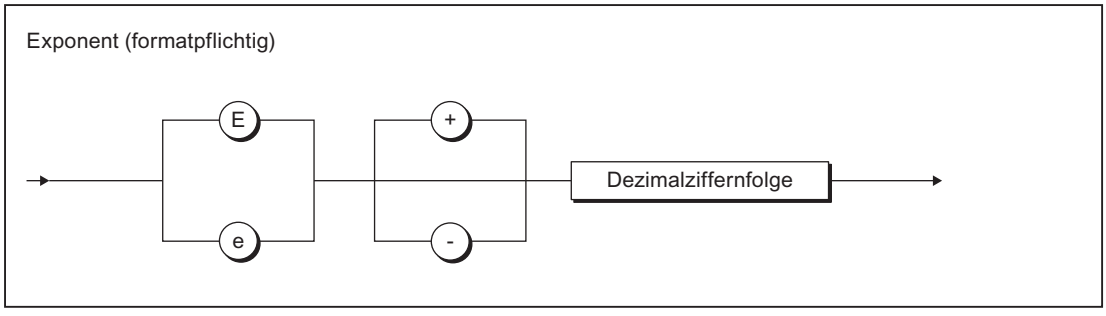


Bild A-8 Exponent

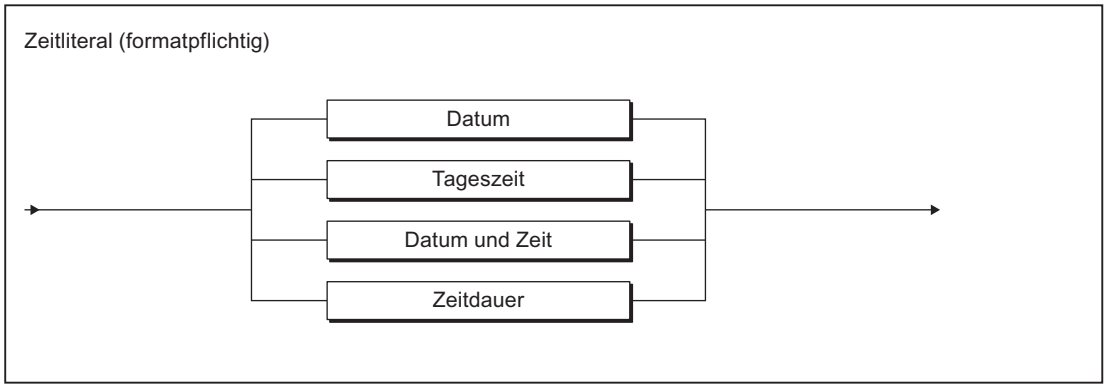


Bild A-9 Zeitliteral

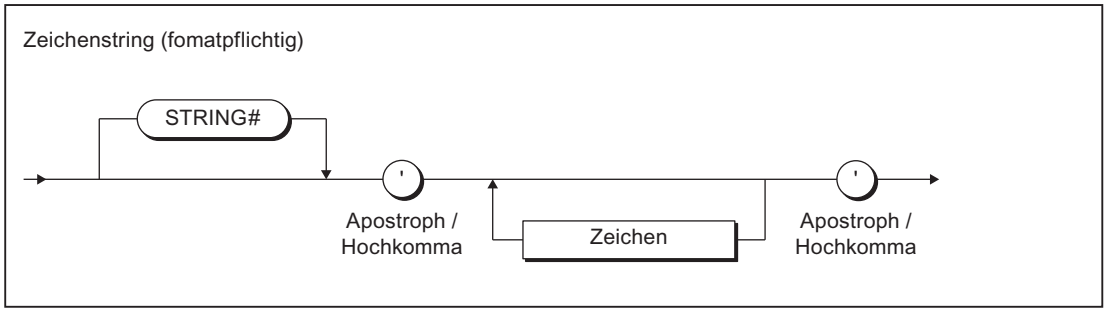


Bild A-10 Zeichenstring

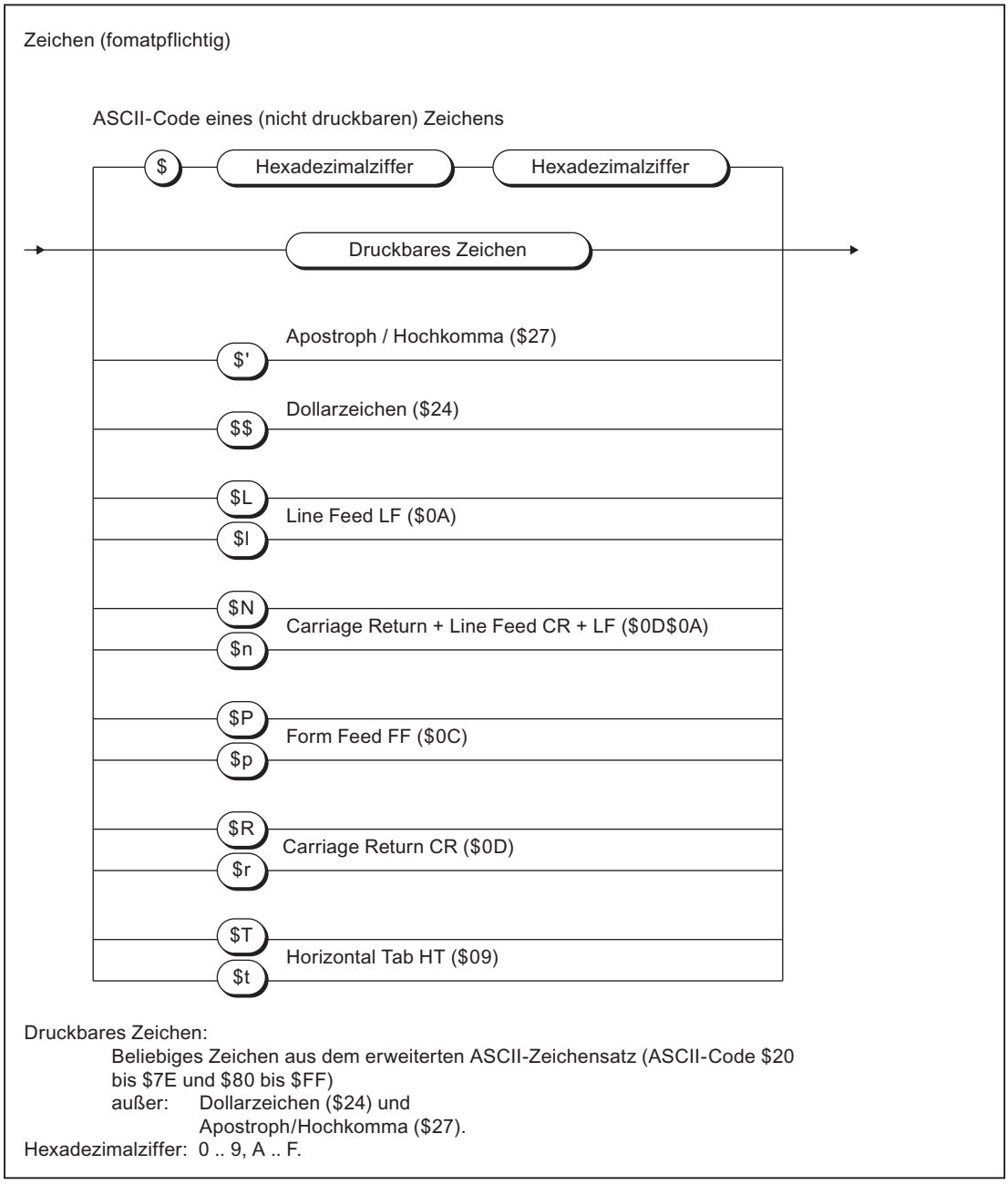


Bild A-11 Zeichen

Ziffernfolgen

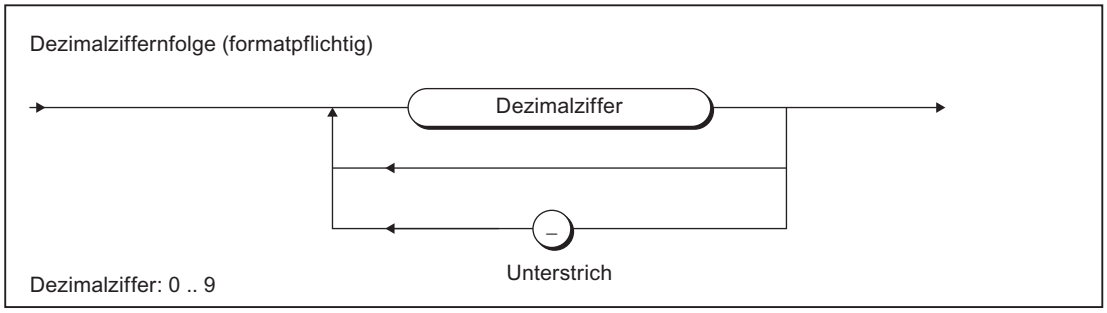


Bild A-12 Dezimalziffernfolge

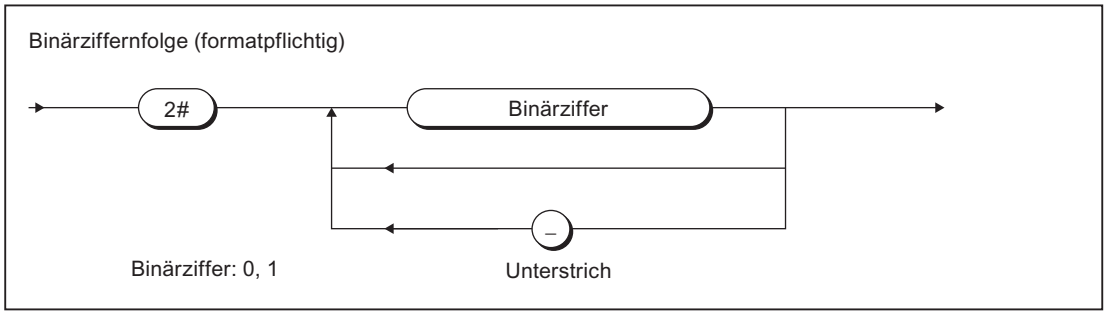


Bild A-13 Binärziffernfolge

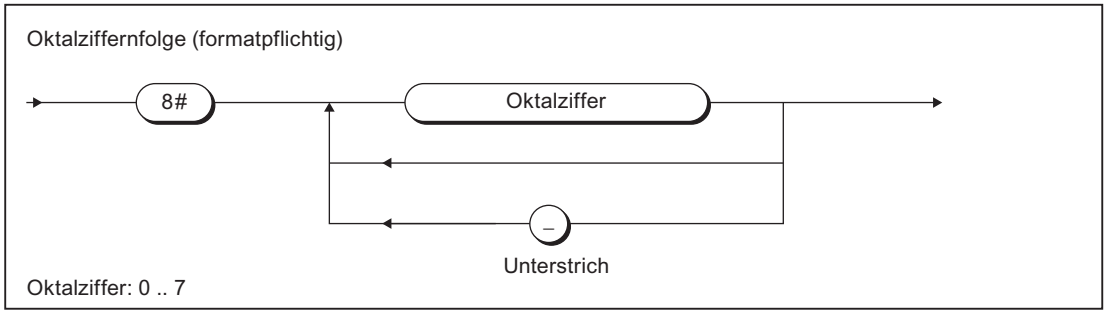


Bild A-14 Oktalziffernfolge

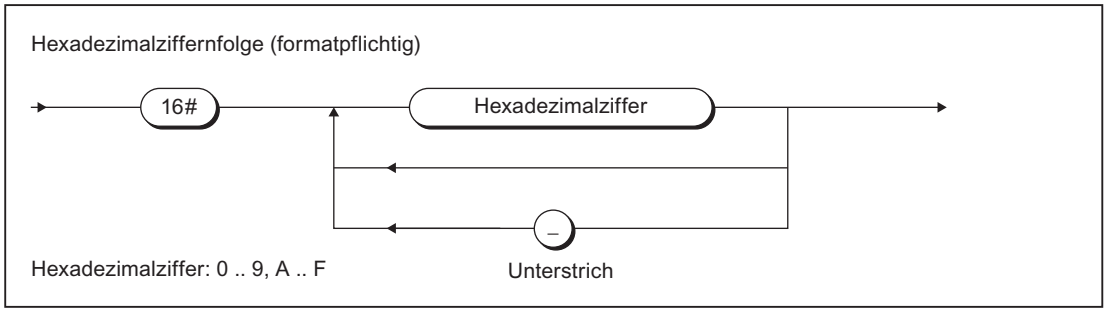


Bild A-15 Hexadezimalziffernfolge

Datum und Zeit

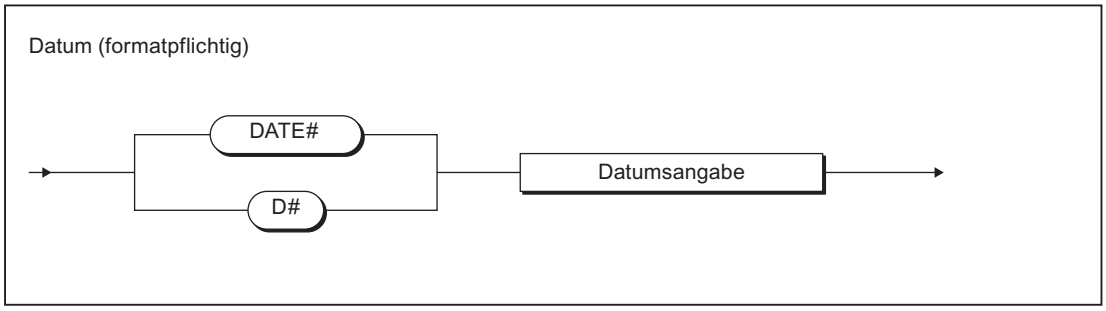


Bild A-16 Datum

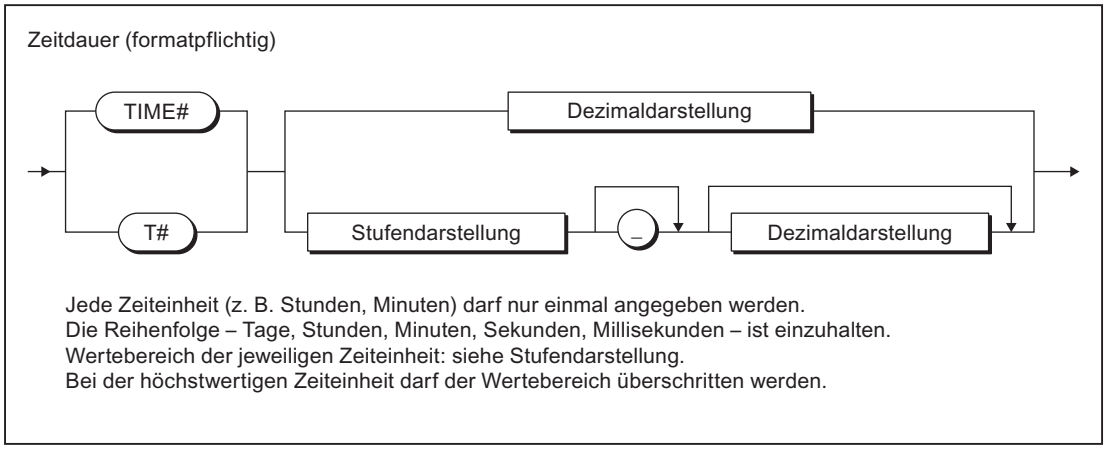


Bild A-17 Zeitdauer

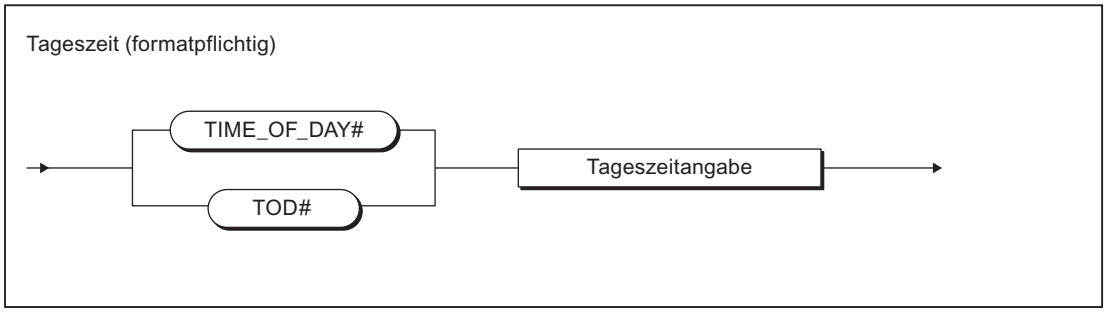


Bild A-18 Tageszeit

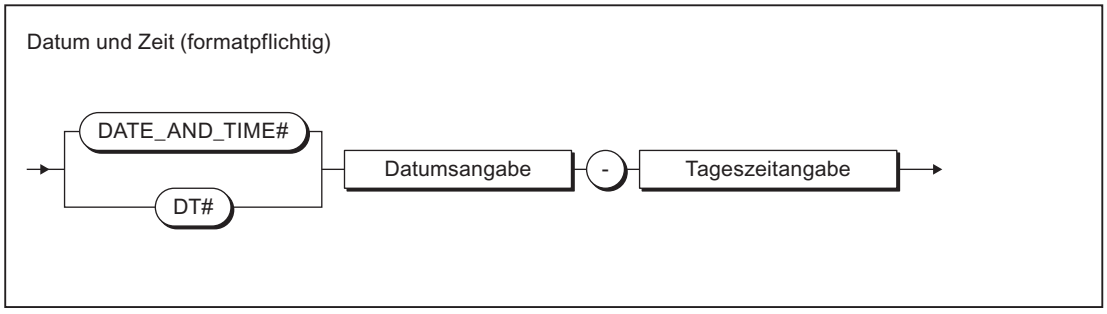


Bild A-19 Datum und Zeit

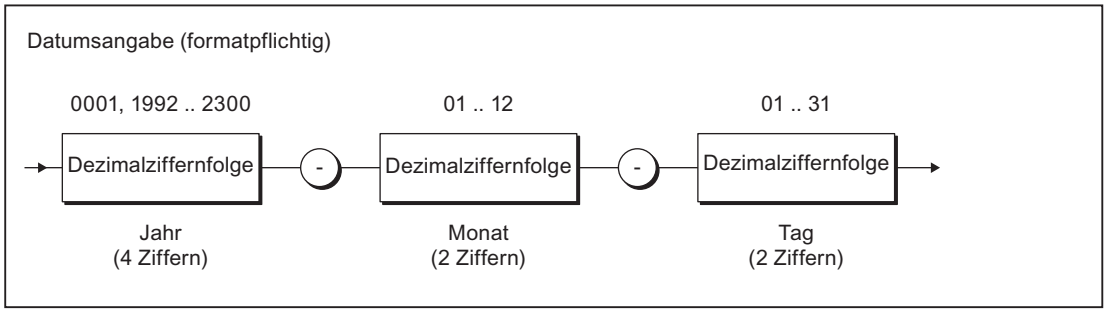


Bild A-20 Datumsangabe

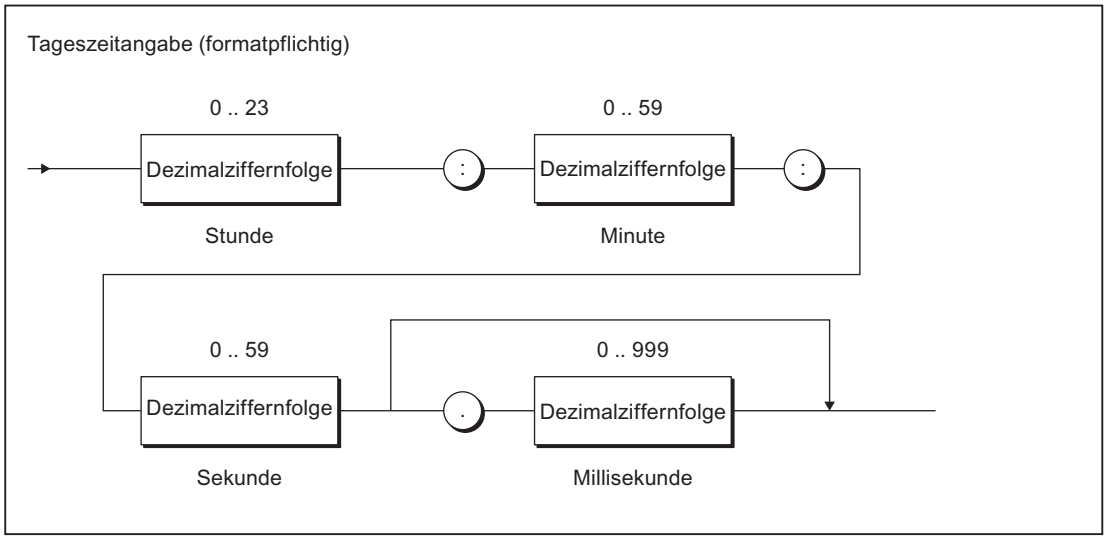


Bild A-21 Tageszeitangabe

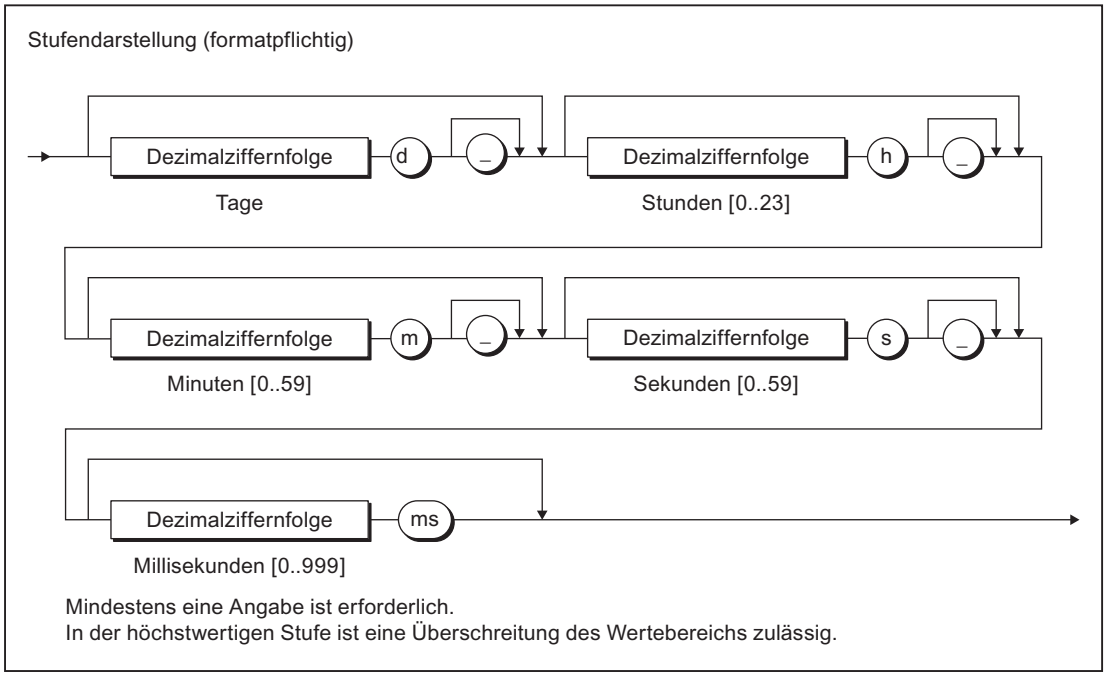


Bild A-22 Stufendarstellung

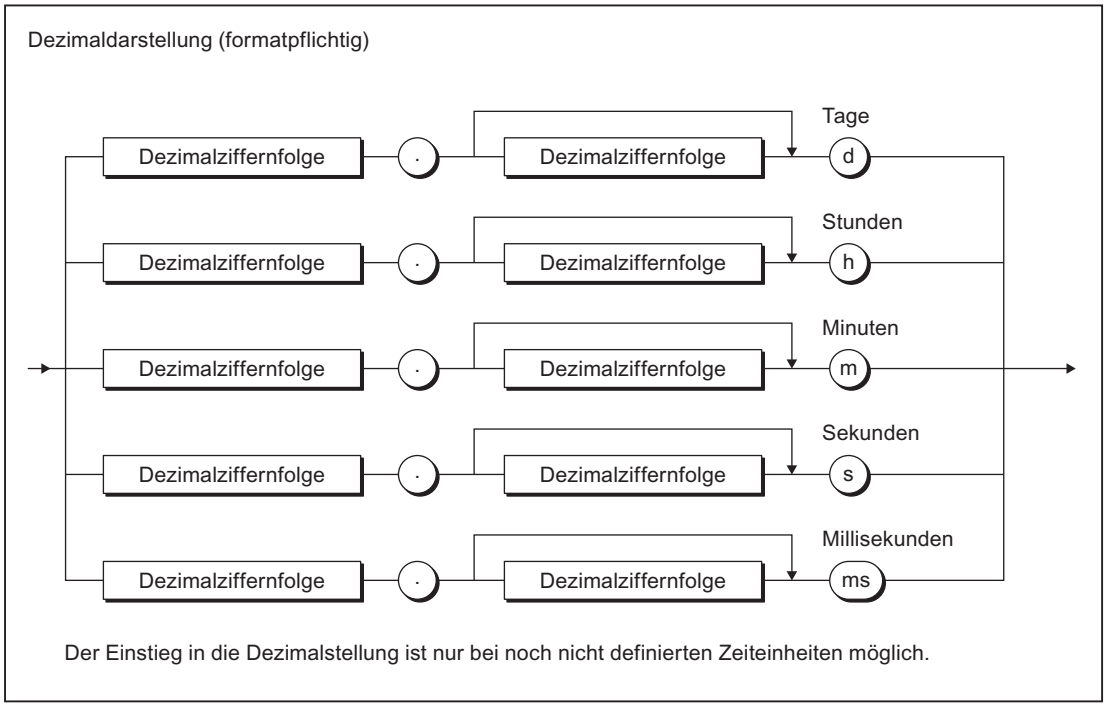


Bild A-23 Dezimaldarstellung

A.1.3.3 Kommentare

Bitte beachten Sie Folgendes beim Einbau von Kommentaren:

- Die Schachtelung von Zeilenkommentaren ist nicht erlaubt.
- Die Schachtelung von Blockkommentaren ist verboten, nicht jedoch die von Zeilenkommentaren in Blockkommentaren.
- Kommentare sind an beliebigen Stellen in den formatfreien (syntaktischen) Regeln erlaubt.
- In formatpflichtigen (lexikalischen) Regeln sind Kommentare verboten.

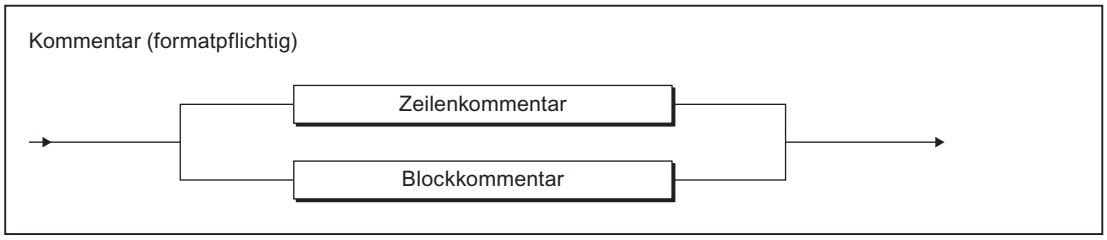


Bild A-24 Kommentar

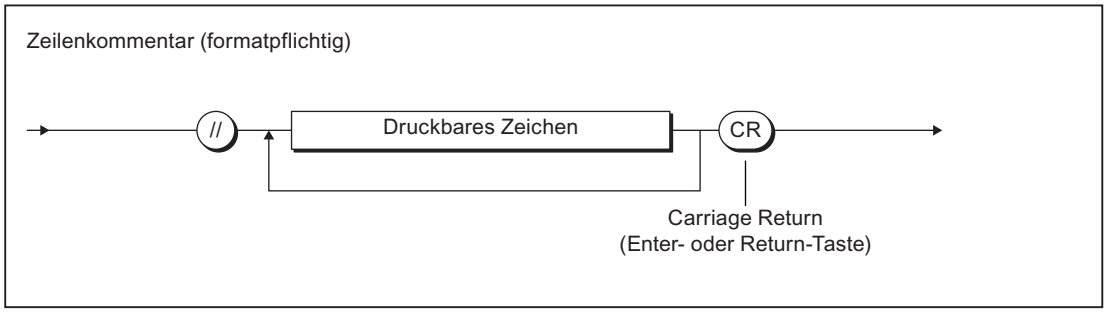


Bild A-25 Zeilenkommentar

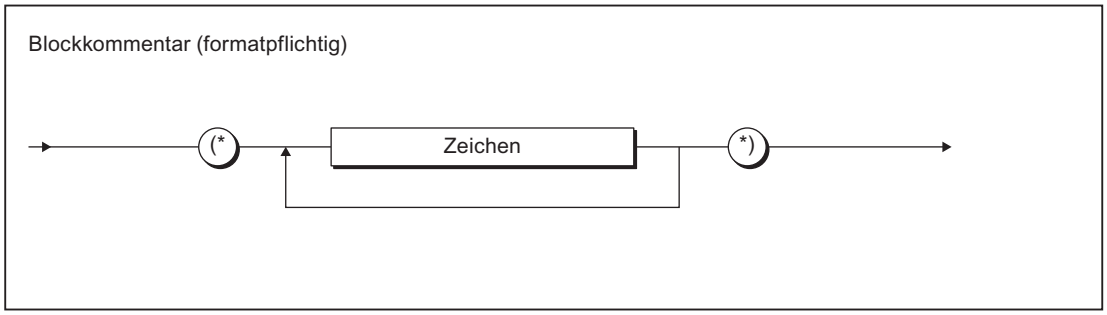
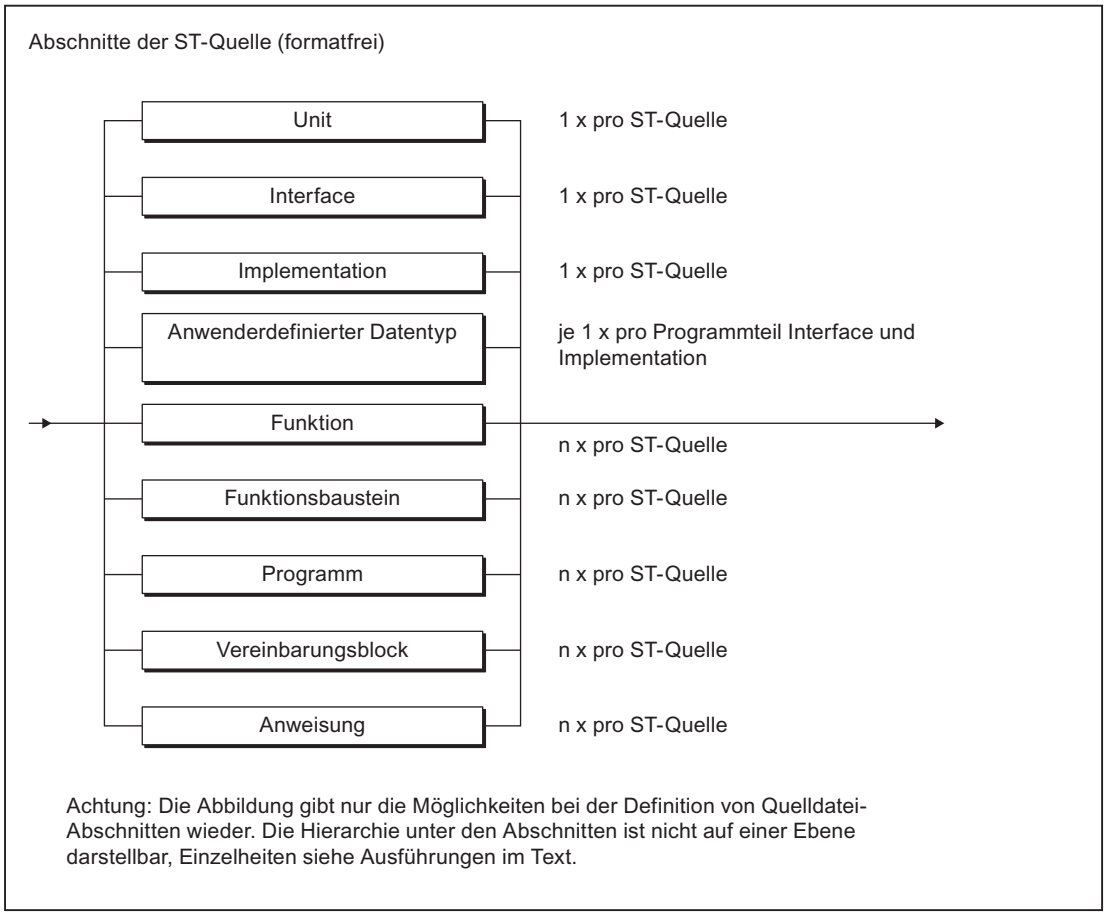


Bild A-26 Blockkommentar

A.1.3.4 Abschnitte der ST-Quelle



Abschnitte der ST-Quelle

A.1.3.5 Gliederungen von ST-Quellen

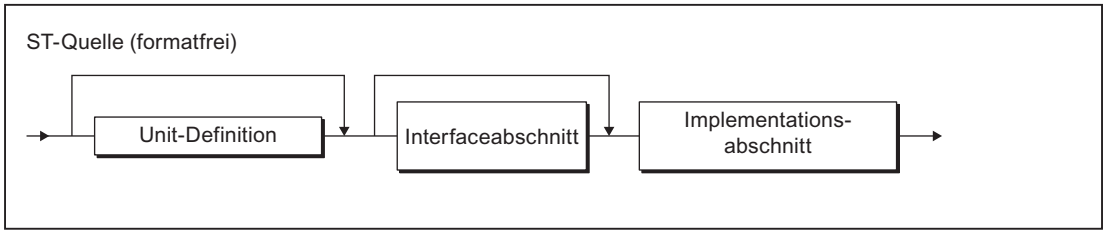


Bild A-27 ST-Quelle

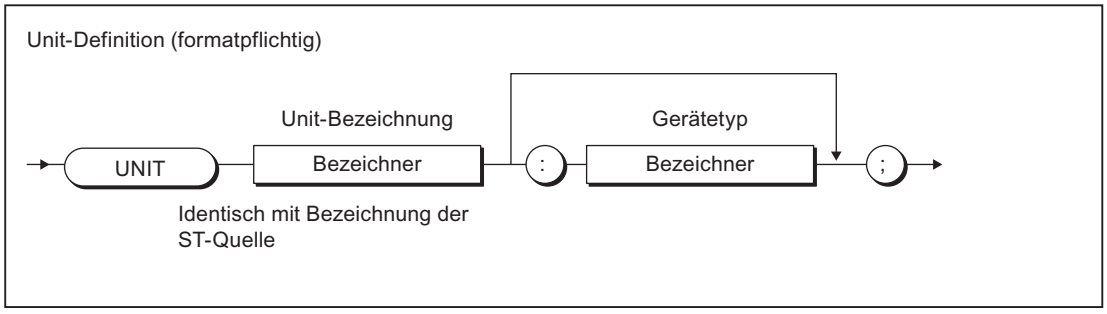


Bild A-28 Unit-Definition

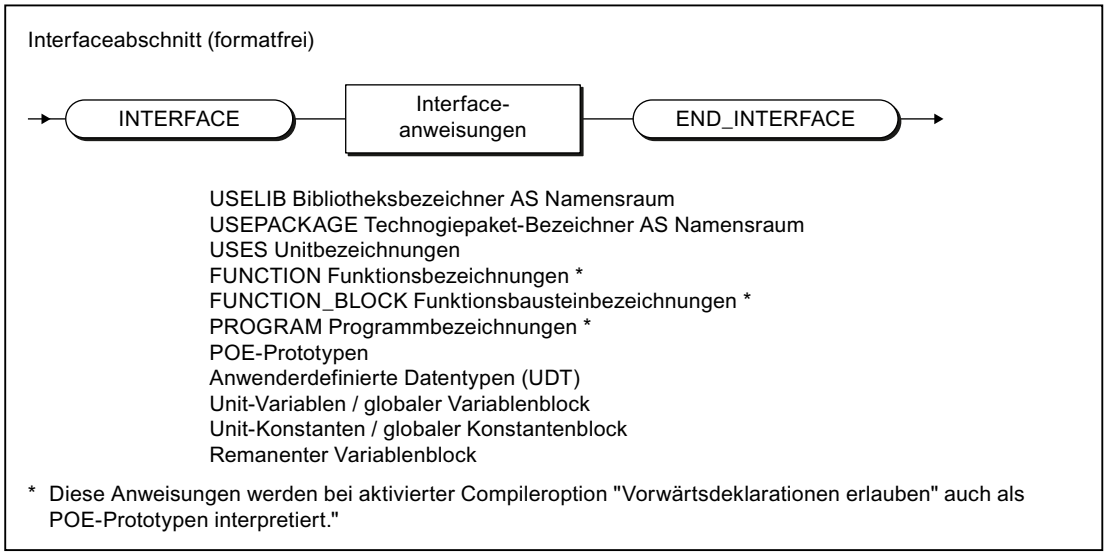


Bild A-29 Interfaceabschnitt

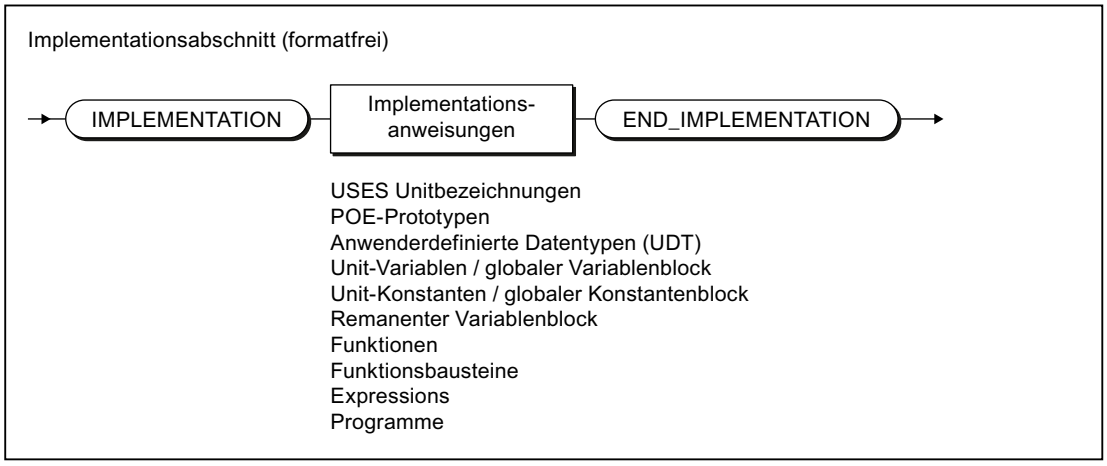


Bild A-30 Implementationsabschnitt

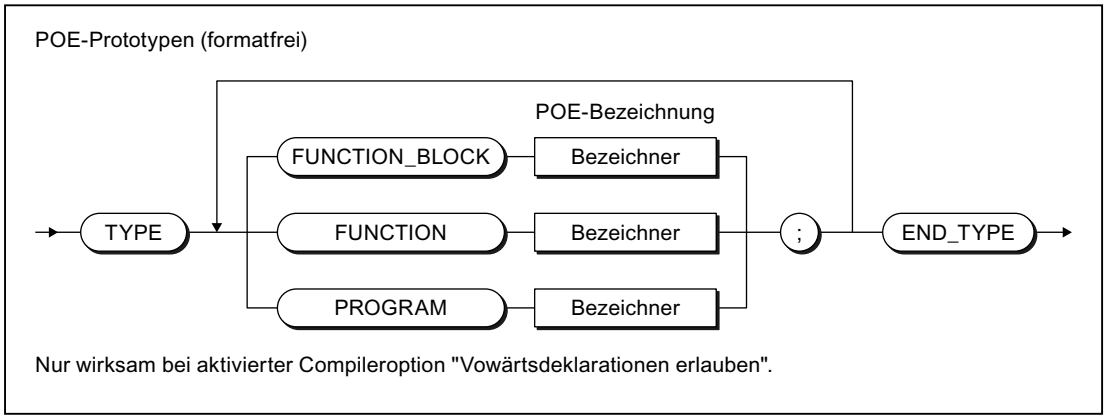


Bild A-31 POE-Prototypen

A.1.3.6 Programmorganisationseinheiten (POE)

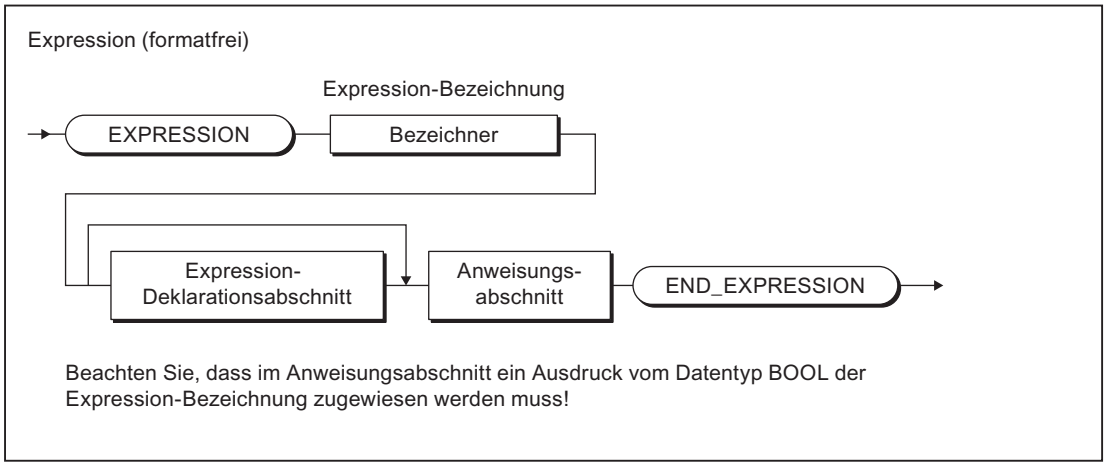


Bild A-32 Expression

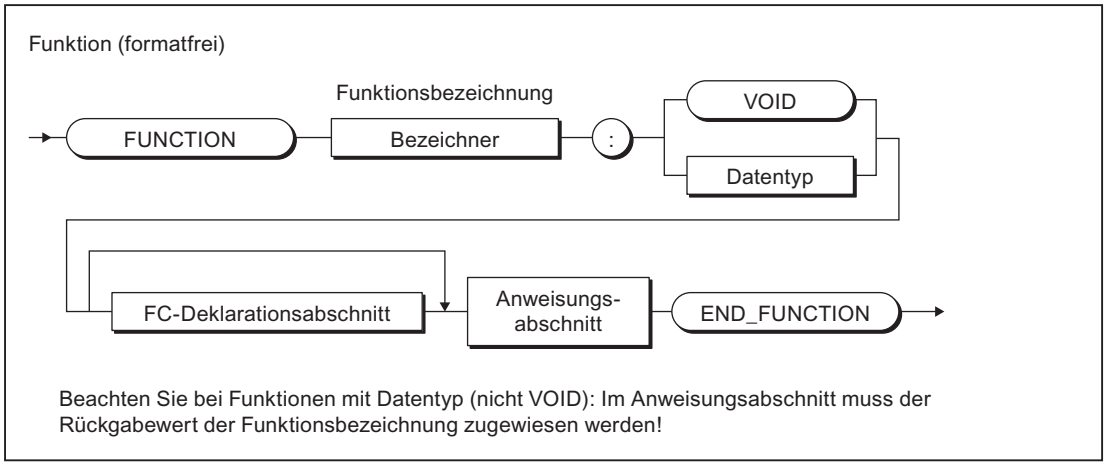


Bild A-33 Funktion FC

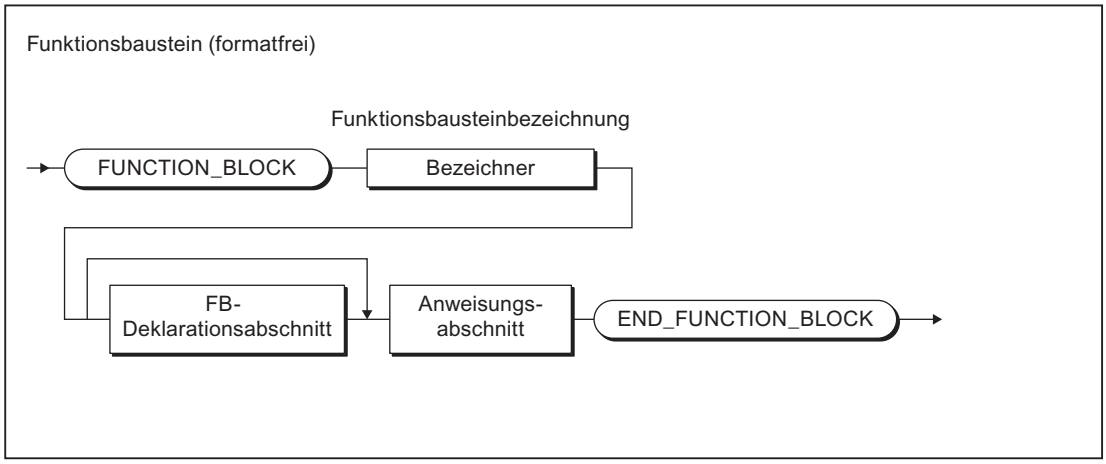


Bild A-34 Funktionsbaustein FB

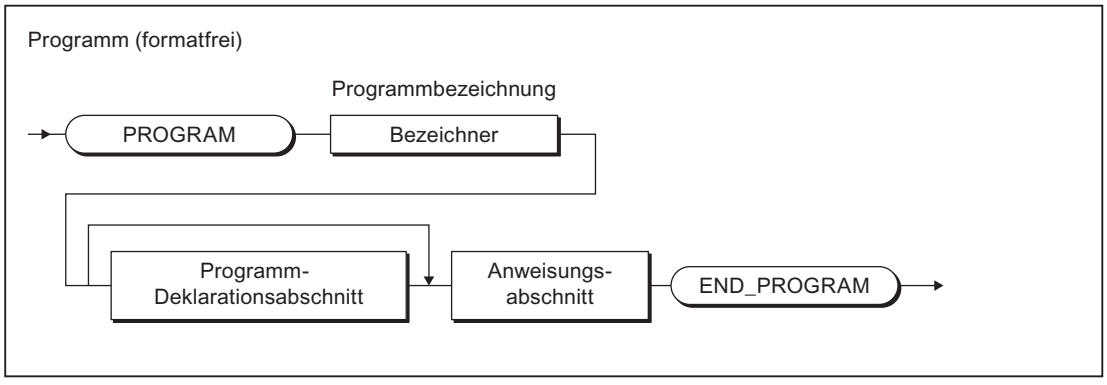


Bild A-35 Programm

A.1.3.7 Deklarationsabschnitte

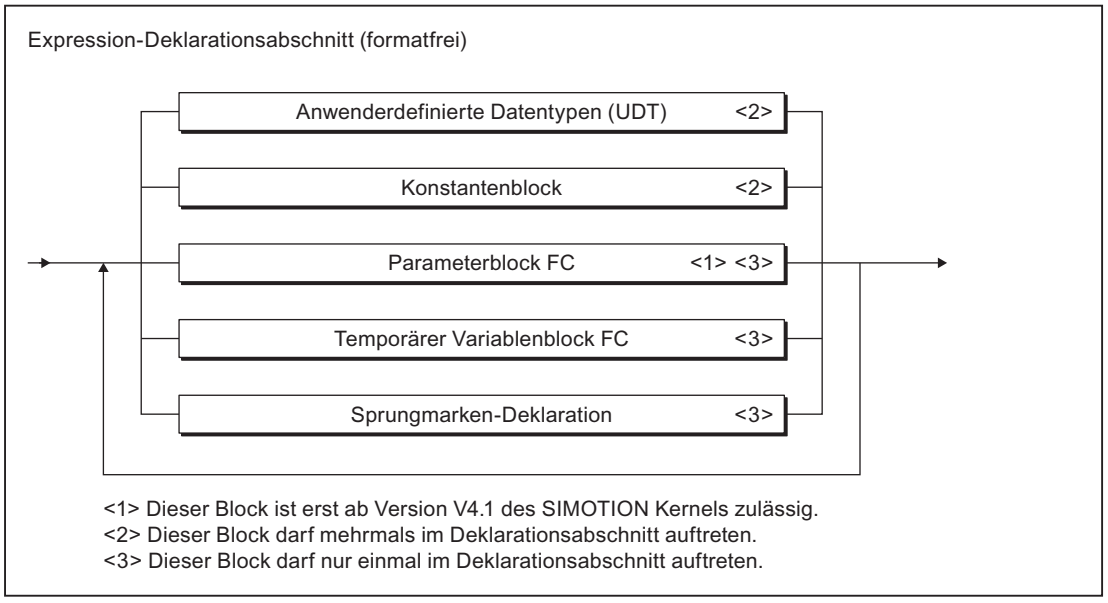


Bild A-36 Expression-Deklarationsabschnitt

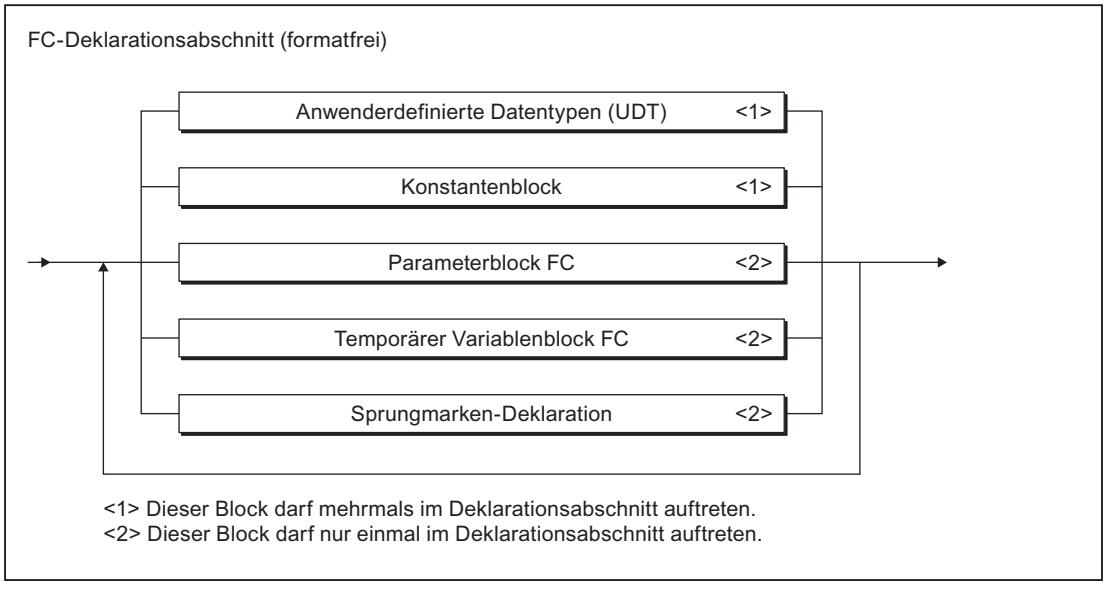


Bild A-37 FC-Deklarationsabschnitt

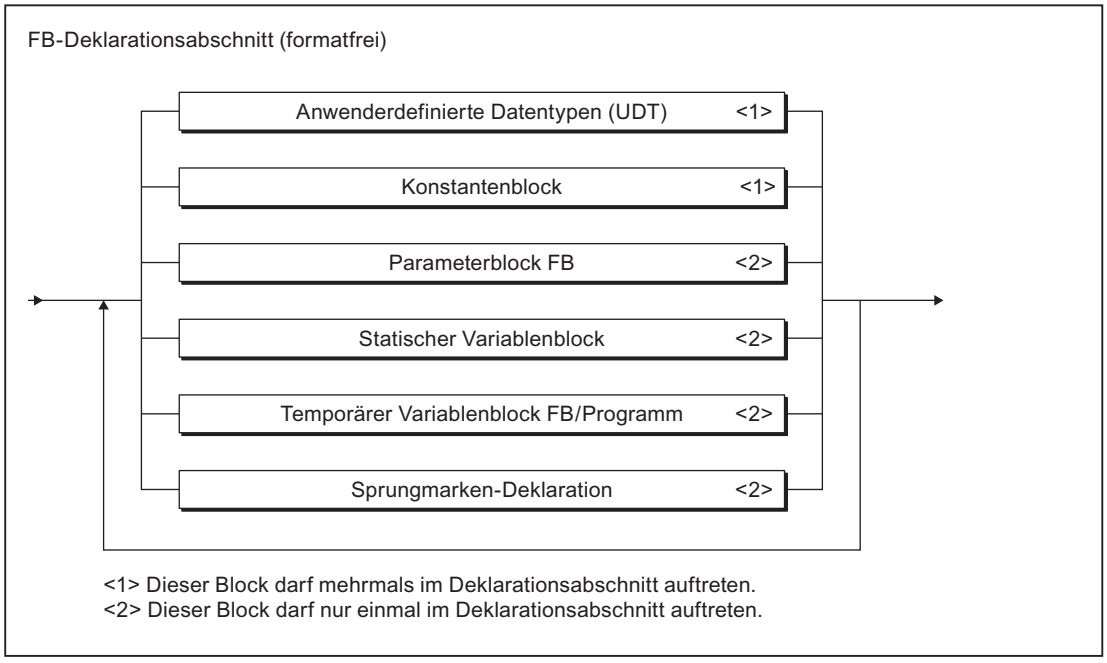


Bild A-38 FB-Deklarationsabschnitt

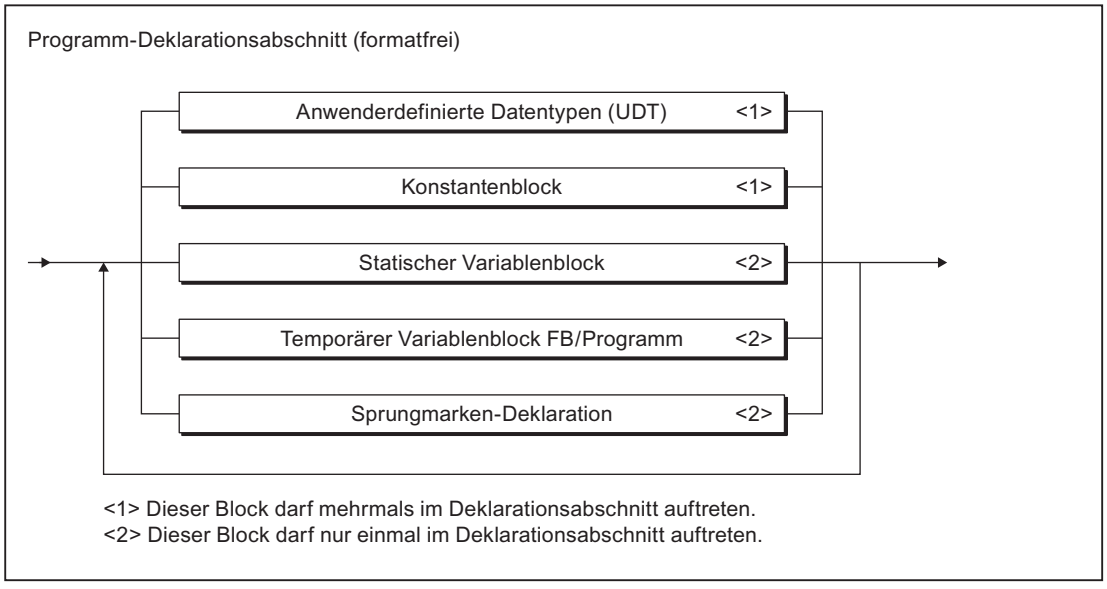


Bild A-39 Programm-Deklarationsabschnitt

A.1.3.8 Aufbau der Deklarationsblöcke

Konstantenblöcke

Bild A-40 Konstantenblock

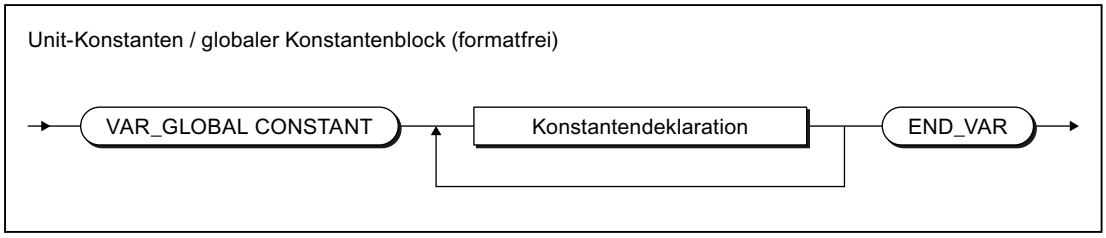


Bild A-41 Unit-Konstanten / Globaler Konstantenblock

Variablenblöcke

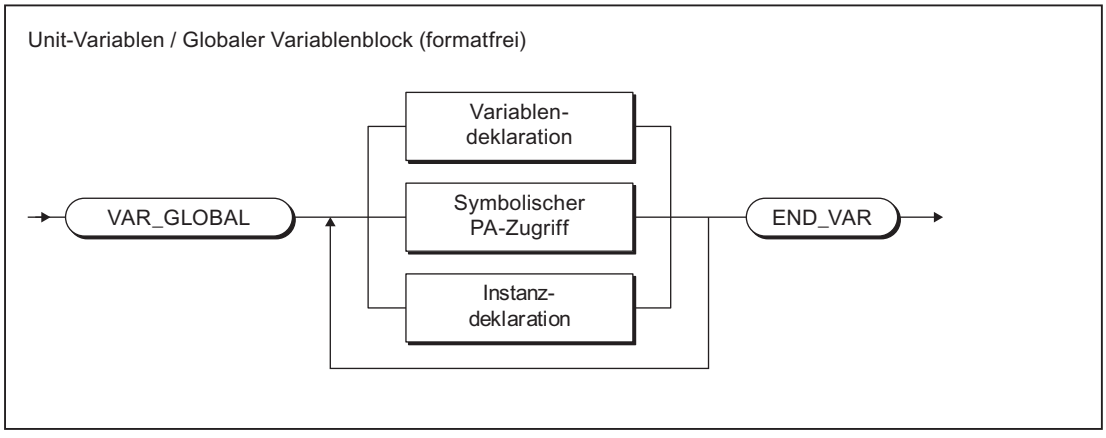


Bild A-42 Unit-Variablen / Globaler Variablenblock

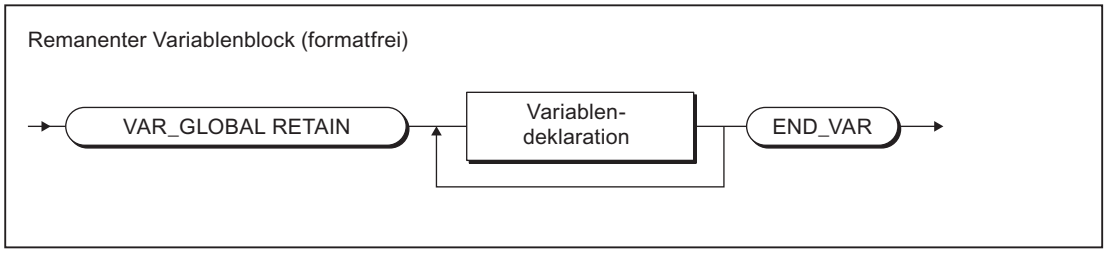


Bild A-43 Remanenter Variablenblock

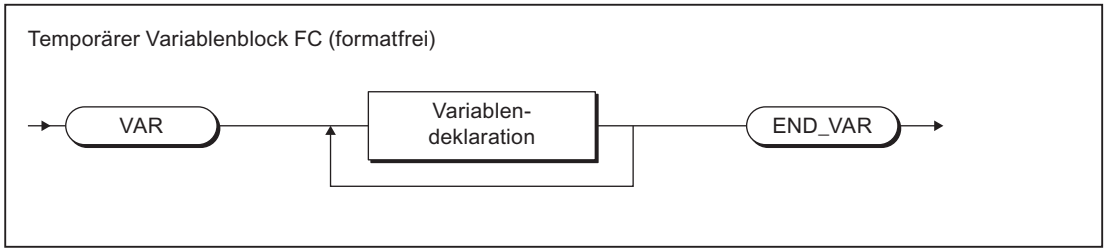


Bild A-44 Temporärer Variablenblock FC

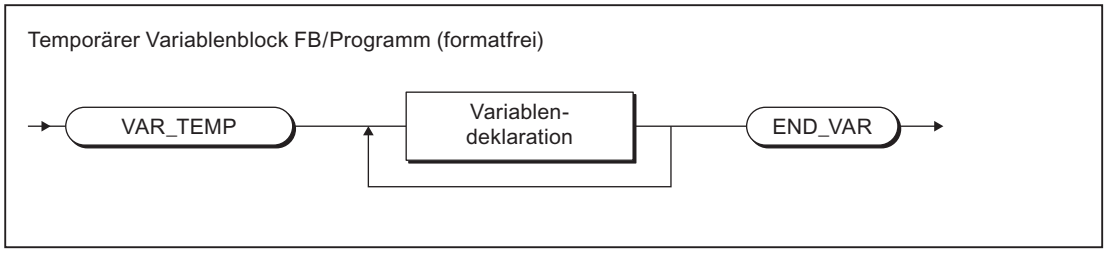


Bild A-45 Temporärer Variablenblock im FB/Programm

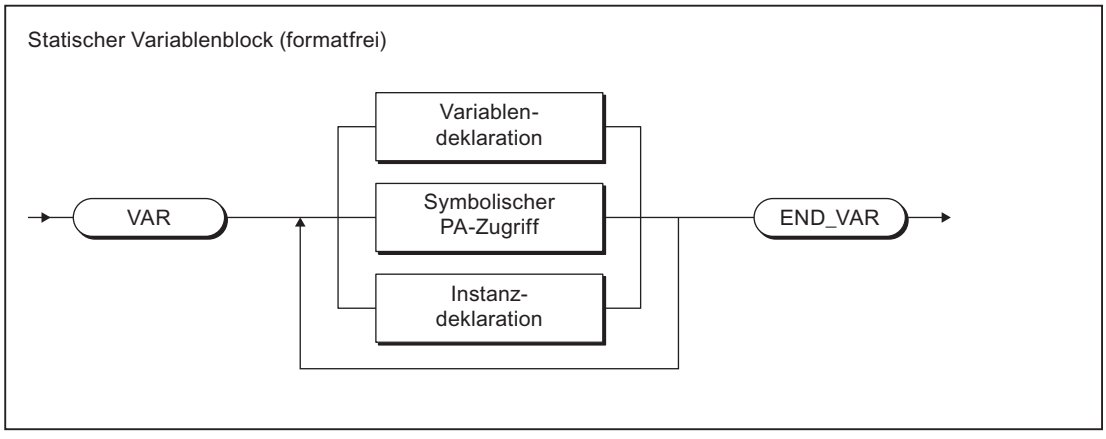


Bild A-46 Statischer Variablenblock

Parameterblöcke

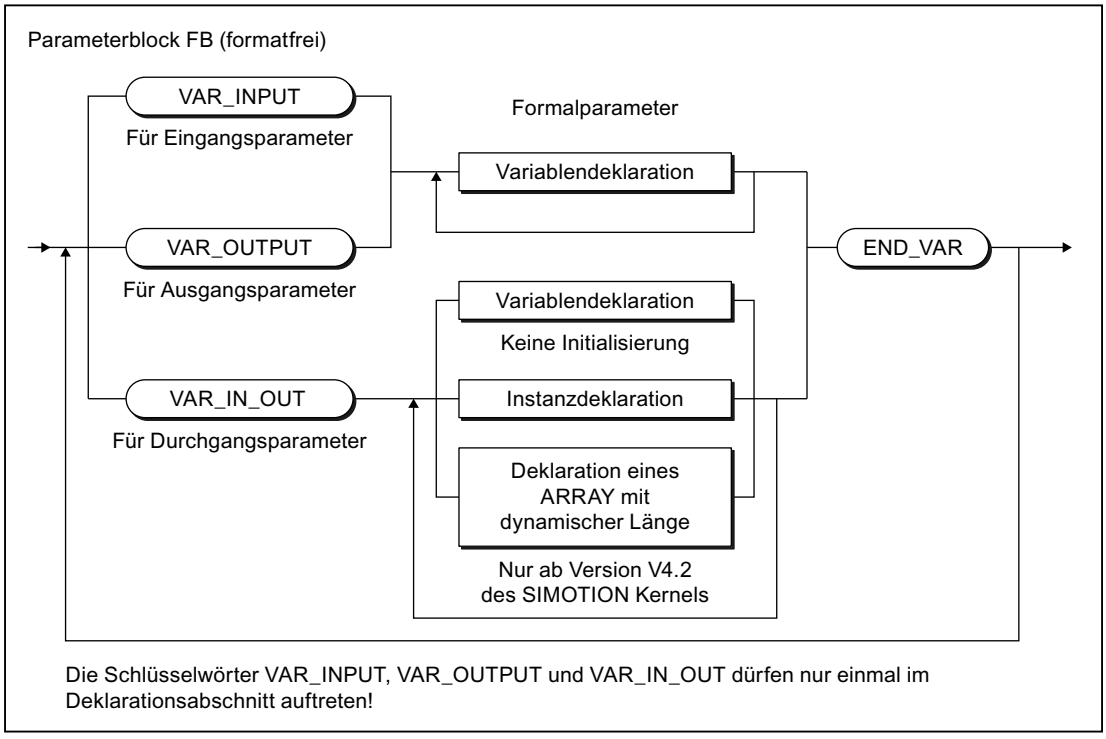


Bild A-47 Parameterblock FB

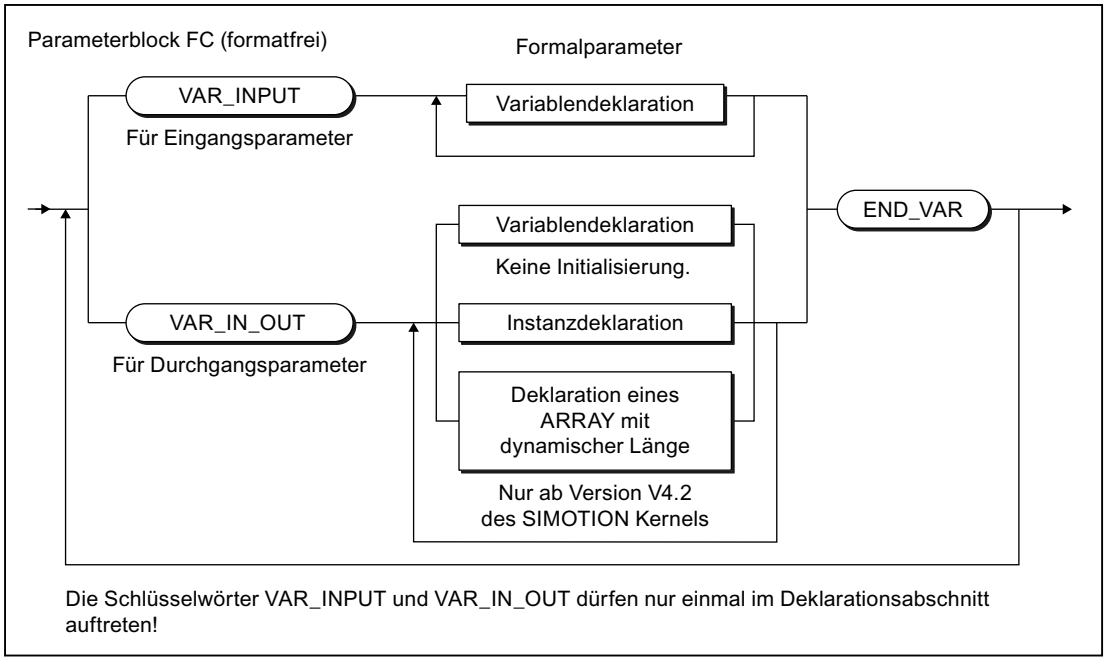


Bild A-48 Parameterblock FC

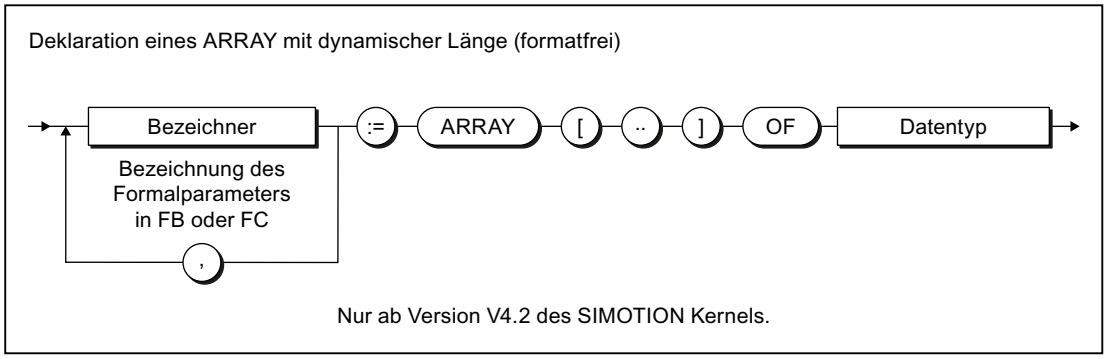


Bild A-49 Deklaration eines ARRAY mit dynamischer Länge

Sprungmarken

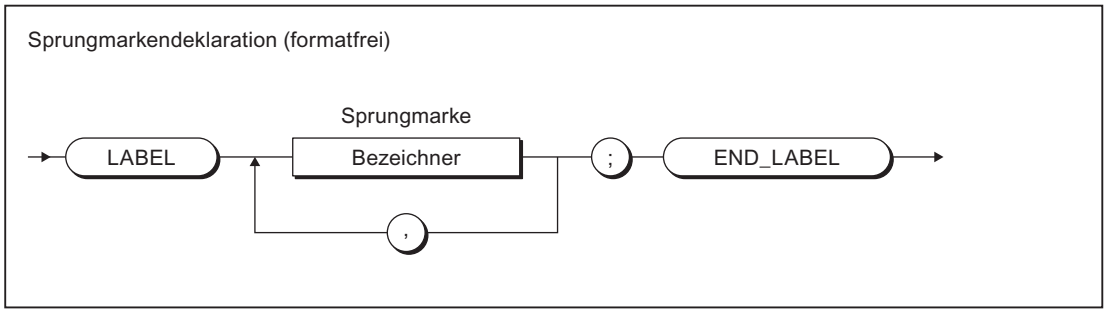


Bild A-50 Sprungmarkendeklaration

Deklarationen

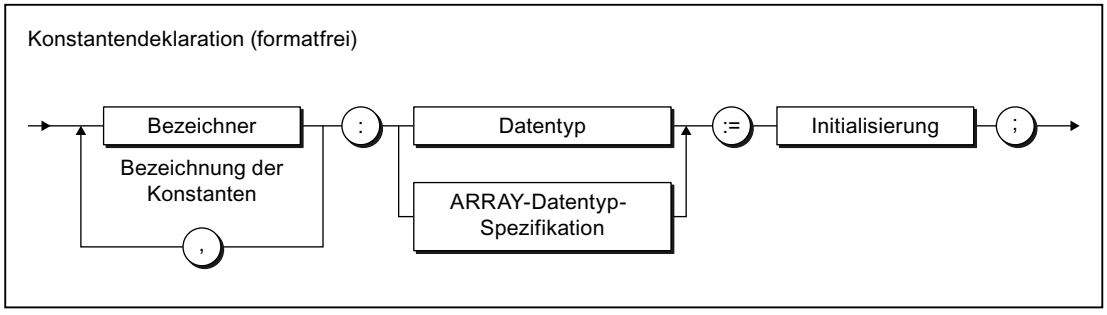


Bild A-51 Konstantendeklaration

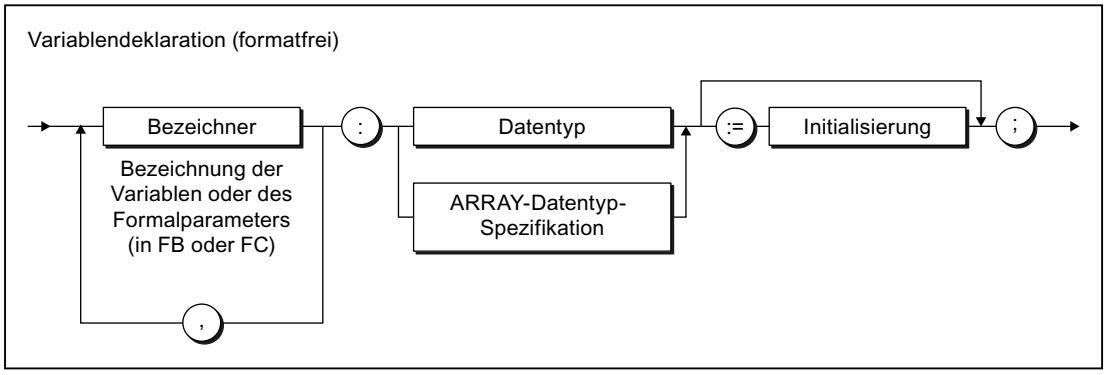


Bild A-52 Variablendeklaration

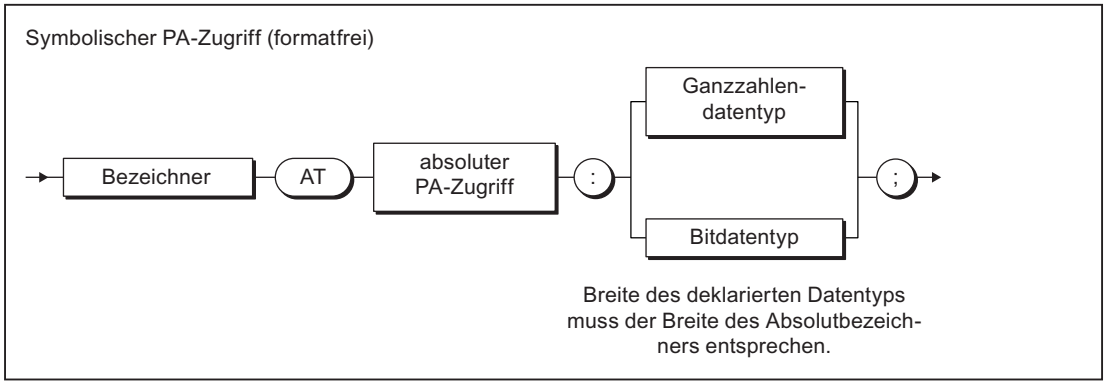


Bild A-53 Symbolischer PA-Zugriff

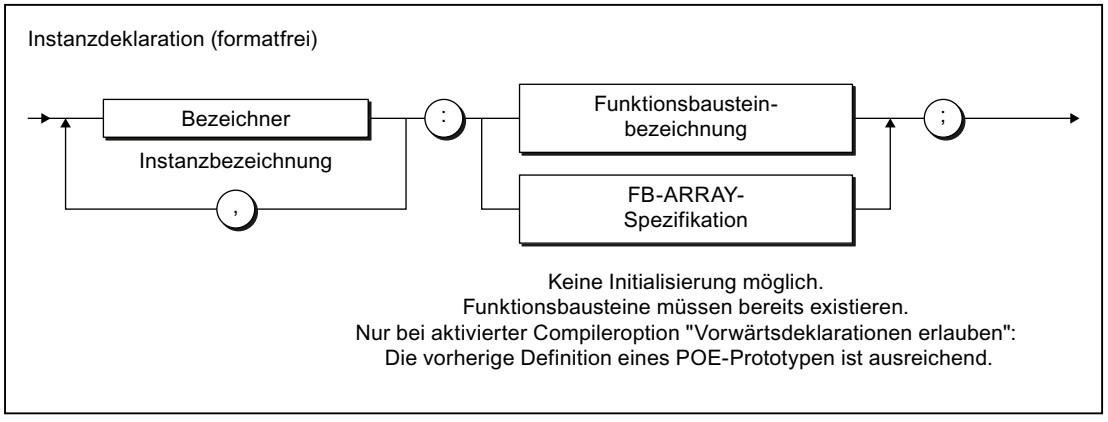


Bild A-54 Instanzdeklaration

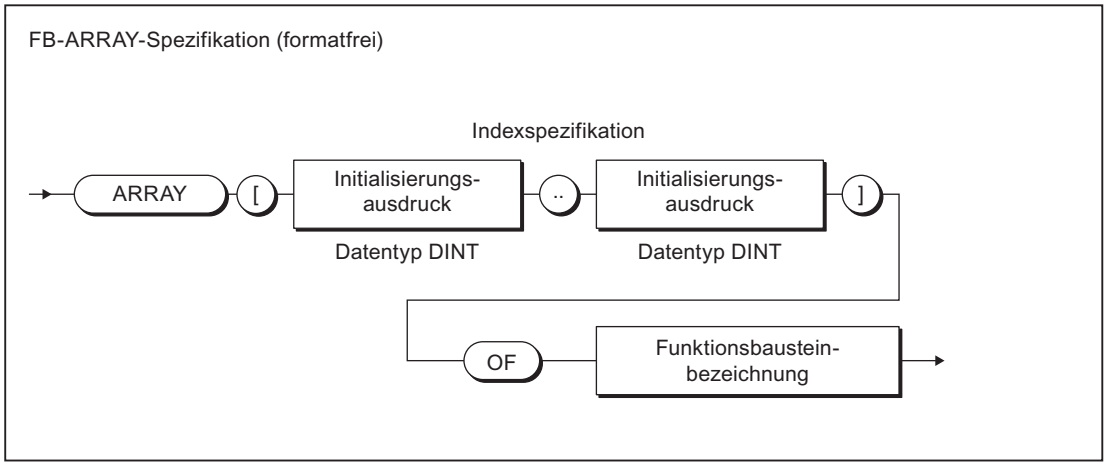


Bild A-55 FB-ARRAY-Spezifikation

Initialisierung

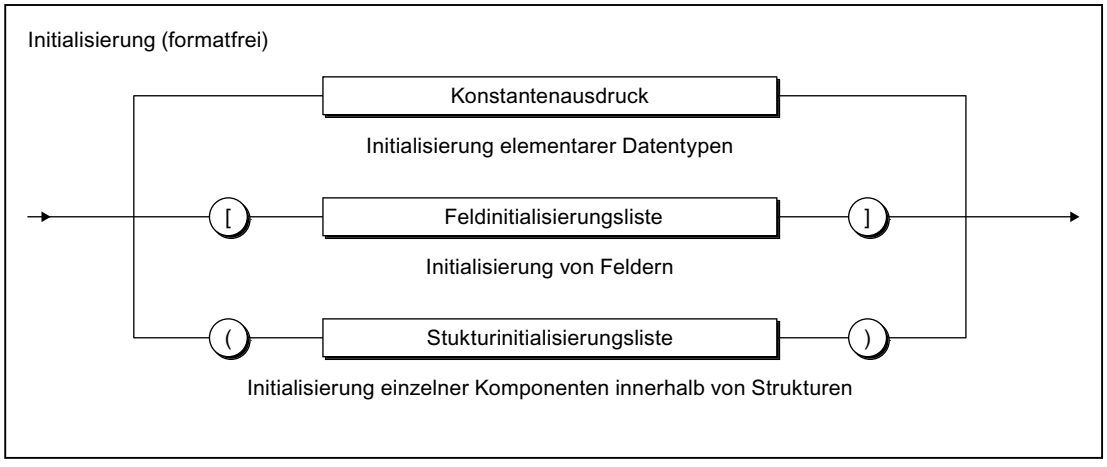


Bild A-56 Initialisierung

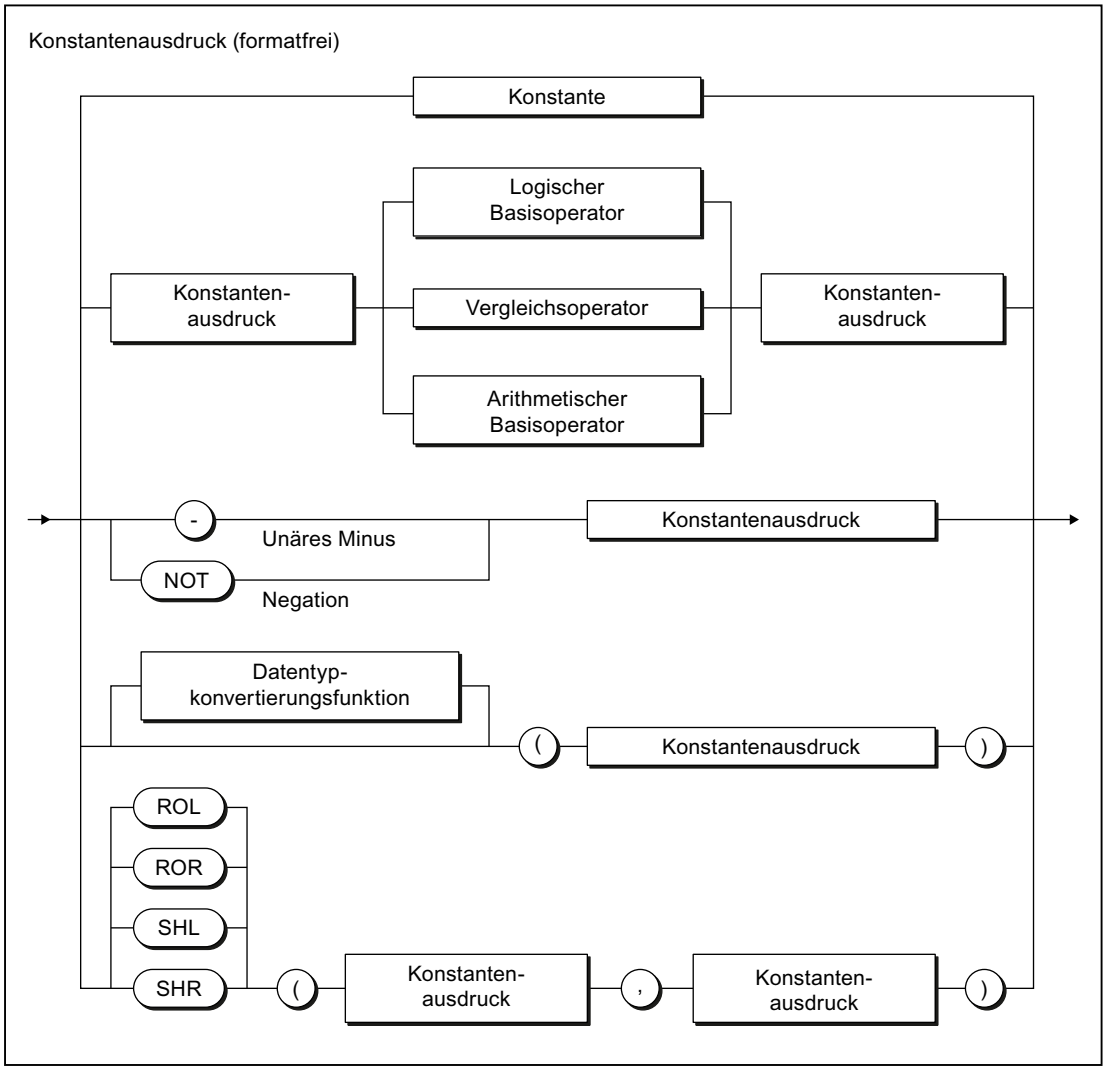


Bild A-57 Konstantenausdruck

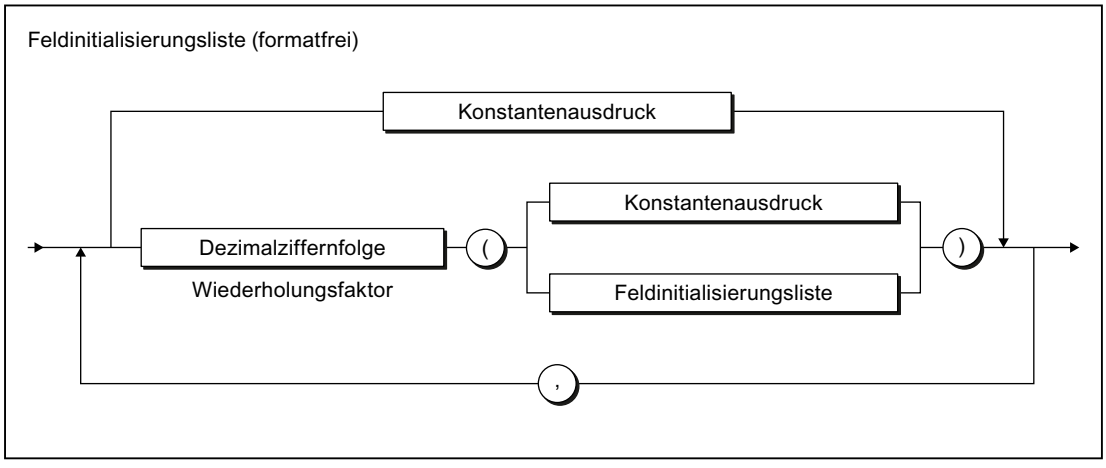


Bild A-58 Feldinitialisierungsliste

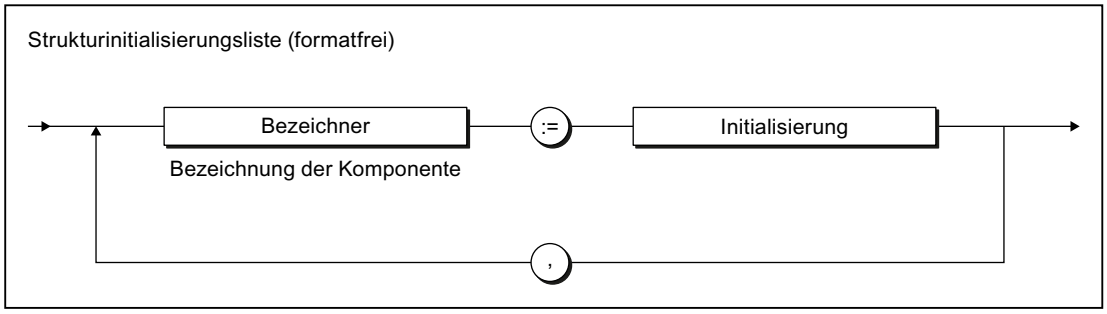


Bild A-59 Strukturinitialisierungsliste

A.1.3.9 Datentypen

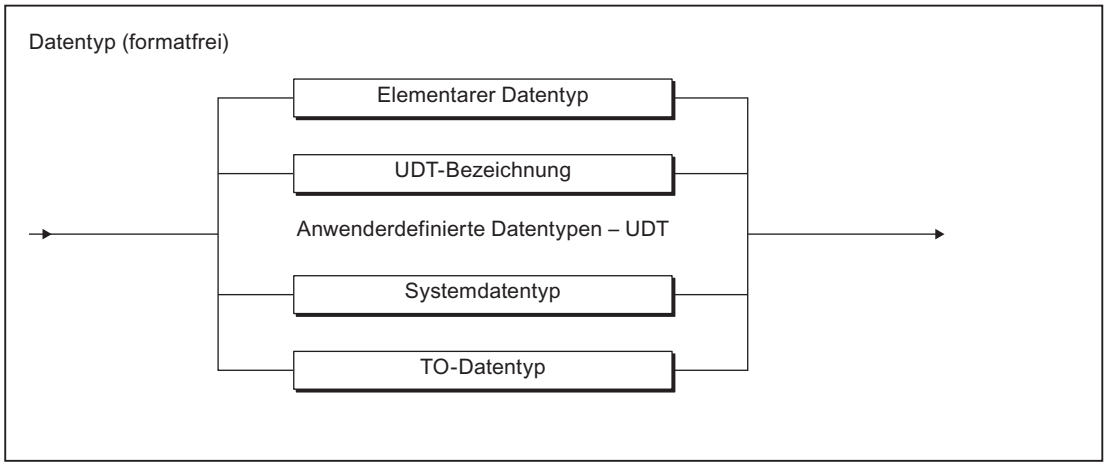


Bild A-60 Datentyp

Elementare Datentypen

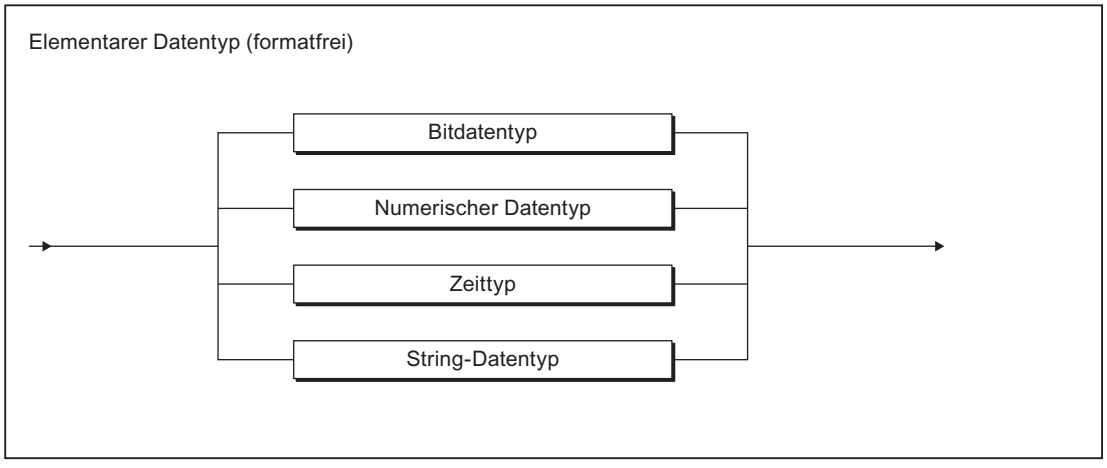


Bild A-61 Elementarer Datentyp

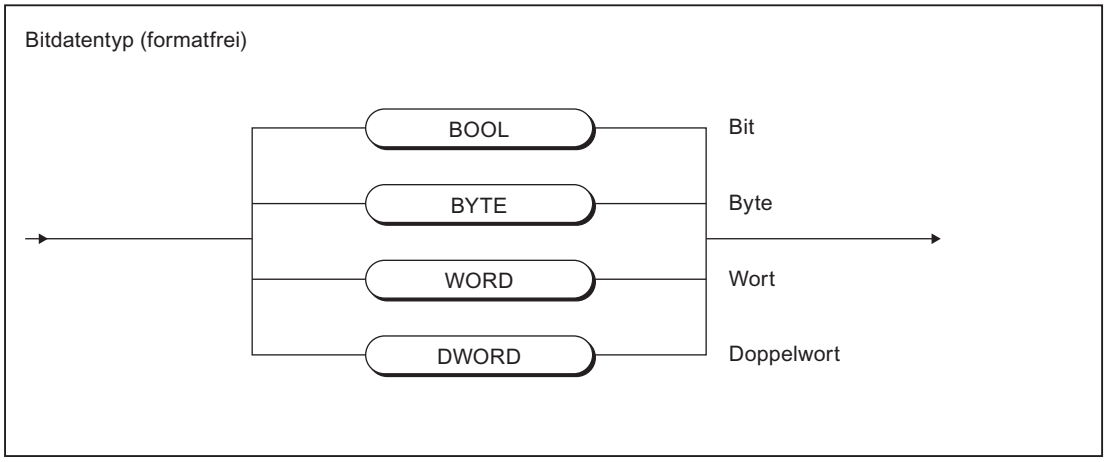


Bild A-62 Bitdatentyp

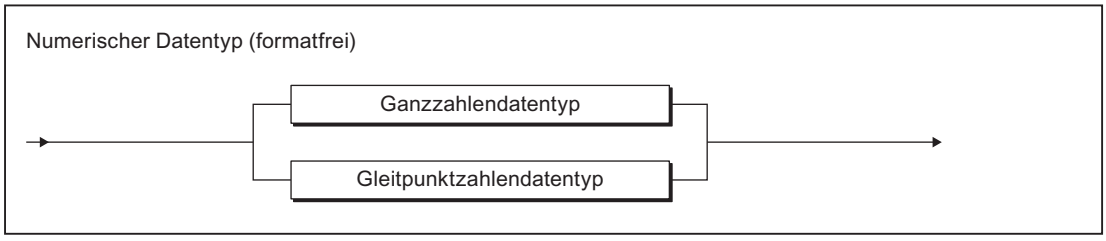


Bild A-63 Numerischer Datentyp

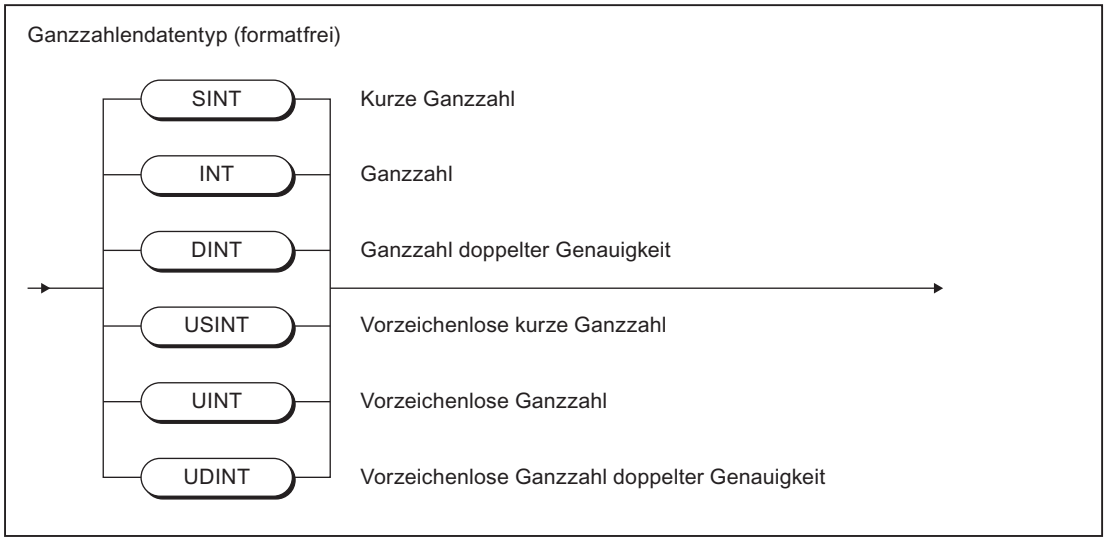


Bild A-64 Ganzzahlendatentyp

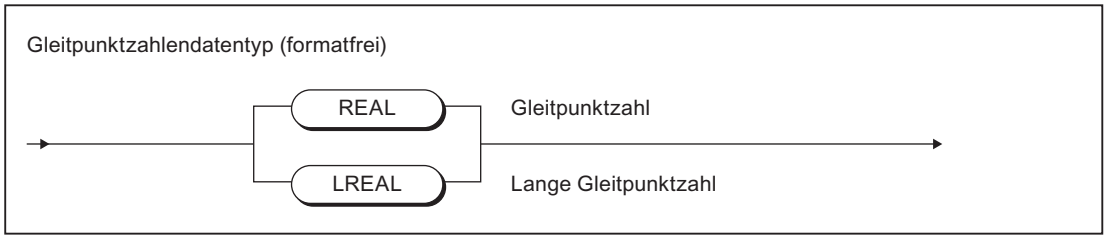


Bild A-65 Gleitpunktzahlendatentyp

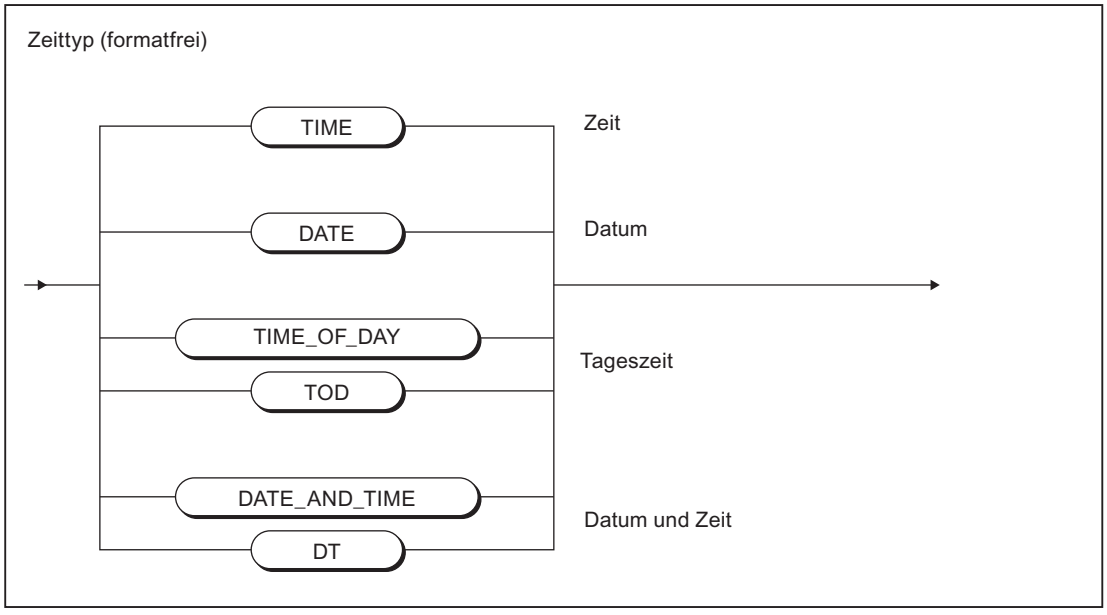


Bild A-66 Zeitdatentyp

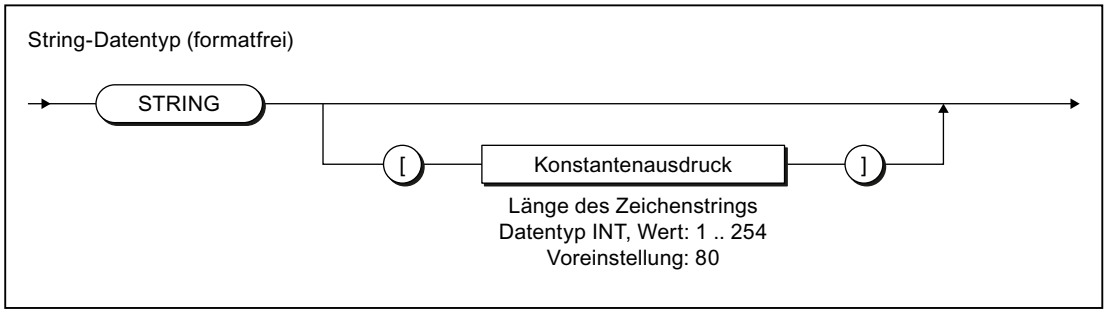


Bild A-67 String-Datentyp

Anwenderdefinierte Datentypen

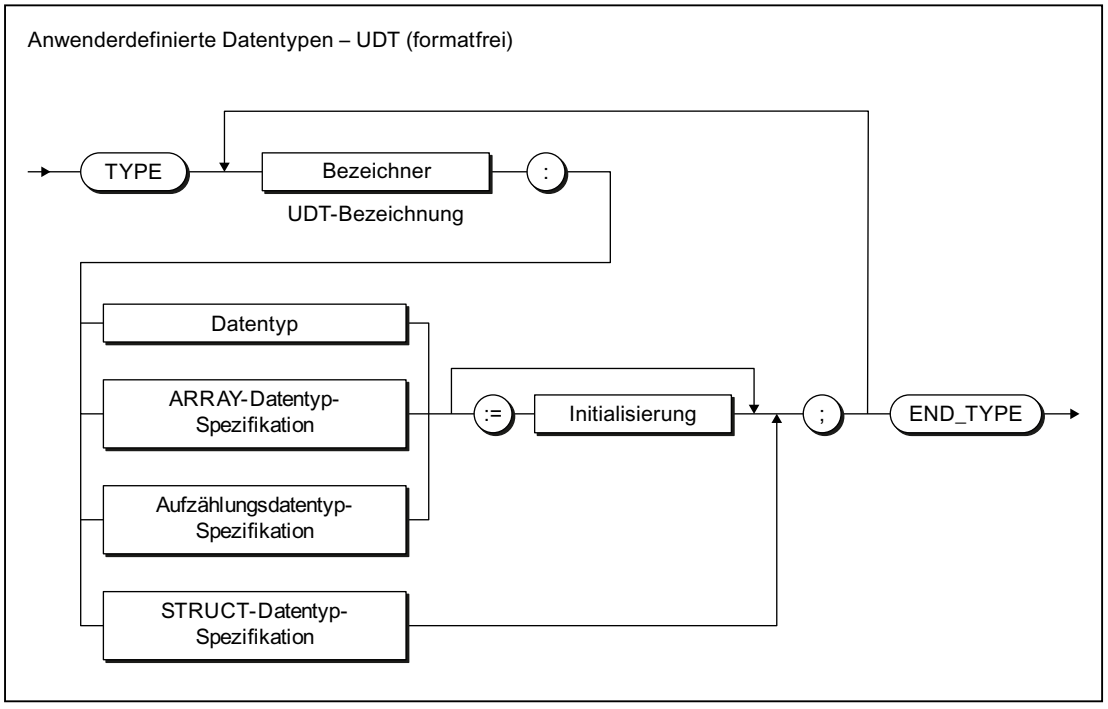


Bild A-68 Anwenderdefinierter Datentyp

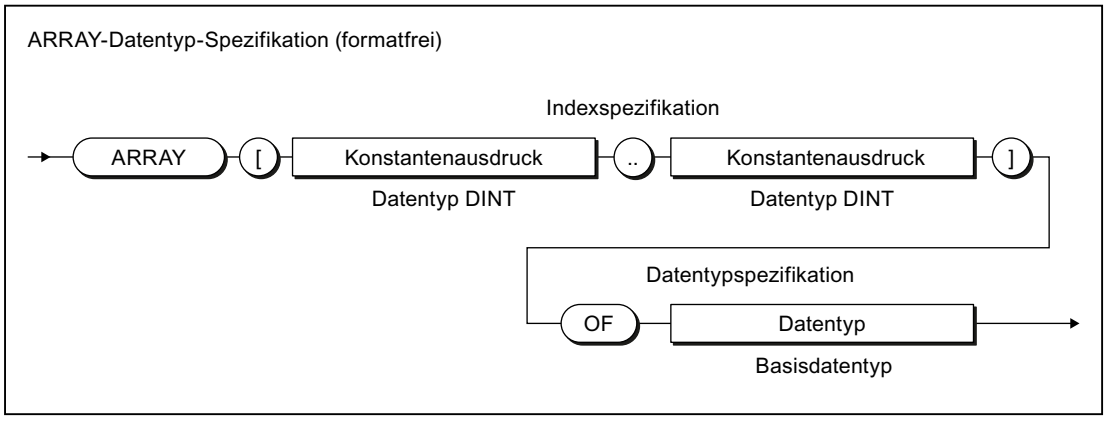


Bild A-69 ARRAY-Datentyp-Spezifikation

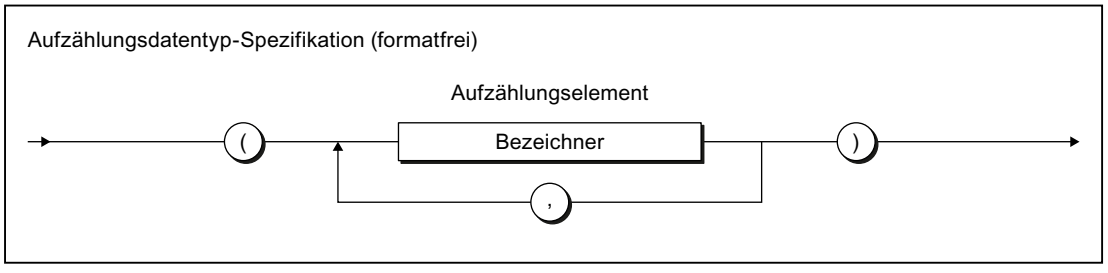


Bild A-70 Aufzählungsdatentyp-Spezifikation

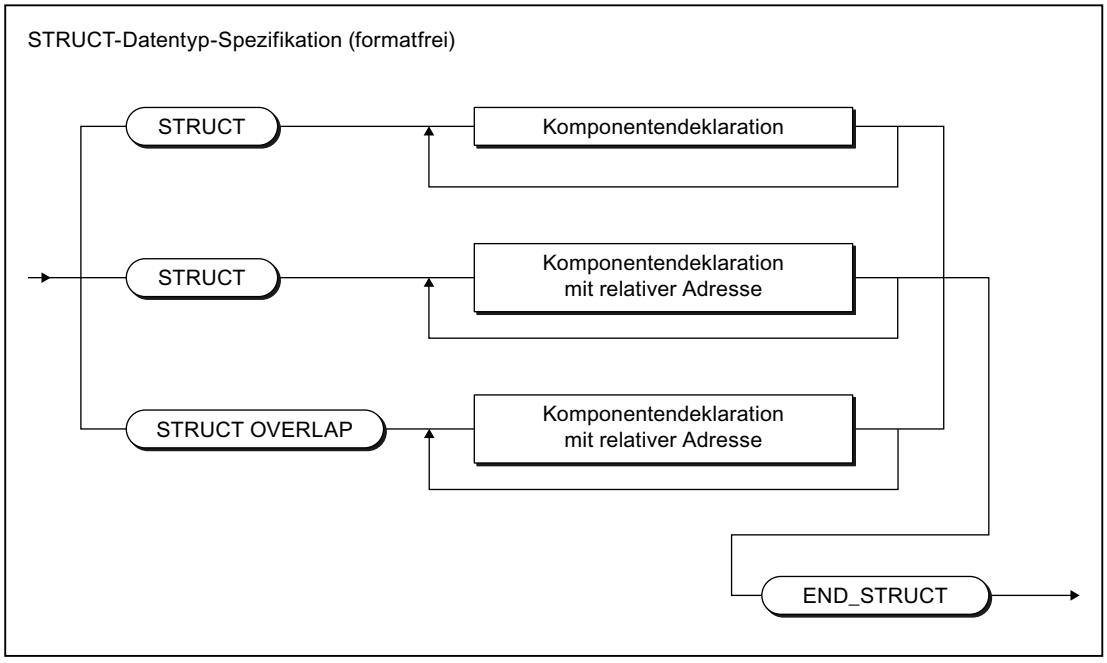


Bild A-71 STRUCT-Datentyp-Spezifikation

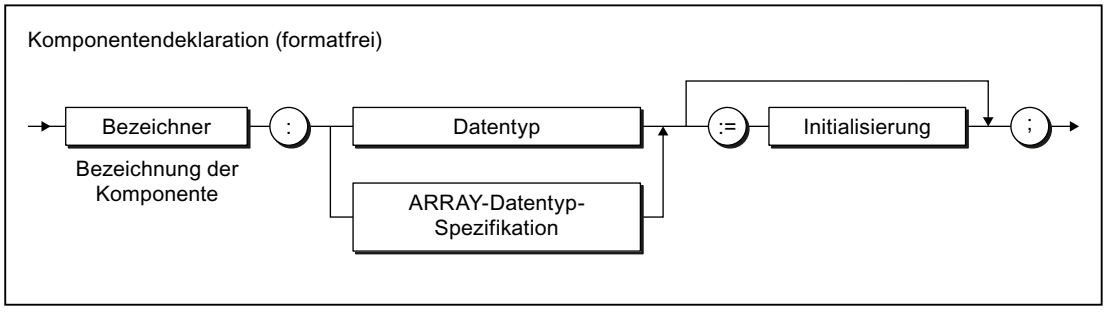


Bild A-72 Komponentendeklaration

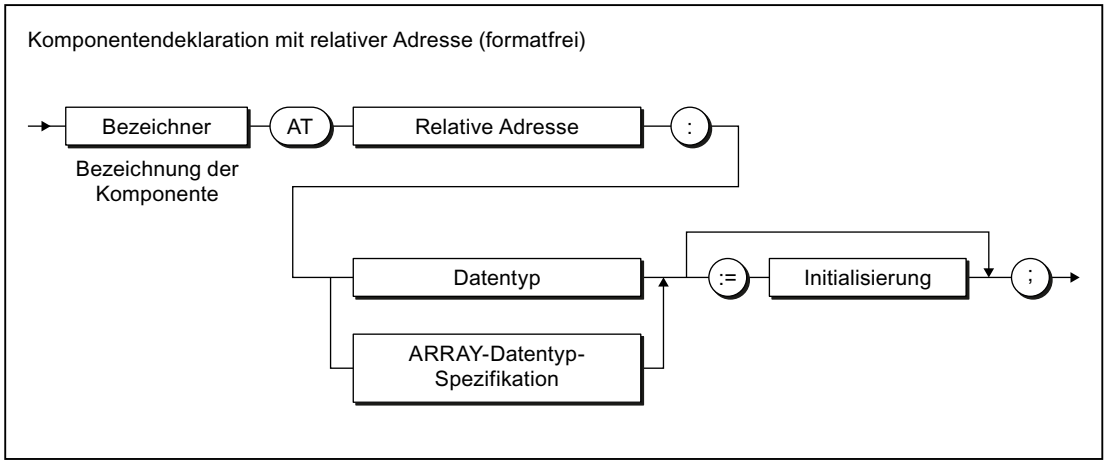


Bild A-73 Komponentendeklaration mit relativer Adresse

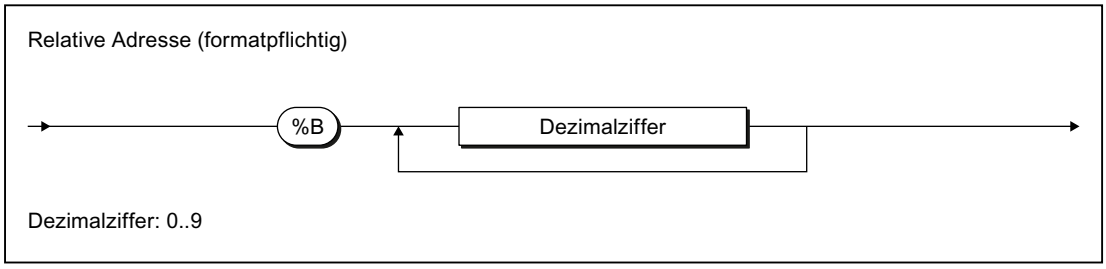


Bild A-74 Relative Adresse

A.1.3.10 Anweisungsabschnitt

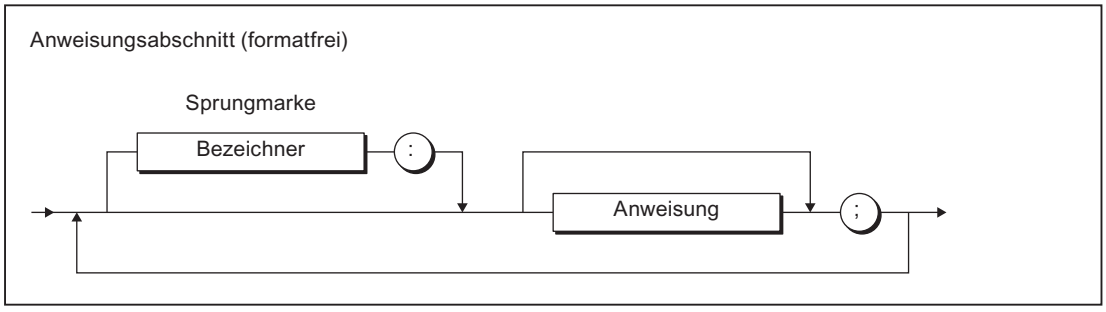


Bild A-75 Anweisungsabschnitt

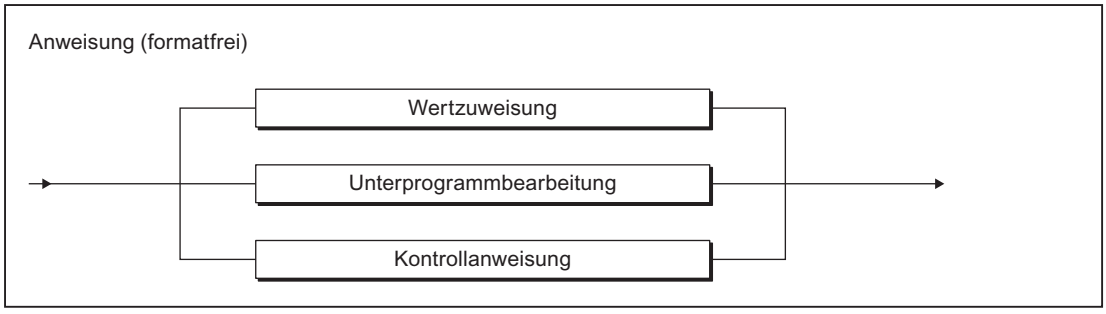


Bild A-76 Anweisung

A.1.3.11 Wertzuweisungen und Operationen

Wertzuweisung und Ausdruck

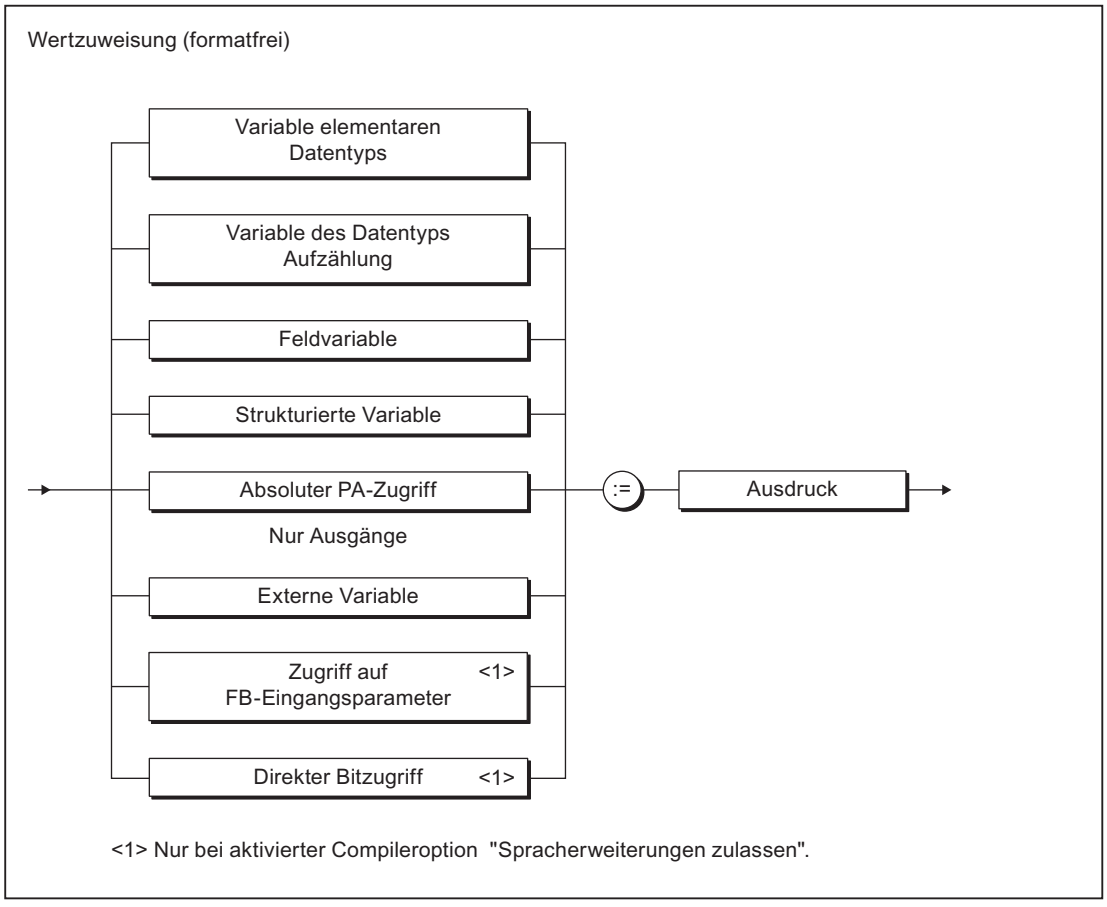


Bild A-77 Wertzuweisung

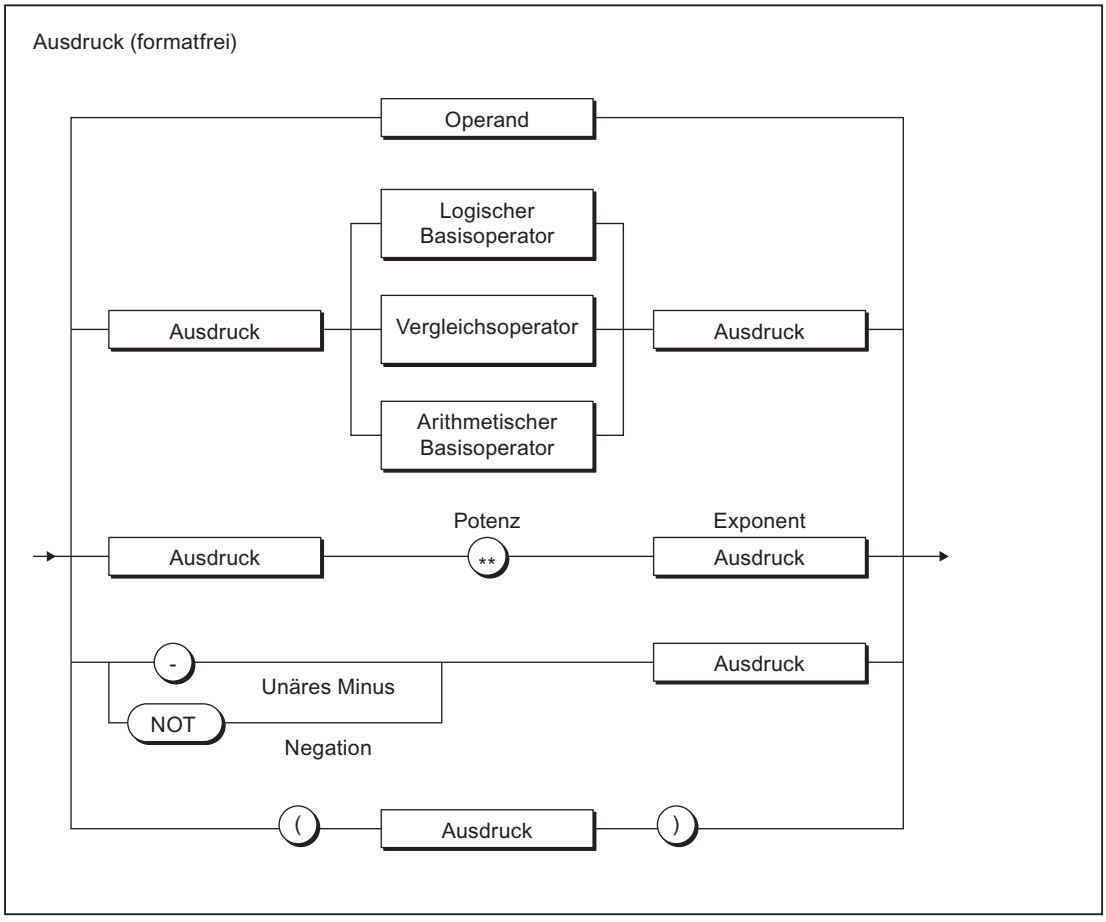


Bild A-78 Ausdruck

Operanden

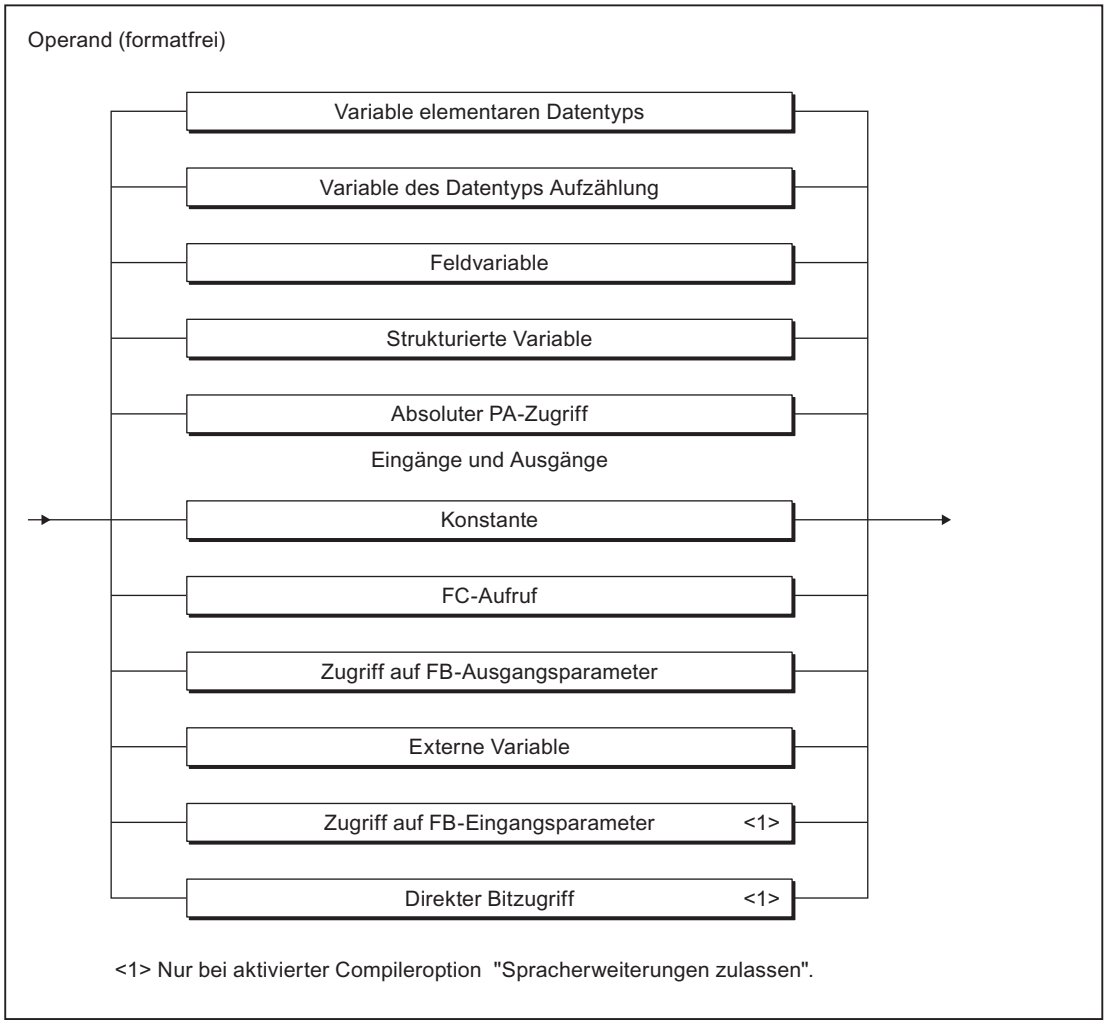


Bild A-79 Operand

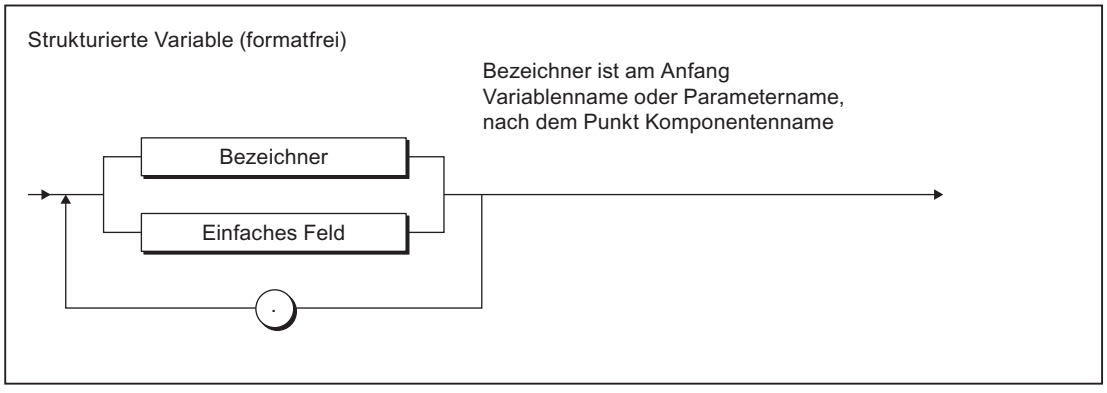


Bild A-80 Strukturierte Variable

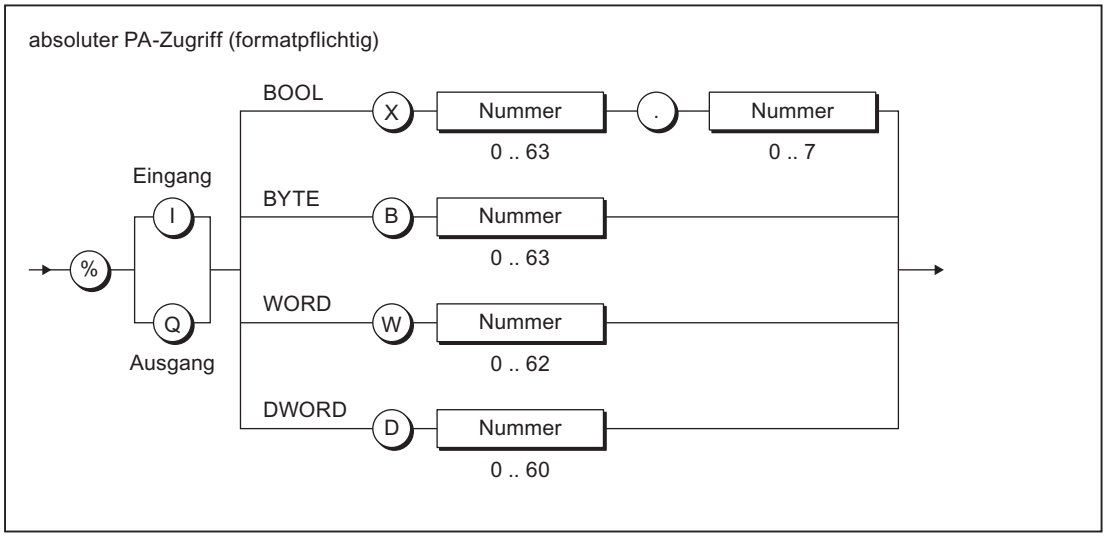


Bild A-81 Absoluter PA-Zugriff

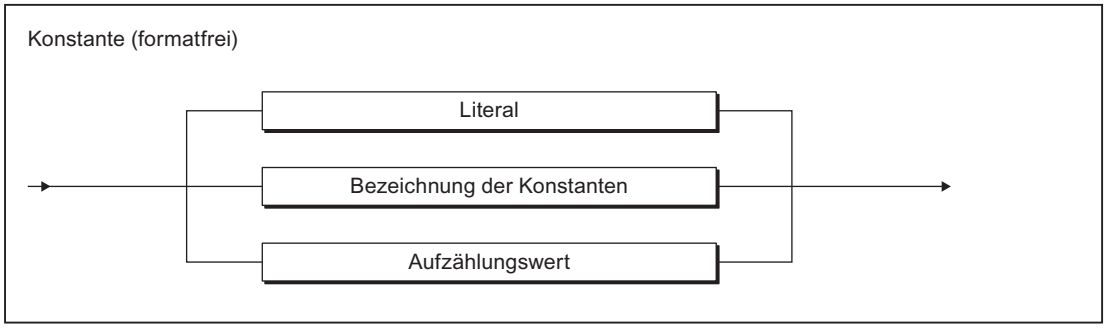


Bild A-82 Konstante

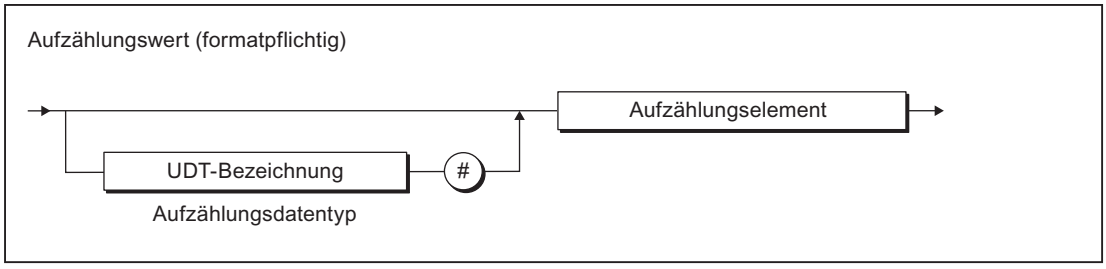


Bild A-83 Aufzählungswert

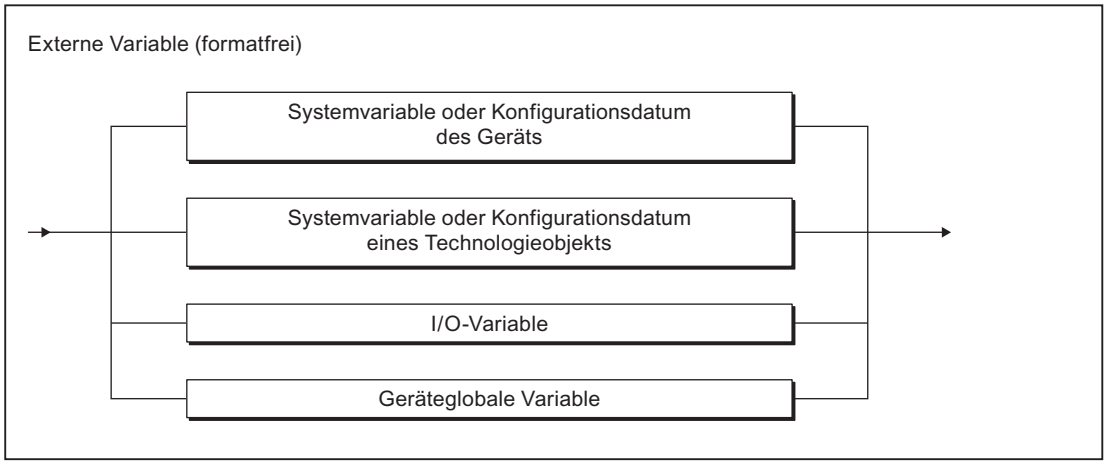


Bild A-84 Externe Variable

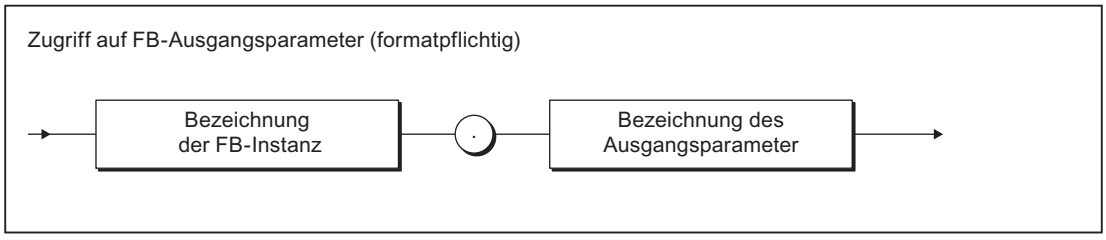


Bild A-85 Zugriff auf FB-Ausgangsparameter

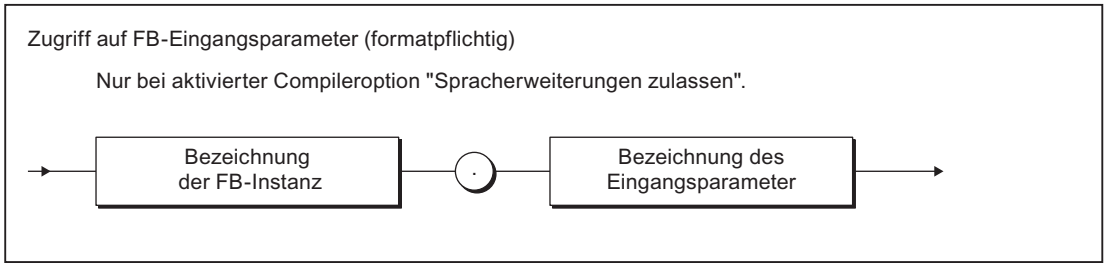


Bild A-86 Zugriff auf FB-Eingangsparameter

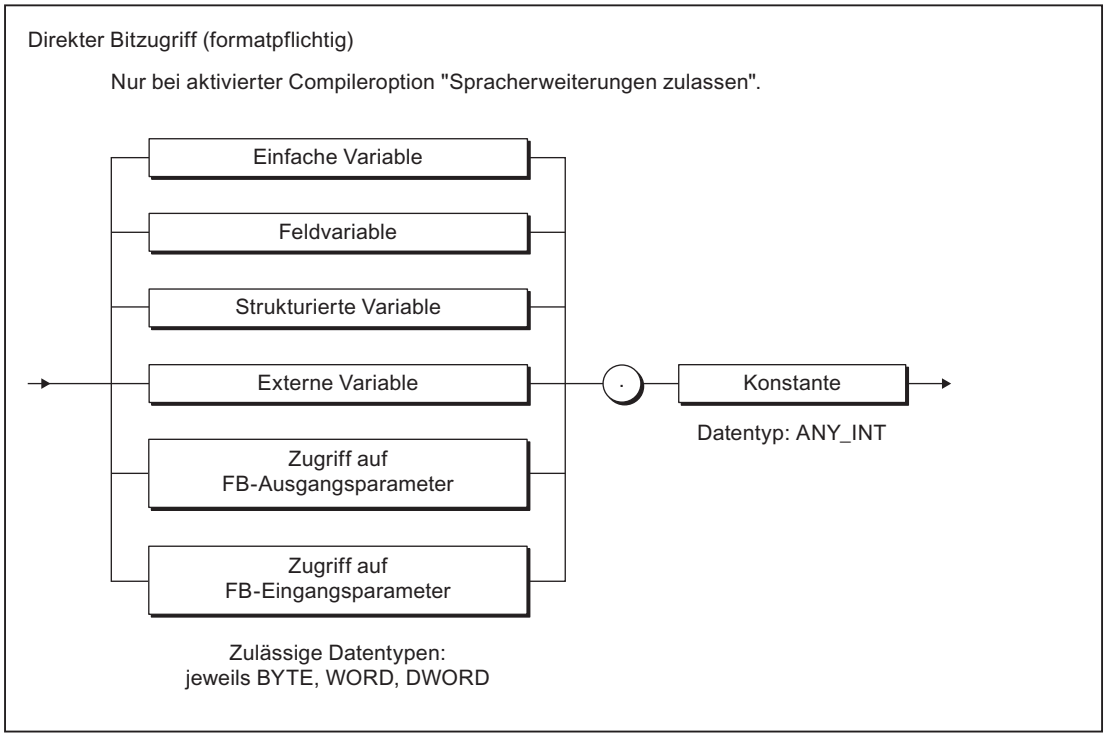


Bild A-87 Bitzugriff

Operatoren

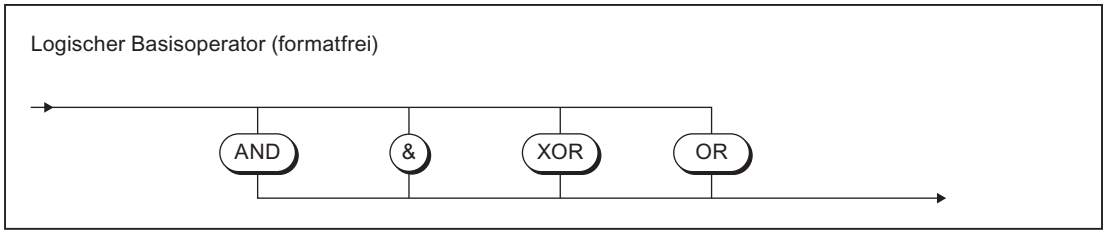


Bild A-88 Logischer Basisoperator

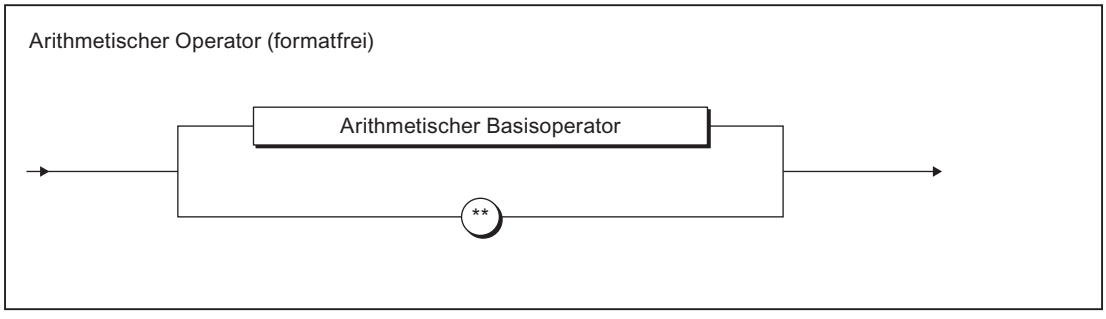


Bild A-89 Arithmetischer Operator

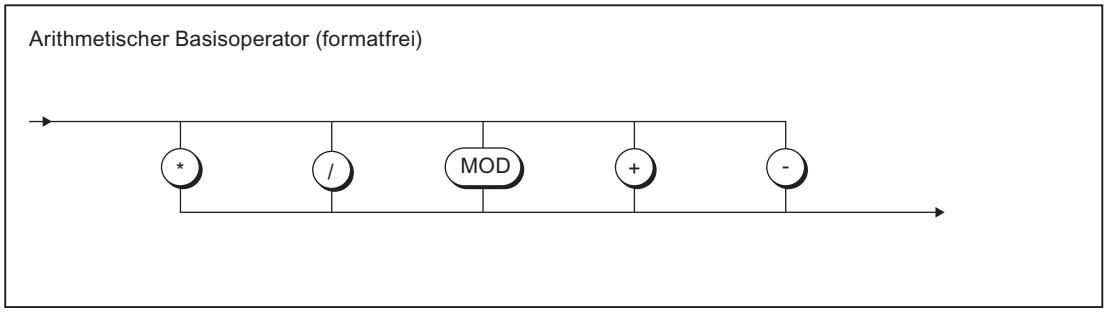


Bild A-90 Arithmetischer Basisoperator

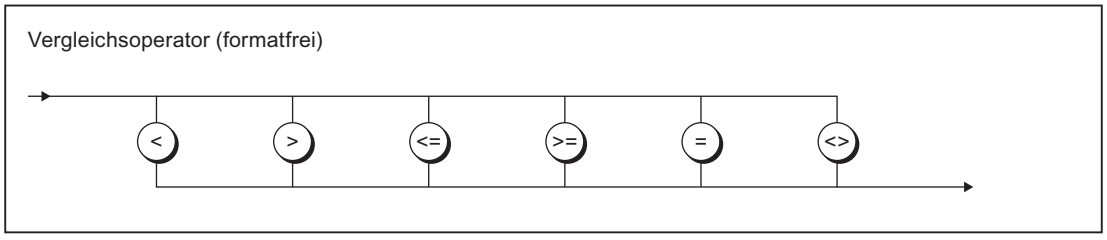


Bild A-91 Vergleichsoperatoren

A.1.3.12 Aufruf von Funktionen und Funktionsbausteinen

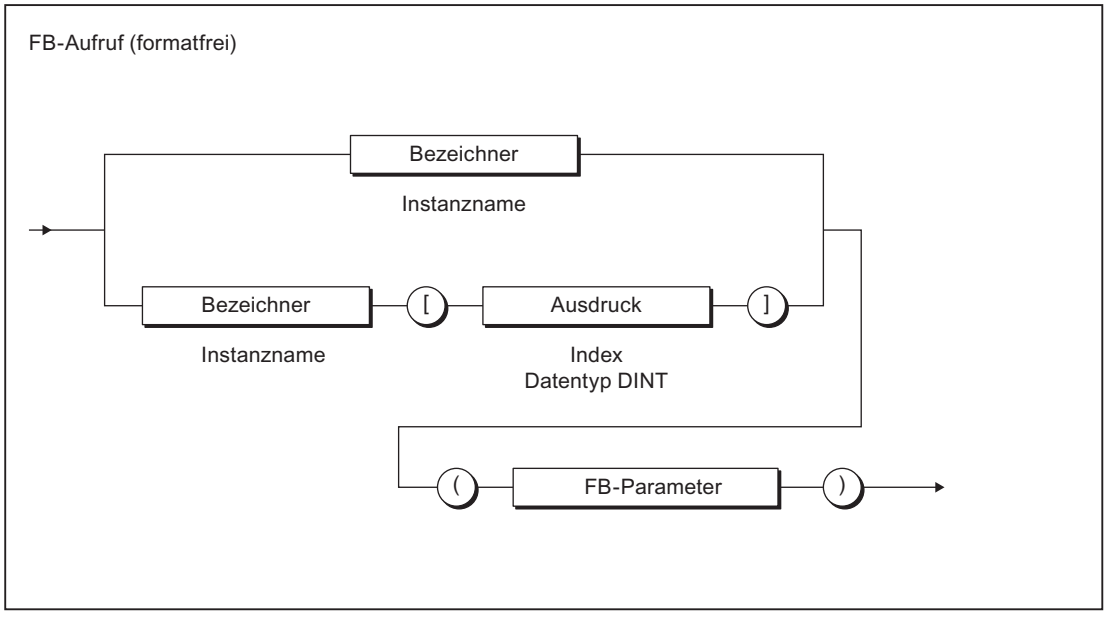


Bild A-92 FB-Aufruf

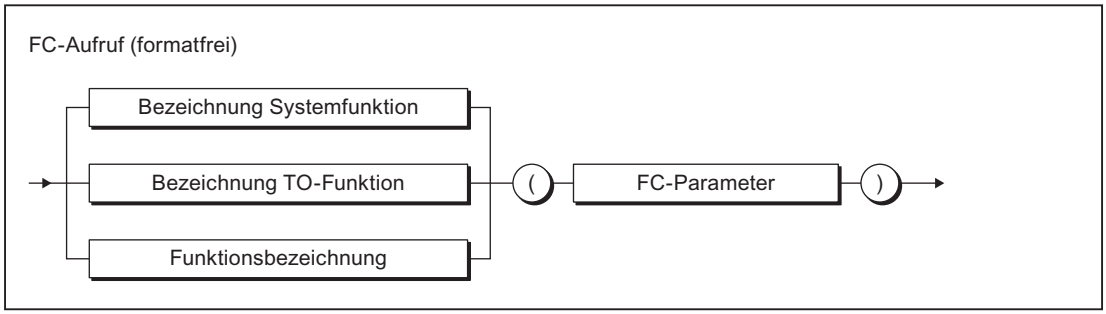


Bild A-93 FC-Aufruf

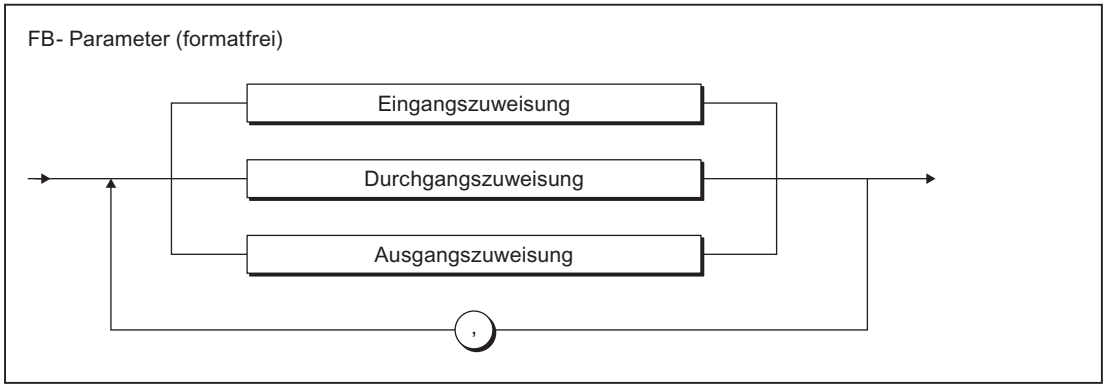


Bild A-94 FB-Parameter

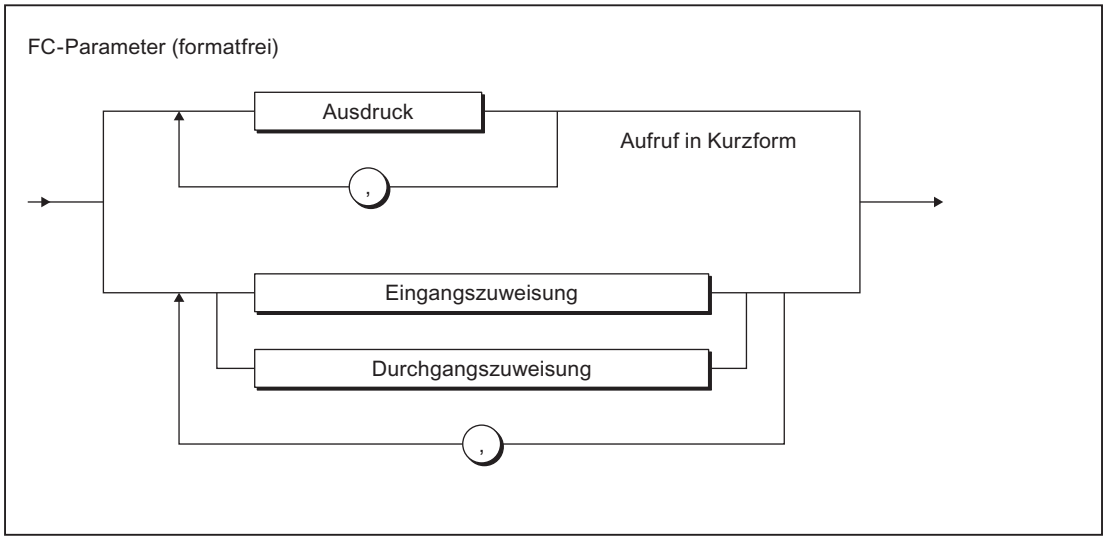


Bild A-95 FC-Parameter

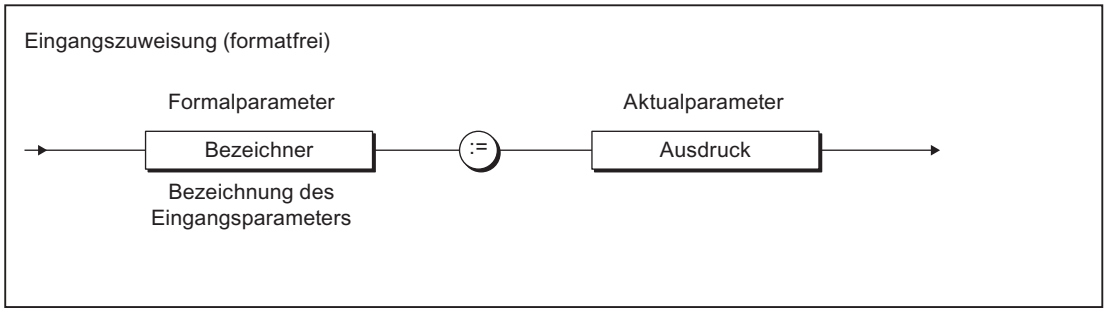


Bild A-96 Eingangszuweisung

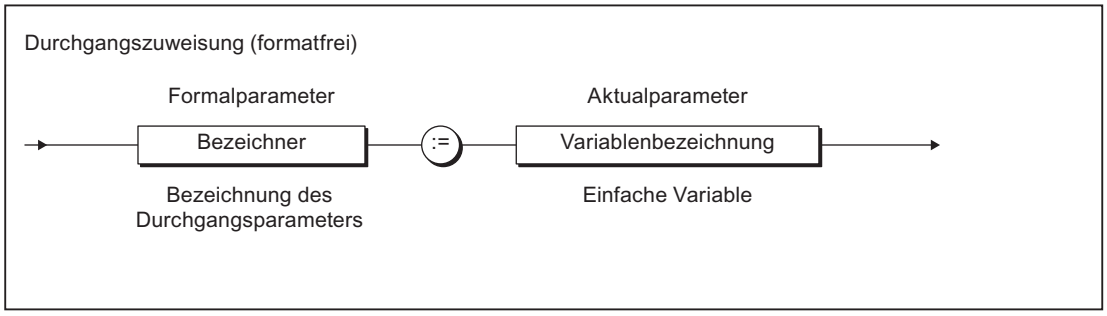


Bild A-97 Durchgangszuweisung

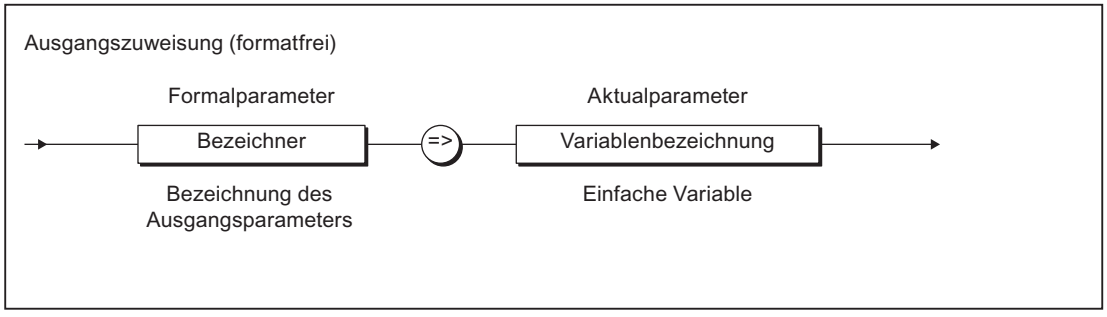


Bild A-98 Ausgangszuweisung

A.1.3.13 Kontrollanweisungen

Verzweigungen

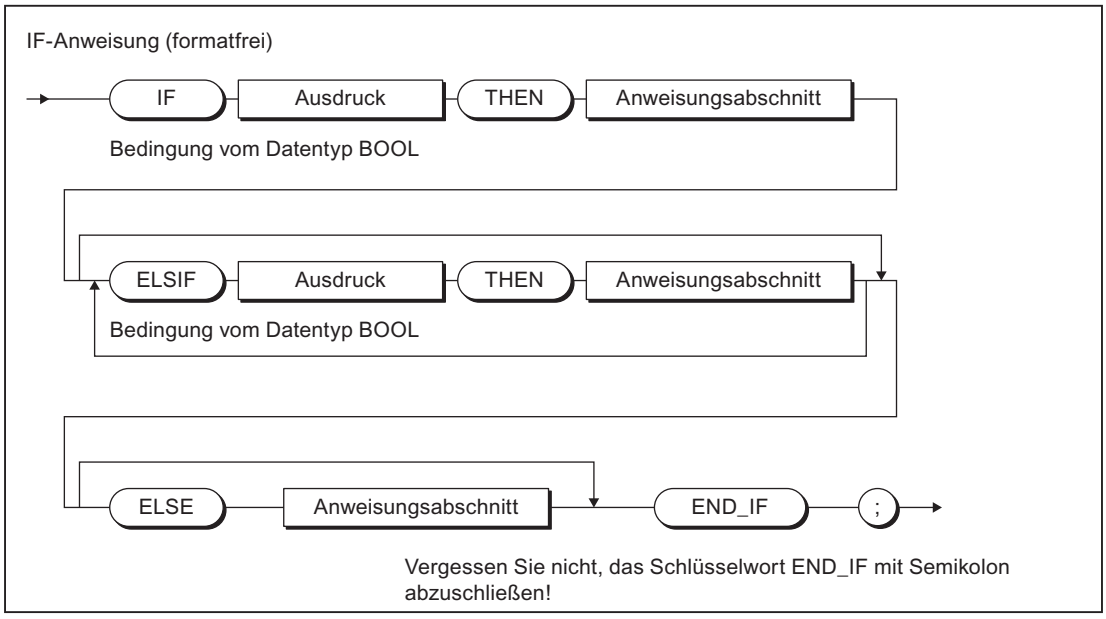


Bild A-99 IF-Anweisung

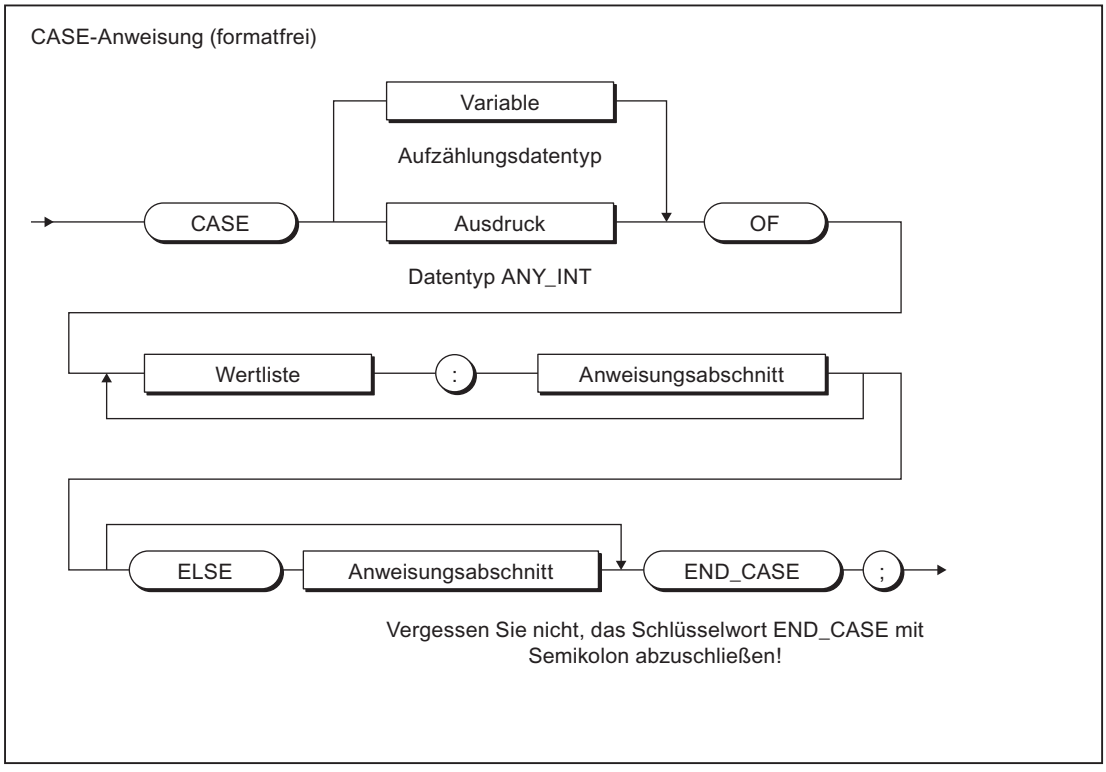


Bild A-100 CASE-Anweisung

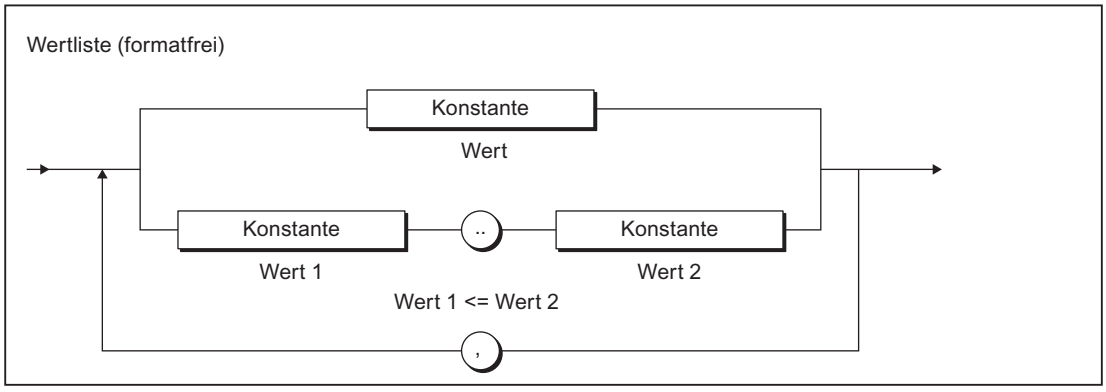


Bild A-101 Wertliste

Wiederholungs- und Sprunganweisungen

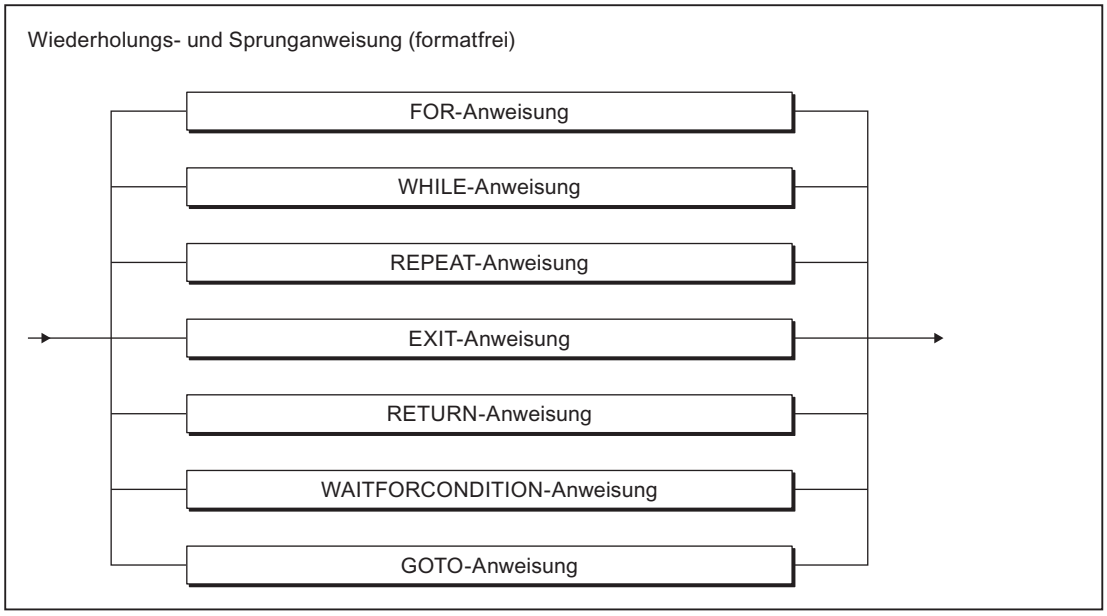


Bild A-102 Wiederholungs- und Sprunganweisung

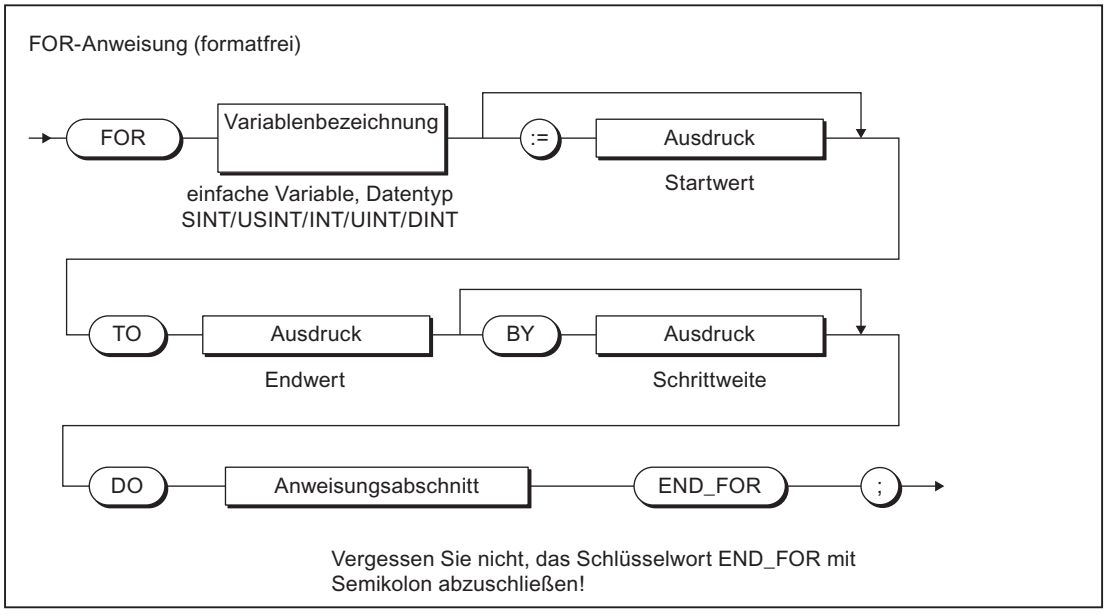


Bild A-103 FOR-Anweisung

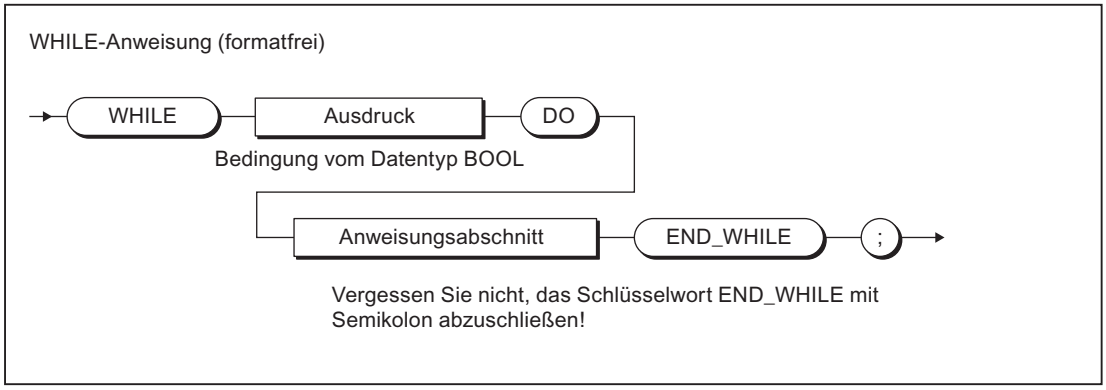


Bild A-104 WHILE-Anweisung

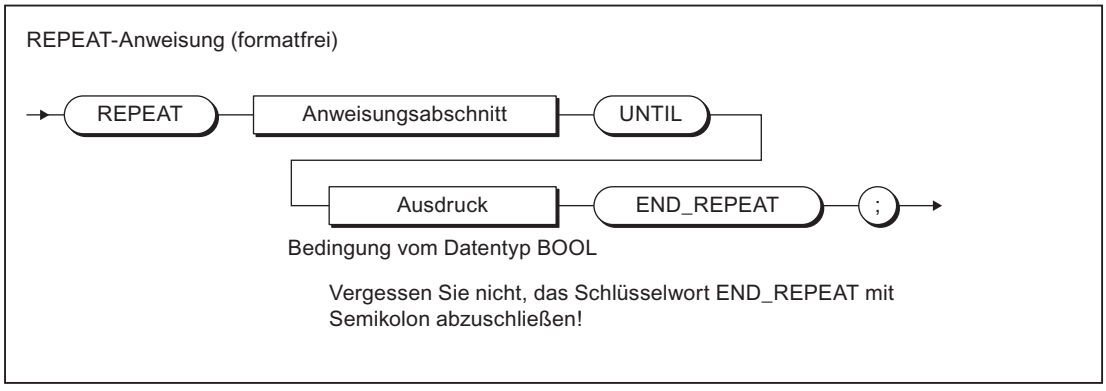


Bild A-105 REPEAT-Anweisung

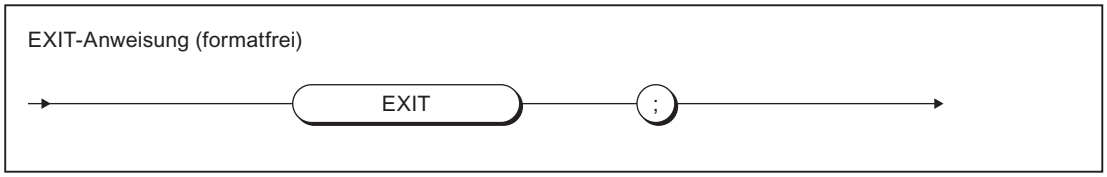


Bild A-106 EXIT-Anweisung

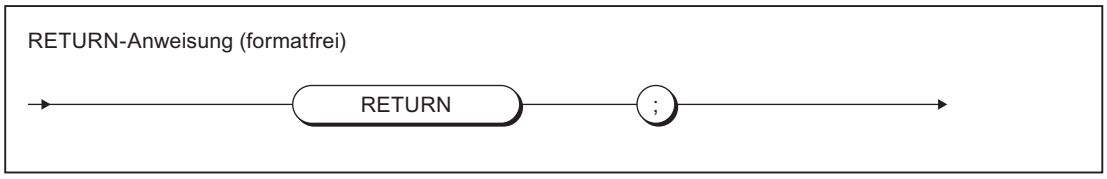


Bild A-107 RETURN-Anweisung

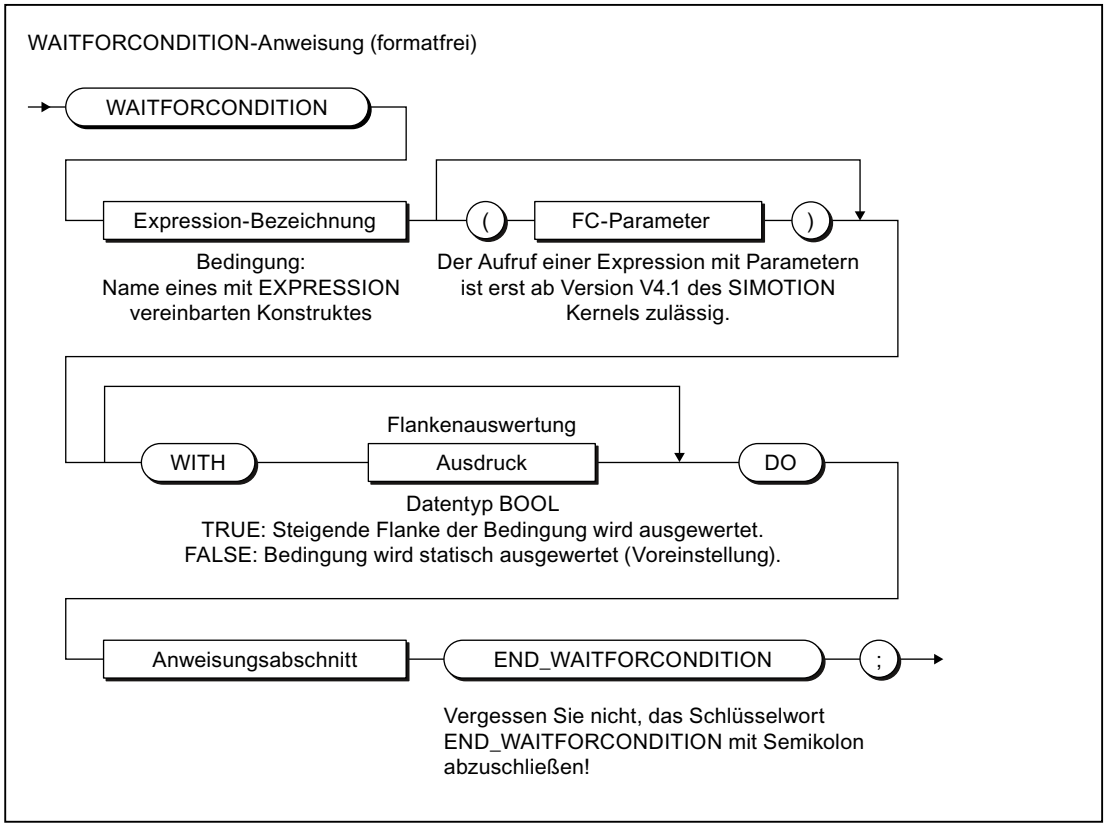


Bild A-108 WAITFORCONDITION-Anweisung

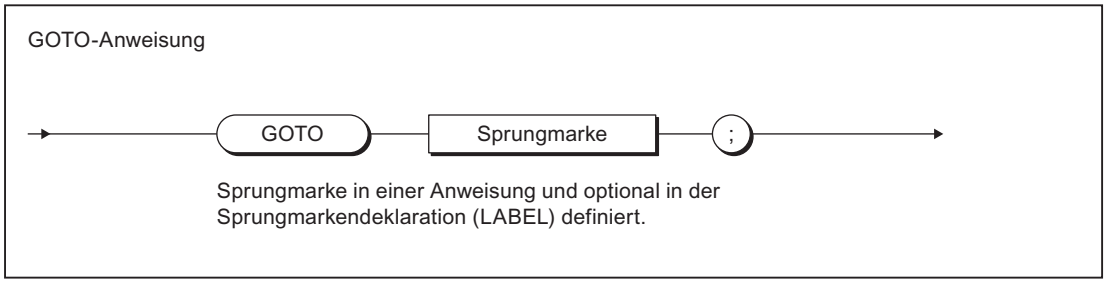


Bild A-109 GOTO-Anweisung

A.2 Fehlermeldungen des Compilers und deren Behebung

Dieses Kapitel gibt Ihnen einen Überblick über die Fehlermeldungen des Compilers und deren Behebung.

A.2.1 Dateizugriffsfehler (1000 … 1100)

Tabelle A-9 Dateizugriffsfehler (1000 … 1100)

Fehler	Beschreibung
1000	Ein Lese-Schreibfehler beim Dateizugriff ist aufgetreten.
1001	Die Datei mit den Klartext-Fehlermeldungen konnte nicht geladen werden, es ist keine textuelle Ausgabe von Fehlermeldungen möglich. Bitte benutzen Sie die Online Hilfe mittels der Fehlernummer!
1002	Der erzeugte Code konnte nicht abgelegt werden. Bitte schließen Sie einige Fenster und übersetzen Sie erneut!
1003	Ein Lese-Schreibfehler beim Öffnen der angegebenen Datei ist aufgetreten. Bitte schließen Sie die Applikation und versuchen Sie es erneut!
1100	Die Option zur Angabe einer Präprozessor-Definition enthält einen ungültigen Bezeichner als definiertes Token. Die korrekte Syntax der Aufrufoption lautet: -D identifier[=[text]] Beispiele: • -D myident // Definition von myident; diese kann mittels #ifdef abgefragt werden. • -D myident= // myident wird als leere Zeichenkette definiert • -D "myident=Das ist ein Text" // myident wird als Zeichenkette 'Das ist ein Text' definiert. Die Anführungszeichen müssen nur gesetzt werden, wenn der Ersetzungstext Leerzeichen enthält.

A.2.2 Scannerfehler (2001, 2002)

Tabelle A-10 Scannerfehler (2001, 2002)

Fehler	Beschreibung
2001	Das angegebene Zeichen ist unzulässig.
2002	Der angegebene Bezeichner enthält nicht zulässige Zeichen oder Kombinationen von Zeichen. Nach IEC61131 muss ein Bezeichner mit einem Buchstaben oder einem Unterstrich beginnen. Es dürfen beliebig viele Buchstaben, Ziffern oder Unterstriche folgen, jedoch nicht mehrere Unterstriche aufeinander.

A.2.3 Deklarationsfehler in POE (3002 … 3027)

Tabelle A-11 Deklarationsfehler in POE (3002 … 3027)

Fehler	Beschreibung
3002	Es wird das Schlüsselwort "IMPLEMENTATION" zur Kennzeichnung des Codeabschnitts der Ladeeinheit erwartet.
3003	Der angegebene Deklarationsblock ist im gegebenen Kontext nicht zulässig.
3004	Die Variablendeklarationsblöcke VAR, VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT, VAR CONSTANT sind je POE nur einmal zulässig.
3005	TASK-Anweisung: Für die angegebene Task wurde die Taskanbindung bereits in der Quelle vorgenommen. Eine weitere Taskanbindung ist nicht möglich.
3006	Die Angabe der Stackgröße der Task ist fehlerhaft. Es sind hier nur positive ganzzahlige Werte zulässig.
3007	Der angegebene Bezeichner muss ein Taskbezeichner sein, siehe Taskkonfiguration.
3008	Der angegebene Bezeichner muss ein Programmbezeichner sein. Die Vereinbarung erfolgt über die Anweisung PROGRAM xx … END_PROGRAM.
3009	Nach dem Schlüsselwort EXPRESSION muss ein Bezeichner folgen. Die Vereinbarung erfolgt über die Anweisung EXPRESSION xx … END_EXPRESSION.
3010	Der angegebene Bezeichner ist kein Bezeichner einer EXPRESSION. Überprüfen Sie, ob die Vereinbarung über die Anweisung EXPRESSION xx … END_EXPRESSION erfolgte.
3011	Die TASK-Anweisung ist in der Unit nicht zulässig. Verwenden Sie die Taskkonfiguration in der Workbench.
3012	Der angegebene Bezeichner wurde bereits an anderer Stelle deklariert. Eine nochmalige Verwendung als Bezeichner einer Funktion ist nicht möglich.
3013	Der angegebene Bezeichner wurde bereits an anderer Stelle deklariert. Eine nochmalige Verwendung als Bezeichner eines Funktionsbausteins ist nicht möglich.
3014	Die UNIT-Anweisung wird erwartet. Folgende Formen sind zulässig: • UNIT myunit; • UNIT myunit : dvtype; Die UNIT-Anweisung ist nur beim Compilieren auf ASCII-Fileniveau erforderlich. Bei Aufruf des Compilers aus der Workbench heraus ist sie optional.
3015	Die Quelle ist nicht mit END_IMPLEMENTATION abgeschlossen. Aufbau einer Quelle beachten!
3016	Nach dem Schlüsselwort END_IMPLEMENTATION dürfen keine weiteren Anweisungen angegeben werden.
3017	Die Taskdeklaration wird nicht mit END_TASK abgeschlossen. Aufbau einer Quelle beachten!
3018	Die POE-Deklaration wird nicht mit END_FUNCTION, END_FUNCTION_BLOCK oder END_PROGRAM abgeschlossen. Aufbau einer Quelle beachten!
3019	Es wird eine POE erwartet, die mit den Schlüsselwörtern FUNCTION, FUNCTION_BLOCK oder PROGRAM beginnt.
3020	Es wird die Anweisung zur Taskanbindung erwartet. Aufbau: TASK tname … END_TASK;
3022	Es wird das Schlüsselwort INTERFACE erwartet. Siehe Aufbau einer Quelle.
3023	Es wird das Schlüsselwort INTERFACE oder das Schlüsselwort IMPLEMENTATION erwartet. Siehe Aufbau einer Quelle.
3024	Syntaxfehler in der TASK-Anweisung. Korrekter Aufbau: TASK tname … END_TASK;
3025	Der angegebene Bezeichner wurde bereits an anderer Stelle deklariert. Eine nochmalige Verwendung als Bezeichner eines Programms ist nicht möglich.

Fehler	Beschreibung
3026	Die WAITFORCONDITION-Anweisung kann nicht rekursiv benutzt werden. Es wurde versucht, innerhalb einer WAITFORCONDITION-Anweisung ein zweites Mal WAITFORCONDITION zu benutzen. Das ist nicht möglich.
3027	Es wurde versucht, innerhalb eines Blocks EXPRESSION … END_EXPRESSION eine WAITFORCONDITION-Anweisung einzufügen. Das ist nicht möglich. Die WAITFORCONDITION-Anweisung kann nicht innerhalb einer Expression verwendet werden.

A.2.4 Deklarationsfehler in Datentypdeklarationen (4001 … 4105)

Tabelle A-12 Deklarationsfehler in Datentypdeklarationen (4001 … 4105)

Fehler	Beschreibung
4001	Der angegebene Bezeichner ist ein Standardfunktionsbezeichner, der nicht überschrieben werden kann. Wählen Sie einen anderen Bezeichner.
4002	Der angegebene Bezeichner wurde bereits verwendet. Die Benutzung als Typbezeichner ist nicht möglich. Wählen Sie einen anderen Bezeichner.
4003	Der angegebene Bezeichner wurde bereits verwendet. Die Benutzung als Konstantenbezeichner ist nicht möglich. Wählen Sie einen anderen Bezeichner.
4004	Der angegebene Initialisierungswert hat ein falsches Format. Wählen Sie den Initialisierungswert entsprechend der Datentypdeklaration.
4005	Syntaxfehler in der Typdeklaration.
4006	Syntaxfehler bei der Angabe eines Strukturelements in der Strukturdeklaration.
4007	Syntaxfehler bei der Deklaration eines ARRAY-Datentyps.
4008	Syntaxfehler bei der Angabe einer Bezeichnerliste. Die Bezeichner sind durch Komma zu trennen.
4009	Der angegebene Konstantenbezeichner wurde mit unterschiedlichen Werten belegt. Dies tritt bei der Deklaration von Aufzählungsdatentypen auf. Gleiche Aufzählungselemente in unterschiedlichen Aufzählungsdatentypen müssen bei der Typdeklaration an derselben Position stehen.
4010	Der angegebene Typbezeichner wird von der Quelle nicht exportiert, obwohl die POE, in der er verwendet wird, exportiert ist. Verwenden Sie einen anderen Datentyp oder deklarieren Sie den Datentyp im Implementationsabschnitt.
4011	Eine Konstantendeklaration erfordert zwingend die Angabe eines Initialisierungswertes. Beispiel: x : DINT := 5;
4012	Der angegebene Datentyp muss außerhalb der POE deklariert sein. Bei VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT dürfen die Typbezeichner nicht lokal in der POE deklariert sein, da sie zum Zweck der Parameterübergabe auch außerhalb bekannt sein müssen.
4013	Der angegebene Wert wird im Aufzählungsdatentyp mehrfach verwendet. Die Werte im Aufzählungsdatentyp müssen sich jedoch voneinander unterscheiden.
4020	Der angegebene Bezeichner wurde bereits als Datentypbezeichner verwendet. Die Definition unterscheidet sich jedoch von der aktuellen Definition. Dies ist nicht zulässig. Wählen Sie entweder einen anderen Bezeichner oder passen sie die Typdefinitionen an. Tritt diese Meldung beim Laden von Bibliotheken oder Technologiepaketen auf, so können sie hier auch Namensräume verwenden (z. B. USELIB mylib AS Namespace_1).
4050	Die Datentyp- oder Variablendeklaration erzeugt einen Datentyp, der größer ist als die angegebene maximal zulässige Datengröße.
4051	Die Variablendeklaration benötigt einen Speicherbereich, der größer ist als die angegebene maximal zulässige Bereichsgröße.

Fehler	Beschreibung
4100	Die Definition der Strukturkomponente erfordert die Angabe eines anwenderdefinierten Offsets. Werden bei einer Strukturdefinition Offsets explizit vorgegeben, so ist die Offset-Angabe an allen Strukturkomponenten notwendig. Des weiteren erfordert das Schlüsselwort OVERLAP die explizite Vorgabe der Offsets.
4101	Die bei der Definition der Strukturkomponente angegebene Offsetangabe ist nicht zulässig, da bereits eine Komponente ohne explizite Offsetangabe definiert wurde. Werden bei einer Strukturdefinition Offsets explizit vorgegeben, so ist die Offset-Angabe an allen Strukturkomponenten notwendig.
4102	Der bei der Definition der Strukturkomponente vorgegebene Offsetwert ist nicht zulässig. Der Wert muß ein Vielfaches der in der Fehlermeldung angegebenen Zahl sein, damit die Datenelemente im richtigen Alignment vorliegen.
4103	Die bei der Strukturdefinition vorgegbenen Offsetwerte führen zu Überlappungen im Speicherlayout. Dies ist nur bei Kennzeichnung der Struktur als OVERLAP zulässig.
4105	Die Überlappungen im Speicherbereich sind für folgende Datentypen nicht zulässig: STRING, ANYOBJECT und alle davon abgeleiteten TO-Referenzen!

A.2.5 Deklarationsfehler in Variablendeklarationen (5001 … 5509)

Tabelle A-13 Deklarationsfehler in Variablendeklarationen (5001 … 5509)

Fehler	Beschreibung
5001	Der angegebene Konstantenwert führt zu einer Wertebereichsüberschreitung und kann nicht in den geforderten Typ konvertiert werden.
5002	Der angegebene Bezeichner wurde bereits verwendet. Die Benutzung als Variablenbezeichner ist nicht möglich. Wählen Sie einen anderen Bezeichner.
5003	Syntaxfehler in der Variablendeklaration.
5004	Es wird die Angabe eines Datentyps erwartet (einfacher oder abgeleiteter Datentyp).
5005	Der angegebene Konstantenwert ist vom falschen Datentyp oder führt zu einer Wertebereichsüberschreitung.
5006	Bei Initialisierung eines Arrays ist die Anzahl der Initialisierungswerte zu prüfen.
5007	Syntaxfehler in der Angabe der Zeit- und Datumsliterale.
5008	Eine Instanz eines Funktionsbausteins kann an der angegebenen Stelle nicht angelegt werden. Beispielsweise können keine FB-Instanzen in Funktionen erzeugt werden. Auch Ausgangsparameter (VAR_OUTPUT) von Funktionsbausteinen können keine FB-Instanzen sein.
5009	Der in der Deklaration angegebene Datentyp ist auf die Variable mit absoluter Adresse nicht anwendbar. Es muss ein Ganzzahl- oder Bitdatentyp mit passender Bitbreite verwendet werden.
5010	Es wurde versucht, einer Variablen eine Speicheradresse zuzuweisen. Das ist an der angegebenen Stelle nicht möglich. Verwenden Sie diese Zuweisung nur innerhalb der VAR_GLOBAL-Deklaration einer Unit oder innerhalb der VAR-Deklaration eines PROGRAM.
5012	Die Vorbelegung der angegebenen Variablen mit einem Initialisierungswert ist nicht möglich.
5014	Die Initialisierung einer Datenstruktur ist fehlerhaft. Der Initialisierungswert für eine Komponente wurde mehrfach angegeben.
5016	Die Initialisierung von Variablen und Datentypen mit Technologieobjekten, die im Projekt definiert sind, ist nicht möglich. Technologieobjekte sind selbst Variablen und können deshalb nicht zur Initialisierung verwendet werden.
5100	Die Vorbelegung der angegebenen Variablen mit einem Initialisierungswert ist nicht möglich.

Fehler	Beschreibung
5110	Die Angabe eines Sonderzeichens über $… sind folgende Angaben möglich: $$, $', $L, $N, $P, $R, $T. Außerdem kann der numerische Wert eines Zeichens mittels $xx erfolgen, wobei xx für die zweistellige hexadezimale Angabe des Zeichencodes steht.
5111	Die Angabe des Sonderzeichens ist nur über $… möglich. Dies betrifft: $L, $N, $P, $R, $T
5112	Mehrzeilige Zeichenkettenkonstanten sind nicht zulässig. Um einen Zeilenumbruch in der Ausgabe zu erzeugen, verwenden Sie in der Zeichenkette die entsprechenden Sonderzeichen, z. B. $N, $R $L.
5200	Die Datentypdefinition enthält entweder direkt oder indirekt eine Rekursion. Dies ist nicht zulässig. Entfernen sie die Verwendung dieses Datentyps an der betreffenden Stelle.
5201	Der Funktionsaufruf erzeugt entweder direkt oder indirekt eine Rekursion. Dies ist nicht zulässig. Entfernen sie den Aufruf der Funktion an der betreffenden Stelle.
5500	Der angegebene Sprungmarkenbezeichner wurde bereits definiert. Verwenden Sie einen anderen Namen.
5501	Der angegebene Sprungmarkenbezeichner wurde nicht definiert. Nehmen Sie diesen Bezeichner in die Deklaration der LABEL auf.
5502	Der Sprungmarkenbezeichner wurde mehrfach zugeordnet. Jede Sprungmarke darf jedoch nur einmal als Marke verwendet werden.
5503	Die Sprungmarke ist als Sprungziel angegeben, es fehlt aber die zugeordnete Marke.
5504	Es sind keine Sprünge in untergeordnete Kontrollstrukturen (z. B. WHILE-Schleifen) möglich. Die angegebene Sprungmarke kann an dieser Stelle nicht verwendet werden.
5505	Es sind keine Sprünge in untergeordnete Kontrollstrukturen (z. B. WHILE-Schleifen) möglich. Das angegebene Sprungziel kann nicht erreicht werden.
5506	Es sind keine Sprünge in WAITFORCONDITION-Blöcke möglich. Die angegebene Sprungmarke kann an dieser Stelle nicht verwendet werden.
5507	Es sind keine Sprünge in WAITFORCONDITION-Blöcke möglich. Das angegebene Sprungziel kann nicht erreicht werden.
5509	Innerhalb einer CASE-Anweisung ist die Angabe von Sprungmarken nicht zulässig. Sie Syntax ermöglicht keine Unterscheidung zwischen einer Sprungmarke und der Wertliste der CASE-Anweisung.

A.2.6 Fehler im Ausdruck (6001 … 6301)

Tabelle A-14 Fehler im Ausdruck (6001 … 6301)

Fehler	Beschreibung
6001	Syntaxfehler: Es wird eine Anweisung erwartet, die mit einem Semikolon abgeschlossen ist, z. B. a := b*c;
6002	Syntaxfehler: Es wird ein Ausdruck erwartet, z. B. x < y.
6003	Der angegebene Bezeichner ist kein Variablenbezeichner. Sie müssen einen Variablenbezeichner angeben. Prüfen Sie, ob der angegebene Bezeichner verdeckt ist. Bis einschließlich V4.0 war der Zugriff auf geräteglobale Bezeichner innerhalb eines gleichnamigen Programms oder Funktionsbausteins trotz Warnung 16021 möglich.
6004	Der Index bei Feldzugriffen muss vom Datentyp DINT sein. Führen Sie eine entsprechende Typ-Konvertierung aus oder verwenden Sie einen anderen Ausdruck.
6005	Im Ausdruck liegt ein Typkonflikt vor. Einer der Operanden lässt sich nicht in den Datentyp der Berechnung konvertieren, oder die Ergebniszuweisung liefert einen Typkonflikt.

Fehler	Beschreibung
6006	Auf die angegebene Variable kann nicht zugegriffen werden. Eine Verwendung im Ausdruck ist nicht möglich. Mögliche Ursachen sind: • Variable ist nicht lesbar. • Versuch, auf eine lokale Variable einer Funktion oder eines Funktionsbausteins von außerhalb zuzugreifen.
6007	Die angegebene Variable kann nicht geschrieben werden. Eine Wertzuweisung ist nicht möglich.
6008	Die angegebene Funktion liefert keinen Rückgabewert. Eine Anwendung im Ausdruck ist daher nicht möglich (Funktion mit Rückgabewert VOID deklariert).
6009	Der angegebene Bezeichner verweist nicht auf eine Funktion oder auf eine Instanz eines Funktionsbausteins. Eine Verwendung als Funktionsbezeichner ist daher nicht möglich. Beim Aufruf eines Programms muss die Compileroption "Sprachweiterungen zulassen" gesetzt sein (-C lang_ext).
6010	Der angegebene Bezeichner ist in der Deklaration der POE (Funktion oder Funktionsbaustein) nicht als Eingangsparameter (VAR_INPUT) oder Durchgangsparameter (VAR_IN_OUT) enthalten. Eine Verwendung beim Aufruf der POE ist nicht möglich.
6011	Die Anzahl der Funktionsargumente im Aufruf unterscheidet sich von der Deklaration, oder es sind notwendige anzugebende Aufrufparameter im Aufruf nicht enthalten.
6012	RETURN ist an dieser Stelle syntaktisch nicht zulässig. RETURN darf nur in Funktionen verwendet werden.
6013	EXIT ist an dieser Stelle syntaktisch nicht zulässig. EXIT darf nur innerhalb von FOR, WHILE, REPEAT verwendet werden.
6014	Der angegebene Indexwert liegt außerhalb der Arraygrenzen. Es sind nur Indexwerte entsprechend der Arraydeklaration zulässig.
6015	Der angegebene Tasksteuerbefehl kann auf die Task nicht angewendet werden. Er ist für den Typ der Task nicht zulässig.
6016	Die angegebene Task ist im Ablaufsystem deaktiviert. Um sie verwenden zu können, muss sie aktiviert werden.
6017	Syntaxfehler bei der Angabe der Programme innerhalb einer Task. Die Programme sind per Name durch Komma getrennt aufzuführen.
6018	Der angegebene Bezeichner verweist nicht auf ein PROGRAM. Eine Verwendung als Programmbezeichner ist daher nicht möglich.
6019	Ein Programm wurde einer Task mehrfach zugeordnet. Es ist nur eine einfache Zuordnung möglich.
6020	Syntaxfehler bei der Angabe von direkt dargestellten Variablen. Inputs müssen die Syntax %Ix.y und Outputs die Syntax %Qx.y haben.
6021	Der angegebene Byteoffset der direkt dargestellten Variablen liegt außerhalb des zulässigen Adressraums.
6022	Der angegebene Bitoffset der direkt dargestellten Variablen liegt außerhalb des zulässigen Adressraums. Zulässig sind die Werte 0 bis 7.
6023	Der Returnwert der Funktion wurde nicht zugewiesen. Eine Zuweisung ist jedoch zwingend erforderlich.
6024	Eine Variable mit dem angegebenen Bezeichner ist nicht in der Taskstartinformation enthalten.
6025	Die Bedingungsvariable und die Bedingungswerte einer CASE-Anweisung müssen vom Datentyp SINT, INT, DINT, USINT, UINT oder UDINT sein. Die Bedingungswerte müssen implizit in den Datentyp der Bedingungsvariablen konvertiert werden können.
6026	Der angegebene Meldungsbezeichner ist nicht in der Meldungsprojektierung enthalten. Wechseln Sie in die Meldungsprojektierung und ergänzen Sie den Bezeichner.

Fehler	Beschreibung
6027	Der Systemvariablenzugriff ist nur direkt über eine TO-Referenz möglich. Der Zugriff über eine Struktur bzw. ein Feld kann nicht erfolgen. Legen Sie eine lokale Variable vom TO-Typ an und weisen Sie dieser Variablen die TO-Referenz zu. Sie können dann über diese lokale TO-Variable auf die gewünschte Systemvariable zugreifen.
6028	Es liegt ein Typkonflikt im Ausdruck an der angegebenen Operation vor. Einer der Operanden lässt sich nicht in den Datentyp der Berechnung konvertieren, oder die Ergebniszuweisung liefert einen Typkonflikt. Es wird der angegebene Datentyp im Ausdruck erwartet.
6029	Der angegebene Funktionsparameter hat keinen Defaultwert und ist deshalb zwingend beim Aufruf anzugeben.
6030	Es wurde versucht, einen Ausdruck an einen Durchgangsparameter (VAR_IN_OUT) zu übergeben. Das ist nicht möglich. Als Durchgangsparameter müssen Anwendervariablen angegeben werden.
6031	Es wurde versucht, eine Systemvariable (TO, I/O-Direktzugriff) an einen Durchgangsparameter (VAR_IN_OUT) zu übergeben. Das ist nicht möglich. Als Durchgangsparameter müssen Anwendervariablen angegeben werden.
6032	Es wurde versucht, eine Variable im Prozessabbild an einen Durchgangsparameter (VAR_IN_OUT) zu übergeben. Das ist nicht möglich. Als Durchgangsparameter müssen Anwendervariablen angegeben werden.
6033	Es wurde versucht, eine Variable an einen Durchgangsparameter (VAR_IN_OUT) zu übergeben, die nicht im Datentyp übereinstimmt. Eine implizite Typkonvertierung ist jedoch nicht möglich. Als Durchgangsparameter müssen Anwendervariablen mit korrektem Datentyp angegeben werden.
6034	Es wurde versucht, eine nicht schreibbare Variable an einen Durchgangsparameter (VAR_IN_OUT) zu übergeben. Das ist nicht möglich, Durchgangsparameter müssen schreibbar sein.
6035	Es wurde versucht, eine Konstante an einen Durchgangsparameter (VAR_IN_OUT) zu übergeben. Das ist nicht möglich, Durchgangsparameter müssen Anwendervariablen sein.
6036	Es wird eine Operation auf eine Konstante angewendet. Dabei liegt der Wert der Konstanten außerhalb des Definitionsbereichs der Funktion. Beispiele hierfür sind: • Anwendung von SQRT auf eine negative Zahl. • Anwendung der Logarithmenfunktionen auf eine Zahl <= 0. • Anwendung von ASIN oder ACOS auf eine Zahl außerhalb des Intervalls [0..1].
6037	Es wurde versucht, eine Division durch Null mit einer Konstanten durchzuführen. Diese Operation ist unzulässig.
6038	Der angegebene Funktionsparameter ist mehrfach in der Argumentliste enthalten.
6039	Die angegebene POE (Funktion oder Funktionsbaustein) ist nicht anwendbar. Mögliche Ursachen: • Die Definition der POE im Implementationsabschnitt fehlt. Es wurde nur der Prototyp im Interfaceabschnitt angegeben. • Die POE wird erst nach ihrer Verwendung (z. B. Aufruf, Instanzdeklaration) vollständig definiert. Verschieben Sie gegebenenfalls in der Programmquelle diese POE vor die POE, in der sie verwendet wird. • Eine Instanz eines Funktionsbausteins kann als Unit-Variable nicht in derselben Programmquelle deklariert werden, in der dieser Funktionsbaustein definiert wird.
6040	Als Semaphore sind nur einfache Variablen anwendbar, eine Indizierung ist nicht möglich.
6041	Die Meldungsfunktion erfordert einen Begleitwert vom angegebenen Datentyp. Eine Typumwandlung ist nicht möglich.
6042	Die Meldungsfunktion erfordert die Angabe einer Meldungsnummer. Die angegebene Meldungsnummer ist ungültig.

Fehler	Beschreibung
6050	Im Ausdruck liegt ein Typkonflikt bei der angegeben Operation / Variablen vor. Einer der Operanden lässt sich nicht in den Typ der Berechnung konvertieren oder die Ergebniszuweisung liefert einen Typkonflikt. Es ist keine Konvertierung zwischen dem Quelltyp und dem Zieltyp möglich.
6051	Im Ausdruck liegt ein Typkonflikt bei der angegebenen Operation vor. Einer der Operanden lässt sich nicht in den Datentyp des anderen Operanden konvertieren, um die Berechnung auszuführen, oder die Datentypen der Operanden sind für die Operation nicht zulässig.
6052	Im Ausdruck liegt ein Typkonflikt vor. Der angegebene Datentyp kann bei der Operation nicht verwendet werden (siehe Marshalling Funktionen).
6053	Im Ausdruck liegt ein Typkonflikt bei der angegebenen Operation vor. Die Operation ist auf dem angegebenen Datentyp nicht zulässig.
6054	Im Ausdruck liegt ein Typkonflikt vor. Die angegebene Variable kann nicht als indizierte Feldvariable verwendet werden.
6060	Beim Funktionsaufruf liegt eine Mischung aus Zuweisungen von Funktionsargumenten und Stellungsparametern vor. Verwenden Sie eine Form des Funktionsaufrufs. Beispiel: • f (x, y); oder • f (in1 := x, in2 := y);
6061	Der angegebene Parameter der Funktion bzw. des Funktionsbausteins ist ein Durchgangsparameter. Beim Aufruf der POE muss ihm deshalb zwingend eine Variable zugewiesen werden.
6062	Der angegebene Bezeichner kann nicht als Funktionsargument verwendet werden. Als Funktionsargument sind nur Variablen aus den Deklarationsblöcken VAR_INPUT und VAR_IN_OUT zulässig.
6063	Der angegebene Bezeichner kann als Funktionsargument nicht verwendet werden. Als Funktionsargument sind nur Variablen aus den Deklarationsblöcken VAR_INPUT und VAR_IN_OUT zulässig.
6064	Die angegebene POE ist ein Prototyp, für die es keine Implementierung gibt. Daher ist kein Aufruf möglich.
6070	Der Zugriff auf Konfigurationsdaten ist nur bei vollständig angegebenen Variablen möglich. Ergänzen Sie den Namen entsprechend der Konfigurationsdaten des angewählten Technologieobjekts.
6071	Der Zugriff auf Konfigurationsdaten ist nur bei vollständig angegebenen Variablen möglich. Dabei dürfen keine Array-Indizes verwendet werden, die erst zur Laufzeit auflösbar sind.
6080	Die angegebene Variable ist keine direkt zugreifbare Eingangs- oder Ausgangsvariable. Eine solche Variable muss am I/O-Container des betroffenen Geräts mit der Syntax PI* oder PQ* vereinbart werden.
6100	Das angegebene Konstrukt kann nur mit gesetztem Gerätetyp übersetzt werden. Ergänzen Sie die UNIT-Anweisung mit dem Gerätetyp oder setzen Sie den Gerätetyp am Programmcontainer.
6110	Das angegebene Konstrukt kann in Bibliotheken nicht verwendet werden.
6111	Das angegebene Konstrukt kann in Bibliotheken nicht verwendet werden.
6112	Das angegebene Konstrukt kann in Bibliotheken nicht verwendet werden.
6113	Der Zugriff auf Technologieobjekte und Geräte ist in Bibliotheken nicht zulässig.
6114	Es wurde ein Funktionsbaustein aus einer anderen Quelle derselben Bibliothek verwendet. Dies ist innerhalb einer Bibliothek nur möglich, wenn die deklarierende Quelle mit der Compilereinstellung "Vorwärtsdeklarationen erlauben" übersetzt wird.
6130	Die Angabe eines Intervalls ist für den genannten Datentyp in der CASE-Anweisung nicht zulässig.
6140	Die Angabe einer Konstanten bei ENUM_TO_DINT erfordert den Vorsatz des Datentyps in der Form enum_type#wert.
6150	Der angegebene Bitoffset liegt bei dem gegebenen Datentyp außerhalb des gültigen Bereichs.
6160	Der angegebene Arraydatentyp ohne Längenangabe ist nur bei der Deklaration von VAR_IN_OUT Parametern in Funktionen und Funktionsbausteinen zulässig.

Fehler	Beschreibung
6161	Die direkte Zuweisung zwischen Arrays ohne Längenangabe ist nicht möglich. Zur Zuweisung müssen Sie über die Elemente iterieren.
6170	Der angegebene Bezeichner ist kein Variablenbezeichner. Sie müssen einen Variablenbezeichner angeben. Dabei ist zu beachten, dass lokale Variablen einer POE in einer eigenständigen EXPRESSION (Bedingung) nicht nutzbar sind.
6180	Die angegebene Funktion ist für Variablen, die Strukturen mit überlappenden Speicherbereichen enthalten (OVERLAP), nicht einsetzbar.
6200	Nur bei Compileroption "Spracherweiterungen zulassen" (-C lang_ext): Das aufgerufene PROGRAM enthält Instanzdaten (Deklarationsblock VAR … END_VAR), die im Anwenderspeicher der zugeordneten Task abgelegt werden. Damit ist ein Aufruf des PROGRAM aus einer anderen POE heraus nicht möglich. Übersetzen Sie die Quelle mit der Compileroption "Programminstanzdaten nur einmal anlegen" (-C prog_once) oder entfernen sie die Instanzdaten.
6201	Nur bei Compileroption "Spracherweiterungen zulassen" (-C lang_ext): Der Aufruf eines PROGRAM wird in Funktionen nicht unterstützt. Solche Aufrufe können nur in Funktionsbausteinen oder anderen PROGRAM vorgenommen werden.
6300	Nur bei Angabe von Pragma "ToHookApplicable = YES": Der Aufruf der angegebenen POE ist nicht möglich, da diese POE nicht als ToHookApplicable gekennzeichnet ist. In Hook-Funktionen können nur dafür geeignete andere Funktionen verwendet werden.
6301	Nur bei Angabe von Pragma "ToHookApplicable = YES": Der Zugriff auf die angegebene Variable ist bei der Übersetzung zur Verwendung in TO-Hooks nicht möglich. In TO-Hooks kann auf folgende Variablen nicht zugegriffen werden: • Variablen von Technologieobjekten • I/O-Variablen • Geräteglobale Variablen

A.2.7 Syntaxfehler, Fehler im Ausdruck (7000 … 7014)

Tabelle A-15 Syntaxfehler, Fehler im Ausdruck (7000 … 7014)

Fehler	Beschreibung
7000	Ein Syntaxfehler ist aufgetreten. Mögliche Ursachen sind: • nicht korrekt abgeschlossene Kontrollstrukturen (z. B.END_IF vergessen), • nicht mit ; abgeschlossene Anweisungen, • fehlende Klammern.
7001	Der angegebene Bezeichner bezeichnet keine Konstante. Bitte geben Sie eine Konstante per Wert oder Bezeichner an.
7002	Es wird eine vorzeichenbehaftete Ganzzahl erwartet. Diese kann vom Datentyp SINT, INT oder DINT sein.
7003	Bei der Intervallangabe muss der Anfangswert kleiner oder gleich dem Endwert sein. Dies gilt bei der Vereinbarung von Arrays und bei der Intervallangabe in CASE-Auswahlbedingungen.

Fehler	Beschreibung
7004	Es wird ein Initialisierungswert erwartet. Dies muss eine Konstante sein. Konstanten können vorgegeben werden: • direkt per Wert, • symbolisch über eine vorhergehende Konstantendeklaration • als Ausdruck, der nur Konstanten enthält.
7005	Die Initialisierung von identischen Datentypen in unterschiedlichen Quellen erfordert auch einen identischen Initialisierungswert. Passen sie die Initialisierungswerte an.
7009	Als Bedingung für WHILE, REPEAT, IF wird ein Ausdruck erwartet, der den Datentyp BOOL liefert. Dies kann durch Angabe einer Variablen vom Datentyp BOOL oder über einen Vergleichsausdruck erreicht werden. Die Angabe einer Funktion mit Rückgabewert vom Datentyp BOOL ist ebenfalls zulässig.
7010	Ein Syntax-Fehler ist aufgetreten. Mögliche Ursachen sind: • nicht korrekt abgeschlossene Kontrollstrukturen (z. B. END_IF vergessen), • nicht mit ; abgeschlossene Anweisungen, • fehlende Klammern.
7011	Ein Syntax-Fehler ist aufgetreten. Mögliche Ursachen sind: • nicht korrekt abgeschlossene Kontrollstrukturen (z. B. END_IF vergessen), • nicht mit ; abgeschlossene Anweisungen, • fehlende Klammern.
7012	Ein Syntax-Fehler in der Anweisung, die auf der angegebenen Zeile beginnt, ist aufgetreten. Mögliche Ursachen sind: • nicht korrekt abgeschlossene Kontrollstrukturen (z. B. END_IF vergessen), • nicht mit ; abgeschlossene Anweisungen, • fehlende Klammern.
7013	Ein Syntax-Fehler ist aufgetreten. Es wird ein nicht zulässiges Konstrukt verwendet.
7014	Ein Syntax-Fehler ist aufgetreten. Mögliche Ursachen sind: • nicht korrekt abgeschlossene Kontrollstrukturen (z. B. END_IF vergessen), • nicht mit ; abgeschlossene Anweisungen, • fehlende Klammern.

A.2.8 Fehler beim Binden einer Quelle (8001, 8100)

Tabelle A-16 Fehler beim Binden einer Quelle (8001, 8100)

Fehler	Beschreibung
8001	Die angegebene POE ist im INTERFACE-Abschnitt exportiert worden, eine Implementierung im IMPLEMENTATION-Abschnitt fehlt jedoch. Löschen Sie entweder die Export-Anweisung oder geben Sie eine gültige Implementierung an.
8100	Die maximale Größe des Datenbereichs, der über HMI erreichbar, ist beträgt 65536 Byte. Diese Grenze wird bei der angegeben Variablen überschritten. Alle nachfolgenden Variablen sind ebenfalls nicht erreichbar.

A.2.9 Fehler beim Laden des Interfaces einer anderen UNIT oder eines Technologiepakets (10000 … 10101)

Tabelle A-17 Fehler beim Laden des Interfaces einer anderen UNIT oder eines Technologiepakets (10000 … 10101)

Fehler	Beschreibung
10000	Die angegebene Unit hat ein ungültiges Dateiformat. Wahrscheinlich wurde sie mit eine älteren Version des Compilers erstellt oder mit inkompatiblen Optionen übersetzt. Handelt es sich um eine Unit, sollten Sie diese zunächst übersetzen und danach die aktuelle Übersetzung wiederholen. Handelt es sich um ein Paket, sollten Sie eine neuere Version installieren.
10001	Der Unit-Name hat ein ungültiges Format. Für Unit-Namen gelten die Regeln für Bezeichner in ST, für ihre Länge gilt: • Bis Version V4.0 des SIMOTION Kernels: 8 Zeichen. • Ab Version V4.1 des SIMOTION Kernels: 128 Zeichen.
10002	Fehler beim Laden des Interfaces einer anderen Unit, einer Bibliothek oder eines Technologiepakets. Der angegebene Bezeichner ist in zwei unterschiedlichen importierten Units, Bibliotheken oder Technologiepaketen enthalten. • Entfernen Sie eine Unit, Bibliothek oder Technologiepaket aus der Importliste oder • Stellen Sie eine Eindeutigkeit zwischen den Bezeichnern in importierten Units, Bibliotheken und Technologiepaketen her. Überarbeiten Sie dazu die exportierenden Units im Interfaceabschnitt oder geben Sie für eine Bibliothek bzw. ein Technologiepaket einen Namespace an (USELIB … AS namespace; USEPACKAGE … AS namespace;).
10003	Der angegebene Datentyp hat ein ungültiges Speicherlayout. Wahrscheinlich wurde er mit eine älteren Version des Compilers erstellt oder mit inkompatiblen Optionen übersetzt. Handelt es sich um eine Unit, sollten sie diese zunächst übersetzen und danach die aktuelle Übersetzung wiederholen. Sie können auch "Speichern und alles neu übersetzen" ausführen. Handelt es sich um ein Paket, sollten Sie eine neuere Version installieren. Tritt der Fehler danach immer noch auf, informieren Sie den Support.
10004	Die exportierten Bezeichner der angegebenen Unit konnten nicht geladen werden. Schließen Sie einige Anwendungen und versuchen Sie es erneut.
10005	Es liegt eine Rekursion beim Laden der Pakete vor. Das angegebene Paket wurde bereits über USEPACKAGE geladen und kann kein zweites Mal angegeben werden.
10006	Es liegt eine Rekursion beim Laden der Units vor. Die angegebene Unit wurde bereits über USES geladen und kann kein zweites Mal angegeben werden.
10007	Die Anzahl der in einer Unit referenzierbaren importierten Units wurde überschritten. Es sind je Ladeeinheit maximal 223 importierte Units zulässig. Es zählen sowohl direkt mit USES importierte als auch indirekt durch diese Units importierte Units.
10008	Die Anzahl der in einer Unit referenzierbaren importierten Pakete wurde überschritten. Es sind je Ladeeinheit maximal 127 importierte Pakete zulässig.
10009	Das angegebene Paket wird in der Unit verwendet, ist jedoch nicht am Gerät verfügbar. Diese Fehlermeldung tritt auf, wenn Sie mit der Option implizite Paketverwendung übersetzen und eine USEPACKAGE-Anweisung programmiert haben, deren Inhalt sich von den auf am Gerät angegebenen Paketen unterscheidet.
10010	Das angegebene Paket wird in der Unit a verwendet, in der Unit b jedoch nicht. Diese Fehlermeldung tritt auf, wenn Sie in Units, die sich gegenseitig mit USES referenzieren, unterschiedliche Pakete bei USEPACKAGE angegeben haben. Korrigieren Sie die USEPACKAGE-Anweisungen.
10011	Die angegebene Unit wird direkt oder indirekt über ein oder mehrere andere Units durch sich selbst benutzt. Korrigieren sie die USES-Anweisungen.

Fehler	Beschreibung
10012	Die angegebene Unit wird direkt oder indirekt in mehreren Units in unterschiedlichen Übersetzungsständen importiert. Übersetzen Sie alle Units, welche die angegebene Unit in der USES-Anweisung referenzieren, erneut.
10013	Die angegebene Unit wurde noch nicht übersetzt, oder bei der letzten Übersetzung ist ein Fehler aufgetreten. Zum erfolgreichen Compilieren ist zunächst diese Unit zu übersetzen.
10014	Der Typ des angegebenen Technologieobjekts (TO) wird durch die bisher bei der Übersetzung mit USEPACKAGE angegebenen Pakete nicht unterstützt. Verwenden Sie ein Paket, das den TO-Typ enthält.
10015	Die Anzahl der in einer Unit referenzierbaren Technologieobjekte (TO) wurde überschritten. Es können maximal 65535 TO referenziert werden.
10016	Die Angabe des Gerätetyps ist nicht verfügbar. Sollte die zu übersetzende Unit keinem Gerät zugeordnet sein, verwenden Sie die Anweisung UNIT xx : dvtype;
10017	Die Angabe des Gerätetyps ist nicht eindeutig. In der Unit wird mit der Anweisung UNIT xx : dvtype; ein anderer Gerätetyp vorgegeben als der über die Zuordnung der Unit zum Gerät ermittelte Gerätetyp.
10018	Die angegebene Unit konnte nicht gefunden werden. Überprüfen Sie, ob der Unitname im Container PROGRAM der Workbench verfügbar ist oder ob die angegebene Datei im aktuellen Arbeitsverzeichnis (nur u7bt00ax-Kommandozeile) enthalten ist.
10019	Das angegebene Technologiepaket konnte nicht gefunden werden. Beachten Sie die vorhergehenden Fehlerausgaben.
10020	Beim Laden des Technologiepakets ist ein Fehler aufgetreten. Beachten Sie weitere Fehlerausgaben.
10021	Das Technologiepaket wird in der angegebenen Quelle verwendet, es ist jedoch auf dem Gerät nicht ausgewählt. Korrigieren Sie die USEPACKAGE-Anweisung oder wählen Sie das Technologiepaket am Gerät aus.
10022	Das angegebene Technologiepaket wird mit unterschiedlichen Versionsständen benutzt. Korrigieren Sie die Einstellungen zur Auswahl des Technologiepakets am Gerät und ggf. an der Bibliothek. Sie können nur einen Versionsstand eines Technologiepakets auf einem Gerät verwenden.
10024	Das angegebene Technologiepaket enthält keine in der Programmierumgebung verwendbaren Bestandteile. Es kann daher auch nicht in der USEPACKAGE-Anweisung verwendet werden.
10025	Der angegebene Bezeichner ist kein Bezeichner eines gültigen oder installierten Technologiepakets. Es kann daher auch nicht in der USEPACKAGE-Anweisung verwendet werden.
10026	Das Technologiepaket wird in der angegebenen Quelle verwendet, es ist jedoch für den aktuellen Gerätetyp an der Bibliothek nicht ausgewählt. Korrrigieren Sie die USEPACKAGE Anweisung in der Quelle oder wählen Sie das Technologiepaket für den Gerätetypen aus.
10030	Die Angabe des Gerätetyps ist nicht eindeutig. In der Unit wird mit der Anweisung UNIT xx : dvtype; ein anderer Gerätetyp vorgegeben, als der über die Zuordnung der Unit zum Bibliothekscontainer ermittelte Gerätetyp.
10031	Die angegebene Bibliothek wird direkt oder indirekt über ein oder mehrere andere Bibliotheken durch sich selbst benutzt. Korrigieren sie die USELIB-Anweisungen.
10032	Die angegebene Bibliothek konnte nicht gefunden werden. Überprüfen Sie Ihr Projekt.
10033	Es liegt eine Rekursion beim Laden der Bibliothek vor. Die angegebene Bibliothek wurde bereits über USELIB geladen und kann kein zweites Mal angegeben werden.

Fehler	Beschreibung
10034	Die angegebene Bibliothek Ist nicht vollständig übersetzt. Mögliche Ursachen: • Die Bibliothek wurde noch nicht übersetzt • Die Bibliothek wurde nicht für alle am Bibliothekscontainer angegebenen Gerätetypen übersetzt (z. B. beim projektweiten Übersetzen). • Bei der letzten Übersetzung ist ein Fehler aufgetreten. Übersetzen Sie zunächst diese Bibliothek einzeln (Übernehmen und übersetzen).
10035	Die angegebene Bibliothek konnte nicht gefunden werden. Überprüfen Sie, ob der Bibliotheksname im Projekt der Workbench verfügbar ist oder ob die angegebene Datei im aktuellen Arbeitsverzeichnis (nur u7bt00ax-Kommandozeile) enthalten ist.
10036	Das angegebene Paket wird in der Quelle verwendet, ist jedoch nicht an der Bibliothek verfügbar. Bibliotheken werden grundsätzlich gegen die am Bibliothekscontainer angegebenen Paketversionen übersetzt. Sie haben eine USEPACKAGE-Anweisung programmiert, deren Inhalt sich von der an der Bibliothek angegebenen Paketen unterscheidet. Wählen Sie entweder die korrekten Paketversionen aus oder entfernen Sie die USEPACKAGE-Anweisung aus der Quelle.
10037	An der angegebenen Bibliothek ist die Codevariante für den aktuellen Gerätetyp nicht angewählt. Damit kann diese Bibliothek nicht verwendet werden. Aktivieren Sie die Codevariante an dieser Bibliothek.
10038	Eine DCC-Bibliothek kann nur im DCC verwendet werden. Die Einbindung einer solchen Bibliothek in andere Programmiersprachen ist nicht zulässig.
10039	Die Übersetzung einer Bibliothek erfordert den Zugriff auf die Quellen. Der Zugriff auf die angegebene Quelle ist auf Grund der gewählten Schutzstufe des Know-how-Schutzes nicht möglich. Zur Übersetzung müssen sie sich einloggen.
10100	Der angegebene Typ eines Technologieobjekts ist in mehreren Paketen enthalten, die durch die Quelle referenziert werden. Entscheiden Sie sich für ein Technologiepaket, das Ihren Anforderungen entspricht.
10101	Das angegebene TO ist nicht kompatibel mit den durch die geladenen Pakete unterstützten Typen der Technologieobjekte. Machen sie ein Update des Pakets oder passen Sie den Typ des Technologieobjekts an.

A.2.10 Implementationsbeschränkungen (15001 … 15700)

Tabelle A-18 Implementationsbeschränkungen (15001 … 15700)

Fehler	Beschreibung
15001	Das angegebene Konstrukt wird von der aktuellen Version des Compilers nicht unterstützt.
15002	Das aktuell gewählte Gerät unterstützt die angegebene Funktion nicht. Wenn Sie die Funktion nutzen möchten, müssen Sie eine andere Geräteversion auswählen. Führen Sie dazu einen CPU-Tausch im Hardwarekatalog und ggf. ein Firmware-Update durch.
15003	Der angegebene Bezeichner ist ein nicht unterstütztes Schlüsselwort und daher nicht anwenderspezifisch verwendbar, um die Kompatibilität zu späteren Compilerversionen zu sichern.
15004	Der angegebene Bezeichner identifiziert eine nicht unterstützte Standardfunktion und ist nicht als anwenderspezifischer Bezeichner verwendbar, um die Kompatibilität zu späteren Compilerversionen zu sichern.
15005	Der angegebene Bezeichner identifiziert einen nicht unterstützten Standardfunktionsbaustein und ist nicht als anwenderspezifischer Bezeichner verwendbar, um die Kompatibilität zu späteren Compilerversionen zu sichern.

Fehler	Beschreibung
15006	Das angegebene Konstrukt kann nur in MCC generierten Quellen verwendet werden. Eine Benutzung im ST ist nicht möglich.
15007	Es wurde eine Quelle/Bibliothek/Paket im Implementationsabschnitt entweder direkt oder indirekt ohne Namespaceangabe verwendet. Im Interfaceabschnitt erfolgt die Verwendung mit Namespace. Lösen Sie diesen Konflikt durch Angabe eines Namespaces im Interfaceabschnitt für die genannte Quelle/Bibliothek/Paket.
15008	Die angegebene Quelle nutzt die Bibliothek oder das Technologiepaket mit unterschiedlicher Namespaceangabe zwischen Interface-und Implementationsabschnitt. Zur erfolgreichen Übersetzung muß die Bibliothek oder das Technologiepaket bereits im Interfaceabschnittl mit USELIB oder USEPACKAGE referenziert werden.
15070	Das angegebene Konstrukt ist nicht konform zum Sprachstandard, wird jedoch aus Kompatibilitätsgründen für alte Plattformen unterstützt. Stellen Sie die Verwendungsstelle auf die angegebene Alternative um.
15152	Es wurde eine USES-, USELIB- oder USEPACKAGE- Anweisung in einem über bedingte Compilierung versteckten Quelldateiabschnitt gefunden. Dies ist nicht zulässig. Quelldateiabschnitte, die diese Statements enthalten, können nicht bedingt übersetzt werden.
15153	Die angegebene Definition wird bei der Codegenerierung nicht berücksichtigt. Es ist nicht möglich, Schlüsselwörter anders zu definieren.
15154	Die Anwendung eines Zeilenkommentars und die Verwendung mehrzeiliger Kommentare ist im Definitionsteil nicht zulässig.
15200	Die Angabe eines Bitoffsets an einer Bitstring-Variablen ist nur unter Nutzung der Compileroption "Spracherweiterungen zulassen" (-C lang_ext) möglich.
15700	Das angegebene Konstrukt wird von der Version des SIMOTION SCOUT, in die konvertiert werden soll, nicht unterstützt.

A.2.11 Warnungen (16001 … 16700)

Die Ausgabe von Warnungen und Informationen können Sie steuern:

- In den globalen Einstellungen des Compilers (Seite 64).
- In den lokalen Einstellungen des Compilers (Seite 67).
- In einer ST-Quelle durch Angabe des folgenden Attributs (Seite 312) innerhalb eines Pragmas (Seite 307): { _U7_PoeBld_CompilerOption := warning:*n*:on } bzw. { _U7_PoeBld_CompilerOption := warning:*n*:off }, wobei *n* die Warnungsklasse oder die Nummer der Warnung bzw. Information ist.

Zusätzlich können Sie einzelne Warnungen und Informationen als Fehler umdefinieren:

- In einer ST-Quelle durch Angabe des folgenden Attributs (Seite 312) innerhalb eines Pragmas (Seite 307): { _U7_PoeBld_CompilerOption := warning:*n*:err }, wobei *n* die Nummer der Warnung bzw. Information ist.

Tabelle A-19 Warnungen (16001 … 16700)

Fehler	Beschreibung
16001	(Warnungsklasse: 0) Nur in Verbindung mit Compileroption "Selektives Linken". Die angegebene Funktion, der Funktionsbaustein oder das Programm werden nicht exportiert und nicht in der eigenen Unit aufgerufen. Es wird kein Code erzeugt.
16002	(Warnungsklasse: 0) Nur in Verbindung mit Compileroption "Selektives Linken". Die angegebene Unit enthält kein exportiertes PROGRAM und keine Taskanbindung. Für die Unit wird kein Code erzeugt.
16003	(Warnungsklasse: 2) Die Operanden der Vergleichsoperation enthalten keine explizite Typfestlegung. Der Datentyp, in dem der Vergleich durchgeführt wird, ist in der Ausgabe der Warnmeldung sichtbar. Geben Sie den Datentyp der verwendeten Konstanten explizit mit <Typ>#<Wert> an.
16004	(Warnungsklasse: 2) Die angegebene Typkonvertierung kann auf Grund der geringeren Darstellungsbreite oder der geringeren Genauigkeit des Zieldatentyps eine Änderung des Variablenwertes zur Folge haben.
16005	(Warnungsklasse: 2) Bei der Typkonvertierung kann in Abhängigkeit des Variablenwertes ein Wechsel des Vorzeichens erfolgen.
16006	(Warnungsklasse: 2) Der angegebene Wert wird auf Grund der nicht ausreichenden Darstellungsbreite auf den nächsten darstellbaren Wert gerundet.
16007	(Warnungsklasse: 2) Bei der Typkonvertierung tritt ein Genauigkeitsverlust auf. Nicht alle Dezimalstellen werden berücksichtigt.
16008	(Warnungsklasse: 2) Bei der Initialisierung der angegebenen Variablen tritt ein Genauigkeitsverlust auf. Die Konstante wird in den angegebenen Datentyp konvertiert. Nicht alle Dezimalstellen werden berücksichtigt.
16009	(Warnungsklasse: 0) Nicht in Verbindung mit Compiler-Option selektives Binden. Die angegebene Unit enthält keine exportierten PROGRAM's und keine Taskanbindung. Der Code der Unit ist nicht erreichbar. Es ist kein Aufruf der enthaltenen POE möglich.
16010	(Warnungsklasse: 0) Das angegebene PROGRAM wird in der Unit nicht exportiert, daher ist eine Verwendung in der Ablaufebenenprojektierung nicht möglich.
16011	(Warnungsklasse: 0) Die Quelle enthält keine exportierten globalen Variablen. Es werden keine Daten ins Zielsystem geladen.
16012	(Warnungsklasse: 0) Der angegebene Quellen-Name wurde vom Container PROGAMME des angewählten Geräts übernommen. Die Bezeichnung der Quelle in der UNIT-Anweisung wurde ignoriert.
16013	(Warnungsklasse: 2) Der angegebene Datentyp ist durch die Marshalling-Funktion nicht portabel konvertierbar. Verwenden Sie nur SIMOTION Geräte im Zusammenhang mit diesem Datentyp oder nehmen Sie eine explizite Konvertierung des Datentyps vor.

Fehler	Beschreibung
16014	(Warnungsklasse: 2) Bei der angegebenen Operation wird eine Datentypkonvertierung zwischen vorzeichenbehaftet (signed) und nicht vorzeichenbehaftet (unsigned) vorgenommen. Da hier der Bitstring übernommen wird, kann das Ergebnis im Zahlenwert vom angegebenen Wert abweichen.
16015	(Warnungsklasse: 2) Bei der Zuweisung der Zeichenkettenkonstanten zu der Variablen wird nur ein Teil der Zeichenkettenkonstanten übernommen, da die Länge der Variablen zur Aufnahme aller Zeichen nicht ausreicht.
16016	(Warnungsklasse: 2) Die Operanden im Ausdruck enthalten keine explizite Typfestlegung. Der Datentyp der Operation wird über die Angabe der Werte bestimmt. Der resultierende Datentyp, in dem der Ausdruck berechnet wird, ist in der Ausgabe der Warnmeldung sichtbar. Zur Festlegung des Datentyps: • Geben Sie den Datentyp der verwendeten Konstanten explizit mit <Typ>#<Wert> an oder • Verwenden Sie eine explizite Datentypkonvertierung.
16017	(Warnungsklasse: 2) Die Operanden im Ausdruck enthalten nur Konstanten. Der Datentyp der Operation konnte über die Angabe des Datentyps (in der Form <Typ>#<Wert>) oder explizite Datentypkonvertierung bestimmt werden. Diese Ausgabe dient der Fehlersuche insbesondere bei der Verwendung symbolischer Konstanten, da hier der Datentyp der Operation meist nicht einfach nachverfolgt werden kann.
16018	(Warnungsklasse: 2) Der Datentyp der Vergleichsoperation wird über den Wert einer Konstanten festgelegt, díe einen größeren Wertebereich als die enthaltene Variable hat. Der Vergleich wird mit dem Datentyp der Konstanten ausgeführt.
16020	(Warnungsklasse: 1) Die Deklaration verdeckt den angegebenen Bezeichner, der in der eigenen oder in einer importierten Quelle global definiert ist. Zugriffe auf den globalen Bezeichner sind aus der POE heraus, in der dieser Bezeichner lokal vereinbart ist, nicht möglich.
16021	(Warnungsklasse: 1) Die Deklaration verdeckt den angegebenen Bezeichner, der am Gerät definiert ist. Zugriffe auf den geräteglobalen Bezeichner sind mit _device.<name> möglich.
16022	(Warnungsklasse: 1) Die Deklaration verdeckt den angegebenen Bezeichner, der im Projekt definiert ist (z. B. Technologieobjekt oder Gerät). Zugriffe auf den projektglobalen Bezeichner sind mit _project.<name> möglich.
16023	(Warnungsklasse: 1) Die Deklaration verdeckt den angegebenen Bezeichner für den Datentyp eines Technologieobjekts. Zugriffe auf den Datentypbezeichner sind nicht mehr möglich.
16024	(Warnungsklasse: 1) Die Deklaration verdeckt den Zugriff auf das Technologieobjekt am Gerät. Zugriffe auf dieses TO sind mit _to.<name> möglich.
16025	(Warnungsklasse: 1) Die Deklaration verdeckt die gleichnamige IEC-Standardfunktion. Zugriffe auf diese Funktion sind im aktuellen Kontext nicht mehr möglich.

Fehler	Beschreibung
16026	(Warnungsklasse: 1) Der angegebene Bezeichner ist von SIEMENS für potentielle Erweiterungen reserviert. Die Verwendung dieses Bezeichners kann in späteren Ständen zu Compilerfehlermeldungen führen. Wenn Sie dies vermeiden wollen, ändern Sie diesen Bezeichner.
16027	(Warnungsklasse: 1) Der angegebene Bezeichner, der zum Zugriff auf I/O-Qualities reserviert ist, ist bereits am Gerät vergeben. Der Zugriff auf die I/O-Qualities ist damit nicht möglich.
16030	(Warnungsklasse: 1) Es wurde eine Marke in einer CASE-Anweisung mehrfach angegeben. Es wird immer nur die erste Marke ausgewertet. Weitere Angaben haben keinen Effekt.
16102	(Warnungsklasse: 3) Die Option zur Ausgabe von Code für die Diagnosefunktion Status Programm wird ignoriert, da keine Debug-Info erzeugt wird. Die Ausgabe der Debug-Info ist über die Compileroptionen abgewählt worden.
16103	(Warnungsklasse: 3) Die Option zur Ausgabe von Code für die Diagnosefunktion Status Programm an der Bibliothek wird ignoriert. Der Code für Status Programm wird entsprechend der Option an den einzelnen Quellen gebildet.
16110	(Warnungsklasse: 3) Die angegebene Pragmaanweisung wird in der aktuellen Version nicht unterstützt.
16111	(Warnungsklasse: 3) Die angegebene Pragmaanweisung wird im verwendeten Kontext nicht unterstützt. Die Position des Pragmas im Quelltext ist nicht korrekt.
16112	(Warnungsklasse: 3) Es wurde ein gültiges Pragma verwendet. Allerdings ist der für das Pragma angegebene Wert nicht gültig. Verwenden Sie einen gültigen Wert.
16113	(Warnungsklasse: 3) Im Pragma liegt ein Syntaxfehler bei der Vereinbarung eines Attributs vor. Die gültige Syntax lautet: <Attributbezeichner> := <Attributwert>; .
16150	(Warnungsklasse: 7) Es wurde eine neue Definition für den angegebenen Bezeichner vorgegeben. Die bisherige Definition ist damit ungültig. Mit dieser Warnung kann die Arbeit des Präprozessors verfolgt werden.
16151	(Warnungsklasse: 7) Es wurde versucht, mittels #undef die Definition des angegebenen Bezeichners zu löschen. Der Bezeichner ist jedoch nicht definiert, oder die Definition ist bereits gelöscht. Mit dieser Warnung kann die Arbeit des Präprozessors verfolgt werden.
16152	(Warnungsklasse: 7) Die angegebene Definition wird bei der Codegenerierung nicht berücksichtigt. Ursache dafür kann sein, dass der Präprozessor für die übersetzte Quelle deaktiviert ist.
16153	(Warnungsklasse: 7) Der Präprozessor ist in der aktuellen Quelle nicht aktiv, obwohl Präprozesssor-Anweisungen verwendet werden. Aktivieren Sie den Präprozessor oder entfernen die Anweisungen.

Fehler	Beschreibung
16154	(Warnungsklasse: 10) Der Präprozessor kann nicht zur Steuerung der Inhalte von USEPACKAGE-, USELIB- und USES-Anweisungen eingesetzt werden. Die automatische Ermittlung der Quellabhängigkeiten bei den Funktionen "Projekt speichern und Änderungen übersetzen" bzw, "Projekt speichern und alles neu übersetzen" ist nicht mehr gewährleistet.
16170	(Warnungsklasse: 10) Die Definitionen aus Quellen, die mittels USES importiert wurden, werden bei der Codegenerierung nicht berücksichtigt.
16171	(Warnungsklasse: 10) Die Definition aus der angegebenen Quelle, die mittels USES importiert wurde, konnte nicht geladen werden. Übersetzten sie die angegebene Quelle vorher.
16200	(Warnungsklasse: 4) Die Anwendung eines Semaphores erfordert eine globale Variable, damit ein Zugriff darauf aus einer anderen Task möglich ist. Tasklokale Vorgänge müssen nicht über Semaphoren verriegelt werden.
16210	(Warnungsklasse: 4) Die Basis der Potenzfunktion (Standardfunktion EXPT bzw. Operators **) ist negativ. Die Operation ist zur Laufzeit nur unter folgenden Bedingungen ausführbar: 1. Die Anwendung erfolgt auf einem Gerät mit einer Version des SIMOTION Kernels ab V4.1. 2. Der Exponent ist ganzzahlig. Bei nicht ganzzahligem Exponenten oder bei Anwendung auf einem Gerät mit einer Version des SIMOTION Kernels bis V4.0 erfolgt die Auslösung der ExecutionFaultTask. Das Programm wird an dieser Stelle abgebrochen.
16220	(Warnungsklasse: 4) Die Bedingung einer IF-Anweisung, WHILE-Anweisung oder REPEAT-Anweisung ist ein konstanter Ausdruck.
16230	(Warnungsklasse: 4) Der Ausdruck bewirkt mit den gegebenen Werten keine Änderung am Ergebnis, Code wird optimiert erzeugt.
16240	(Warnungsklasse: 4) Der Ausdruck mit den gegebenen Werten überschreitet den Definitionsbereich der Operation. Das Ergebnis ist womöglich falsch.
16250	(Warnungsklasse: 4) Es erfolgt eine Modifikation der Laufvariablen einer FOR-Schleife innerhalb dieser Schleife. Diese Modifikation wird entweder nicht wirksam oder führt zu einem nicht erwarteten Ergebnis bei der Bearbeitung dieser Schleife. Wenn die Modifikation der Variablen innerhalb der Schleife notwendig ist, dann verwenden Sie hier anstelle von FOR besser WHILE oder REPEAT.
16300	(Warnungsklasse: 5) Der Begleitwert besitzt einen Datentyp, der nicht in den projektierten Datentyp der Meldung konvertiert werden kann.
16301	(Warnungsklasse: 5) Der angegebene Begleitwert wird bei der Ausgabe der Meldung nicht ausgewertet.
16302	(Warnungsklasse: 5) Der Datentyp des Begleitwertes kann nicht aus der Meldungsprojektierung ermittelt werden. Es wird der angegebene Datentyp eingesetzt.

Fehler	Beschreibung
16303	(Warnungsklasse: 5) Es wurde kein Begleitwert an der Funktion angegeben, obwohl die Meldungsprojektierung einen solchen Wert fordert. Es wurde ein Defaultwert des entsprechenden Datentyps ergänzt.
16304	(Warnungsklasse: 5) Es wurde ein Alarmbegleitwert über eine Konstante oder einen Konstantenausdruck angegeben. Der resultierende Datentyp des Alarmbegleitwerts ist in der Ausgabe der Warnmeldung sichtbar. Zur Festlegung des Datentyps: • Geben Sie den Datentyp der verwendeten Konstanten explizit mit <Typ>#<Wert> an oder • Verwenden Sie eine explizite Datentypkonvertierung.
16400	(Warnungsklasse: 6) Es wurde eine globale Variable in einer Bibliothek deklariert. Dies kann ggf. dazu führen, dass die Bibliothek nicht mehrfach einsetzbar ist.
16401	(Warnungsklasse: 6) Es wurde eine PROGRAM in einer Bibliothek deklariert. Dieses kann im Ablaufsystem nicht verwendet werden. Setzen sie die Compileroption"'Programminstanzdaten nur einmal anlegen" (-C prog_once) oder deklarieren sie einen FUNCTION_BLOCK
16420	(Warnungsklasse: 6) Es wurde innerhalb der Funktion der Rückgabewert nicht zugewiesen. Wird eine solche Funktion aufgerufen, so gibt sie einen zufälligen Wert zurück.
16421	(Warnungsklasse: 6) Es wurde eine Variable deklariert, die im Code weder zugewiesen noch gelesen wurde.
16430	(Warnungsklasse: 6) Es wird versucht, einen Funktionsbaustein, der entweder selbst ein Systemfunktionsbaustein ist einen solchen beinhaltet, als temporäre-Variable zu verwenden. Dies kann bei einer Nutzung des Bausteins in einer zyklischen Ebene zu Laufzeitproblemen führen.
16450	(Warnungsklasse: 10) Es wurde eine globale Variable im remanenten Speicherbereich angelegt. Diese Deklaration ist an der angegebenen Stelle nicht zulässig.
16451	(Warnungsklasse: 10) Die Initialisierung großer Felder mit Werten ungleich 0 verursacht ein hohes Datenaufkommen in der Steuerung. Dies führt sowohl zu langen Ladezeiten als auch zu einer hohen Speicherbelastung.
16452	(Warnungsklasse: 10) Das angegebene Programm hat eine große Menge zu initialisierender Instanzdaten. Dies kann beim Anlauf der Task zu einer Laufzeitüberschreitung führen, da hier sowohl Initialisierungs- als auch der Anwendercode ausgeführt werden. Insbesondere ist Vorsicht bei SynchronousTasks angebracht.
16470	(Warnungsklasse: 10) Das angegebene Konstrukt ist nicht konform zum Sprachstandard, wird jedoch aus Kompatibilitätsgründen für alte Plattformen unterstützt. Stellen Sie die Verwendungsstelle auf die angegebene Alternative um.
16600	(Warnungsklasse: 6) Die angegebene Variable ist in der Initialisierungsliste nicht enthalten. Es wird der Default-Initialisierungswert verwendet.
16601	(Warnungsklasse: 6) Die angegebene Variable ist in der Initialisierungsliste nicht enthalten. Es wird der Default-Initialisierungswert verwendet.

Fehler	Beschreibung
16602	(Warnungsklasse: 6) Die angegebene Variable ist in der Initialisierungsliste nicht enthalten. Es wird der Default-Initialisierungswert verwendet.
16603	(Warnungsklasse: 6) Der angegebene Funktionsbaustein besitzt keine Instanzdaten und hat damit eine Größe von 0 Byte. Bei der Referenzübergabe als dimensionsloses Array wird die Elementgröße auf 1 Element festgelegt.
16604	(Warnungsklasse: 6) Der in der Deklaration verwendete Datentyp hat keinen eindeutigen Initialisierungswert. Es wird der letzte bekannte Initialwert verwendet. Um an dieser Stelle ein eindeutiges Verhalten zu erzielen, muss der Initialwert für die Varaiblen- bzw Datentypdeklaration explizit angegeben werden.
16605	(Warnungsklasse: 6) Der Speicherbereich der angebenen Variablen überlappt sich mit einer anderen Variablen. Damit kann der Initialisierungswert nicht übernommen werden. Der verbleibende Speicherbereich wird mit 0 vorbelegt.
16606	(Warnungsklasse: 6) Der Speicherbereich der angebenen Variablen überlappt sich mit einer anderen Variablen. Damit kann der Datentyp nicht in die OPC-XMP-Info übernommen werden. Die Komponente ist damit symbolisch nicht erreichbar und auch in Datensicherungen mittels _exportUnitDataSet nicht enthalten.
16700	(Warnungsklasse: 3) Das SIMOTION Gerät kann auch mit früheren Versionen des SIMOTION SCOUT bearbeitet werden. Das angegebene Konstrukt wird nicht von allen früheren Versionen des Compilers unterstützt.

A.2.12 Informationen (32010 ... 32653)

Die Ausgabe der Informationen steuern Sie gemeinsam mit den Warnungen (Seite 437):

Tabelle A-20 Informationen (32010 … 32653)

Fehler	Beschreibung
32010	(Warnungsklasse: 6) Der angegebene Sprungmarkenbezeichner wurde deklariert, aber nicht verwendet.
32020	(Warnungsklasse: 10) Die angegebene Variable wurde in dieser Quelle oder in einer anderen Quelle global mit dem angegebenen Datentyp deklariert. Diese Information ist eine Hilfe bei der Suche nach der Ursache eines Fehlers beim Compilieren. Sie wird zusammen mit Fehlermeldungen ausgegeben.
32021	(Warnungsklasse: 10) Die angegebene Variable wurde am Gerät als I/O-Variable, als geräteglobale Variable oder als Systemvariable deklariert. Diese Information ist eine Hilfe bei der Suche nach der Ursache eines Fehlers beim Compilieren. Sie wird zusammen mit Fehlermeldungen ausgegeben.

Fehler	Beschreibung
32022	(Warnungsklasse: 10) Die angegebene Variable wurde am Projekt als globaler Bezeichner vereinbart. Diese Information ist eine Hilfe bei der Suche nach der Ursache eines Fehlers beim Compilieren. Sie wird zusammen mit Fehlermeldungen ausgegeben.
32023	(Warnungsklasse: 10) Es wurde bisher keine gültige Deklaration für den angegebenen Bezeichner gefunden. Diese Information wird zusammen mit Fehlermeldungen ausgegeben.
32024	(Warnungsklasse: 0) Die angegebene Variable wurde in der aktuellen oder in einer importierenden Unit als globaler Bezeichner vereinbart. Diese Information ist eine Hilfe bei der Suche nach der Ursache eines Fehlers beim Compilieren. Sie wird zusammen mit Fehlermeldungen ausgegeben.
32025	(Warnungsklasse: 10) Der angegebene Bezeichner wurde in dieser Quelle oder in einer anderen Quelle im angegebenen Namensraum deklariert. Um diesen Bezeichner zu verwenden, müssen sie den Bezeichner des Namensraumes voranstellen. Es kann sich hier um eine Variable, eine Konstante oder einen anderen Bezeichner handeln. Diese Information ist eine Hilfe bei der Suche nach einem Compilerfehler. Sie wird zusammen mit Fehlermeldungen ausgegeben.
32030	(Warnungsklasse: 0) Die angegebene Feldinitialisierung ist nicht konform zu IEC 61131-3. Für portable Programme sollten sie die Feldinitialisierungswerte in eckige Klammern einschließen. Beispiel für eine normkonforme Feldinitialisierung: x : ARRAY [0..1] OF INT := [1, 2];
32050	(Warnungsklasse: 0) Die maximale Größe, die über HMI erreichbar ist beträgt 65536 Byte. Diese Grenze wird bei der angegebenen Variablen überschritten. Alle nachfolgenden Variablen sind ebenfalls nicht erreichbar.
32300	(Warnungsklasse: 1) Es wurde eine Marke in einer CASE-Anweisung mehrfach angegeben. Es wird immer nur die erste Marke ausgewertet. Weitere Angaben haben keinen Effekt.
32650	(Warnungsklasse: 7) Der angegebene Bezeichner wird im Weiteren durch den ausgegebenen Text ersetzt. Mit dieser Information kann die Arbeit des Präprozessors verfolgt werden.
32651	(Warnungsklasse: 7) Die Definition des angegebenen Bezeichners wurde mit #undef gelöscht. Mit dieser Information kann die Arbeit des Präprozessors verfolgt werden.
32652	(Warnungsklasse: 7) Der Bezeichner wird mit dem angegebenen Ersetzungstext in der Quelle verwendet. Es erfolgt eine Übersetzung mit dem Ersetzungstext. Mit dieser Information kann die Arbeit des Präprozessors verfolgt werden.
32653	(Warnungsklasse: 7) Der angegebene Bezeichner wird im Weiteren durch den ausgegebenen Text ersetzt. Diese Information erscheint, wenn mit einer USES-Anweisung zusätzliche Ersetzungen geladen werden. Mit dieser Information kann die Arbeit des Präprozessors verfolgt werden.

A.3 Template für Beispiel-Unit

A.3.1 Vorabinformationen

In diesem Kapitel ist das ausführlich kommentierte Template abgedruckt, das Sie in der Online-Hilfe aufrufen können. Sie können es als Vorlage für eine neue ST-Quelle verwenden.

```
//============================================================================
//(organisation)
//(division / place)
//(c)Copyright 2009  All Rights Reserved
//----------------------------------------------------------------------------
//  project name: (name)
//  file name: (name as soon as saved)
//  library: (that the source is dedicated to)
//  system: (target system)
//  version: (SIMOTION / SCOUT version)
//  application: (relation to project/ product/ usage)
//  restrictions:
//  requirements: (hardeware, technological package, memory needed, etc.)
//  search items: (with the purpose of browser usage)
//  functionality: (that is implemented)
//----------------------------------------------------------------------------
//  change log table:
//  version    date      expert in charge     changes applied
//
//============================================================================

INTERFACE
    //  Alle zwischen INTERFACE und END_INTERFACE eingefügten Anweisungen/
    //  Schlüsselwörter dienen dazu festzulegen, welche Quellinhalte
    //  (Variablen, Funktionen, Funktiosbausteine etc.) auch in anderen
    //  Quellen (Units) zur Verfügung stehen bzw. exportiert werden.

    USEPACKAGE cam;
    // Hier werden zu verwendende Technologiepakete bekannt und dadurch
    // in der Quelle nutzbar gemacht. Technologieobjekt(TO)-spezifische
    // Befehle können nur in dieser UNIT genutzt werden, wenn das
    // entsprechende Paket eingebunden wurde.
    // Falls per USES eine Quelle eingebunden wird, die USEPACKAGE cam
    // verwendet, so wird dies "vererbt". USEPACKAGE kann dann entfallen.
    // Das im Beispiel verwendete Paket ist "cam". Es sind aber auch andere
    // Technologiepakete verwendbar (s. Dokumentation).

    // USELIB testlib;

    // Sollen Funktionen einer Bibliothek in der Quelle genutzt werden, müssen
    // sie der Quelle ebenfalls bekannt gemacht werden. Falls die Bibliothek
    // mit dem Namen "testlib" nicht im Projekt existiert, wird die
    // Fehlermeldung
    // "Fehler 10035 Bibliothek "testlib.lib" konnte nicht geladen werden"
    // "Fehler 10032 Bibliothek "testlib" konnte nicht geladen werden"
    // ausgegeben.
    // Wird nicht mit Bibliotheken gearbeitet, kann diese Zeile gelöscht
    // werden.

    // USES header;

    // Mittels USES werden die von einer anderen Quelle (NAME hier "header")
    // exportierten Inhalte importiert und in "sttemp_l_de" nutzbar gemacht.
    // Falls die Quelle mit dem Namen "header" nicht im Projekt existiert,
    // wird die Fehlermeldung
    // "Fehler 10018 Quelle "header" konnte nicht geladen werden"
```

```
// ausgegeben.
```

A.3.2 Typdefinition im Interface

```
// ****************************************************
// * Typdefinition im INTERFACE                       *
// ****************************************************

VAR_GLOBAL CONSTANT
       PI             : REAL := 3.1415;
       ARRAY_MAX1     : INT  := 4;
       ARRAY_MAX2     : INT  := 4;
       COLLECTION_MAX : INT  := 6;
       GLOBARRAY_MAX  : INT  := 12;
END_VAR
// Deklaration einer globalen Konstanten. Dem Bezeichner kann in der Quelle
// kein anderer Wert zugewiesen werden.

// Zwischen TYPE und END_TYPE werden
// anwenderdefinierte Variablentypen (UDT) definiert.
TYPE
    ai16Dim1 : ARRAY [0..ARRAY_MAX1-1] OF INT;
    // Definition eines eindimensionalen Feldes mit vier Feldelementen vom
    // Typ INT unter dem Namen "ai16Dim1". Mit "ai16Dim1" als Datentyp
    // können nun in allen Quellabschnitten eindimensionale Felder
    // vom Typ INT deklariert werden.

    aaDim2 : ARRAY [0..ARRAY_MAX2-1] OF ai16Dim1;
    // Ein zweidimensionales Feld ist ein Feld eindimensionaler Felder.
    // Hier entsteht ein zweidimensionales Feld mit 16 Elementen
    // des Typs INT unter dem Namen "aaDim2"

    eTrafficLight : (RED, YELLOW, GREEN);
    // Definition eines Enumerators "eTrafficLight" als
    // anwenderdefinierter Variablentyp. Variablen diesen Typs können
    // ausschließlich die Werte "RED", "YELLOW" und "GREEN" annehmen.

    sCollection : STRUCT
        toAxisX          : posaxis;
        aInStructDim1    : ai16Dim1;
        eTrafficInStruct : eTrafficLight;
        i16Counter       : INT;
        b16Status        : WORD;
    END_STRUCT;
    // Hier wird eine anwenderdefinierte Struktur erzeugt. Es können
    // elementare Datentypen (hier INT und WORD) oder bereits definierte
    // Anwenderdatentypen (hier "array1dim" und "eTrafficLight") zu
    // einer Struktur zusammengefasst werden. Daneben können auch Typen
    // technologischer Objekte verwendet werden.
    // Im Beispiel enthält die Stuktur ein Element vom TYPE
    // einer Positionierachse (posaxis).
    // Bei der Definition sollte darauf geachtet werden, die Variablen
    // nach ihrer Größe in absteigender Reihenfolge zu sortieren
    // (ARRAY, STRUCT, LREAL, DWORD, INT, BOOL ...)

    aCollection : ARRAY [0..5] OF sCollection;
    // Eine Verschachtelung ist ebenfalls möglich. Der Typ "aCollection"
    // enthält ein Feld aus sechs Elementen vom Typ "sCollection"
```

```
END_TYPE
```

A.3.3 Variablendeklaration im Interface

```
// ****************************************************
// * Variablendeklaration im INTERFACE                *
// ****************************************************

VAR_GLOBAL  // Im Anwenderspeicher der UNIT.
            // Ist auch über B&B Dienste sichtbar.

    gaMyArray : ARRAY [0..GLOBARRAY_MAX-1] OF REAL := [3 (2(4), 2(18))];
    // Beispiel einer Deklaration eines eindimensionalen Feldes ohne
    // vorherige Typdeklaration. Die hier vorgenommene Initialisierung
    // ist wie folgt zu lesen:
    // Jeweils zwei Elemente werden mit dem Wert 4 initialisiert,
    // zwei Elemente mit dem Wert 18. Dieses Muster wird im Feld
    // "gaMyArray" dreimal in Folge angewendet.
    // Die Feldelemente sind also wie folgt belegt:
    // 4, 4, 18, 18, 4, 4, 18, 18, 4, 4, 18, 18.

    gaMy2dim : aaDim2;
    // Beispiel einer Deklaration eines zweidimensionalen Feldes.

    gaMy1dim : ai16Dim1;
    // Beispiel einer Deklaration eines eindimensionalen Feldes unter
    // Verwendung einer Typdeklaration.

    gsMyStruct : sCollection;
    // Erzeugung einer Variablen vom Typ bzw. mit der Struktur von
    // user_struct.

    gaMyArrayOfStruct : aCollection;
    // Die hier erzeugte Variable beinhaltet ein Feld aus
    // Strukturelementen wie sie im Abschnitt TYPE/END_TYPE deklariert
    // wurden.

    gtMyTime : TIME := T#0d_1h_5m_17s_4ms;
    // ...wie elementare Zeittypen und abgeleitete Datentypen.

    geMyTraffic : eTrafficLight := RED;
    // Hier wird ein Enumerator vom Typ "eTrafficLight" angelegt und
    // mit dem Wert "RED" vorbelegt.

    gi16MyInt : INT := -17;
    // Variablen elementarer numerischer Datentypen können in
    // Variablendeklarationen ebenso deklariert werden...

END_VAR

VAR_GLOBAL RETAIN
END_VAR
// Die mit dem Zusatz RETAIN deklarierten Variablen werden im
// RETAIN Datenbereich der eingesetzen Hardware-Plattform gespeichert und
// sind damit netzausfallsicher.


// Die Deklaration von VAR, VAR CONSTANT, VAR_TEMP, VAR_INPUT, VAR_OUTPUT
```

```
    // und VAR_IN_OUT ist hier nicht zulässig.
    // Variablen die in diesem Abschnitt definiert und damit exportiert werden
    // können mittels USES "sttemp_l_de" in einer anderen Quelle (UNIT) wieder
    // importiert werden.

    FUNCTION FCmyFirst;
    FUNCTION_BLOCK FBmyFirst;
    PROGRAM myPRG;
    // Die im Implementationteil definierten Funktionsbausteine (FBs),
    // Funktionen(FCs) und Programme sind hier im Interfaceteil zu exportieren,
    // damit sie in anderen Units verwendet werden können.
    // Nicht exportierte FBs und FCs können nur in dieser Quelle verwendet
    // werden ("information hiding", nur das ins Interface legen,
    // was andere Units unbedingt benötigen).
    // Ein Programm, das nicht exportiert wurde, kann keiner TASK zugeordnet
    // werden!

END_INTERFACE
```

A.3.4 Implementation

```
// ****************************************************
// * IMPLEMENTATION-Teil                              *
// ****************************************************

IMPLEMENTATION
    // Im IMPLEMENTATION Teil einer Unit sind die ausführbaren Codeabschnitte
    // in verschiedenen Programmorganisationseinheiten (POEs) hinterlegt.
    // Eine POE kann ein Programm, ein FC oder ein FB sein.

    VAR_GLOBAL CONSTANT
    END_VAR

    TYPE
    END_TYPE
    // Die Typdefinition kann auch im IMPLEMENTATION Teil vorgenommen werden.
    // Allerdings ist diese Definition dann nicht in einer anderen Quelle
    // importierbar. Die Typdefinition kann jedoch für Variablen in allen POEs
    // der Quelle "sttemp_l_de" verwendet werden. Die Definition von Typen muss
    // vor der Deklaration der Variablen erfolgen.

    VAR_GLOBAL  // Im Anwenderspeicher der UNIT
        gboDigInput1 : BOOL;
        // Boolsche Variable für das Beispiel "EXPRESSION" (s.u).
    END_VAR

    VAR_GLOBAL RETAIN
    END_VAR
    // Die mit dem Zusatz RETAIN deklarierten Variablen werden im
    // RETAIN Datenbereich der eingesetzten Hardware-Plattform gespeichert und
    // sind damit netzausfallsicher.

    // Variablendeklaration im IMPLEMENTATION Teil.
    // Die Deklaration von VAR, VAR CONSTANT, VAR_TEMP, VAR_INPUT, VAR_OUTPUT
    // und VAR_IN_OUT ist hier nicht zulässig.

    EXPRESSION xCond
        xCond := gboDigInput1;
    END_EXPRESSION
    // Definition einer EXPRESSION.
    // Eine EXPRESSION ist ein Sonderfall einer Funktion und kennt nur den
    // Rückgabewert TRUE und FALSE. Sie wird im Zusammenhang mit der
    // Anweisung WAITFORCONDITON verwendet (s. myPRG) und sollte nur verwendet
    // werden, wenn das Programm im Rahmen einer MotionTask zur Ausführung
    // kommt. Falls "gboDigInput1" (gewöhnlich ein digitaler Eingang oder eine
    // Bedingung im Programm) den Wert 1 annimmt, wird der Rückgabewert der
    // EXPRESSION TRUE.
```

A.3.5 Function

```
// ****************************************************
 // * FUNCTION                                         *
 // ****************************************************

 // Die Deklaration eines FB oder FC muss vor der eigentlichen Verwendung
 // (dem Aufruf) in der Quelle platziert werden, damit der Code des
 // Bausteins an der aufrufenden Stelle bereits bekannt ist.

 FUNCTION FCmyFirst : INT
     // Hier beginnt der Anweisungsteil der POE FUNCTION. Der Rückgabewert
     // der Funktion hat in diesem Fall den Typ Integer.
     // Der Stack der aufrufenden TASK wird jedes mal bei Aufruf
     // initialisiert. Der Returnwert liegt auf dem Stack und wird von der
     // FUNCTION geschrieben.

     VAR CONSTANT
     END_VAR

     TYPE
     END_TYPE
     // Die Typdeklaration kann auch in POEs vorgenommen werden. Der
     // grundlegende Unterschied besteht im Gültigkeitsbereich der
     // Typdekaration. Ein in einer POE deklarierter Typ kann nur für
     // Variablen innerhalb der betreffenden POE verwendet werden.

     VAR_INPUT   // Im Stack der aufrufenden TASK, wird bei Aufruf auf
                 // Stack gelegt, optional zu belegen.
     END_VAR

     VAR         // Im Stack der aufrufenden TASK,
                 // wird in FUNCTION genutzt.
     END_VAR

     // Variablendeklaration in einem FC.
     // Die Deklaration von VAR_TEMP, VAR_GLOBAL, VAR_GLOBAL CONSTANT,
     // VAR_GLOBAL RETAIN, VAR_OUTPUT und VAR_IN_OUT ist hier nicht
     // zulässig.

     // Die Verwendung von unitglobalen Variablen zur Datenübergabe an FCs
     // und FBs ist die zur Laufzeit schnellste Möglichkeit. Die Verwendung
     // der Eingangsparameter VAR_INPUT und die Rückgabe mittels des
     // Rückgabewertes ist langsamer, da die Werte jeweils kopiert werden.

     // Anmerkung: Die mit VAR und VAR CONSTANT deklarierten Variablen sind
     // temporär. Beim nächsten Aufruf Stehen die Inhalte aus dem letzten
     // Aufruf im Gegensatz zum FB nicht mehr zur Verfügung.

     // ******************************************
     // * Bereich für Code bzw. Anweisungen des FC *
     // ******************************************)
     // Code liegt im Anwenderspeicher.

     geMyTraffic := YELLOW; // z.B. Umschalten der Ampel.
```

```
        FCmyFirst := 17;
        // Die Funktion liefert in diesem Beispiel den Wert "17" an das
        // aufrufende Programm zurück.

    END_FUNCTION
```

A.3.6 Function Block

```
// ****************************************************
// * FUNCTION_BLOCK                                   *
// ****************************************************

// Die Deklaration eines FB oder FC muss vor der eigentlichen Verwendung
// (dem Aufruf) in der Quelle platziert werden, damit der Code des
// Bausteins an der aufrufenden Stelle bereits bekannt ist.

FUNCTION_BLOCK FBmyFirst
    // Hier beginnt der Anweisungsteil der POE FUNCTION_BLOCK.
    // Instanzdaten liegen abhängig wo die Instanz gebildet wird
    // (siehe Kommentare am Template-Ende) im Anwenderspeicher der UNIT
    // oder TASK und werden bei STOP->RUN bzw. Starten der TASK
    // initialisiert.
    // Der Zeiger auf die Instanzdaten wird beim Aufruf übergeben.

    VAR CONSTANT
    END_VAR
    // Die unter VAR und VAR CONSTANT deklarierten Variablen sind
    // statisch, d.h. die Inhalte sind beim nächsten Bausteinaufruf noch
    // vorhanden und gültig.

    TYPE
    END_TYPE
    // Die Typdefinition kann auch in POEs vorgenommen werden. Der
    // grundlegende Unterschied besteht im Gültigkeitsbereich der
    // Typdefinition. Ein in einer POE definierter Typ kann nur für
    // Variablen innerhalb der betreffenden POE verwendet werden.

    VAR_INPUT   // Im Anwenderspeicher der UNIT oder TASK,
                // optional bei Aufruf zu belegen.
    END_VAR

    VAR_IN_OUT  // Im Anwenderspeicher der UNIT oder TASK,
                // Referenz muss bei Aufruf belegt werden.
    END_VAR

    VAR_OUTPUT  // Im Anwenderspeicher der UNIT oder TASK.
    END_VAR

    VAR         // Im Anwenderspeicher der UNIT oder TASK,
                // kann im FB verwendet werden.
    END_VAR

    VAR_TEMP    // Im Stack der aufrufenden TASK,
                // wird bei jedem Aufruf initialisiert.
    END_VAR


    // Variablendeklaration in einem FB.
    // Die Deklaration von VAR_GLOBAL, VAR_GLOBAL CONSTANT und
    // VAR_GLOBAL RETAIN ist hier nicht zulässig.

    // ******************************************
```

```
        // * Bereich für Code bzw. Anweisungen des FB *
        // ********************************************

        geMyTraffic := GREEN;  // z.B. Umschalten der Ampel.


    END_FUNCTION_BLOCK
```

A.3.7 Program

```
// ******************************************************
// * PROGRAM                                            *
// ******************************************************

PROGRAM myPRG
    // Hier beginnt der Anweisungsteil der POE PROGRAM.

    VAR CONSTANT
    END_VAR

    TYPE
    END_TYPE
    // Die Typdefinition kann auch in POEs vorgenommen werden. Der
    // grundlegende Unterschied besteht im Gültigkeitsbereich der
    // Typdefinition. Ein in einer POE definierter Typ kann nur für
    // Variablen innerhalb der betreffenden POE verwendet werden.

    VAR  // Im Anwenderspeicher der TASK.

        instFBMyFirst : FBmyFirst;
        // Um einen FB aufrufen zu können, muss ein Bereich für statische
        // Variablen (Bilden einer Instanz) erzeugt werden. Dabei handelt
        // es sich um das "Gedächtnis" des FB.

        retFCMyFirst : INT;
        // Variable für den Rückgabewert der Funktion.
    END_VAR

    VAR_TEMP    // Im Stack der TASK, in jedem Durchlauf initialisiert.
    END_VAR
    // Variablendeklaration in einem PROGRAMM.
    // Die Deklaration von VAR_GLOBAL, VAR_GLOBAL CONSTANT,
    // VAR_GLOBAL RETAIN, VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT
    // ist hier nicht zulässig.

    // Anmerkung: Ob die mittels VAR deklarierten lokalen Variablen
    // temporär sind, hängt von dem Taskkontext ab, in dem das PROGRAM
    // verwendet wird.
    //
    // In nichtzyklischen Tasks (StartupTask, ShutdownTask, MotionTasks,
    // SystemInterruptTasks und UserInterruptTasks) stehen die vorherigen
    // Inhalte von VAR und VAR_TEMP nicht mehr zur Verfügung.
    // Die Variablen sind somit temporär.
    //
    // In zyklischen Tasks (BackgroundTask, IPOsynchronousTask,
    // IPOsynchronousTask_2 und TimerInterruptTasks) bleiben die Inhalte
    // von im Abschnitt VAR deklarierten Variablen für den folgenden
    // Durchlauf erhalten. Die Variablen sind somit statisch.
    // Variablen aus VAR_TEMP sind in jedem Fall temporär.

    instFBMyFirst ();
    // Aufruf des FB mit einer gültigen Instanz.

    retFCMyFirst := FCmyFirst();
```

```
        // Aufruf des FC und Zuweisung des Rückgabewerts.

        WAITFORCONDITION xCond WITH TRUE DO
            // Die hier programmierten Anweisungen kommen sofort zur
            // Ausführung wenn die in der zugehörigen EXPRESSION definierte
            // Bedingung "xcond" logisch wahr wird.
            ;
        END_WAITFORCONDITION;
        // WAITFORCONDITION kommt im Allgemeinen nur in MotionTasks zu
        // Anwendung. Diese werden auf der Stelle angehalten und die im
        // EXPRESSION-Ausdruck definierte Bedingung wird hochprior überprüft.

    END_PROGRAM

END_IMPLEMENTATION
```

Index

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z