SIEMENS

SIMATIC S7

SCL für S7-300/400 Bausteine programmieren

Handbuch

Dieses Handbuch hat die Bestellnummer 6ES7811-1CA02-8AA0

Sicherheitstechnische Hinweise

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise sind durch ein Warndreieck hervorgehoben und je nach Gefährdungsgrad folgendermaßen dargestellt:

Gefahr

bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **werden**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Warnung

bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **können**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht

bedeutet, daß eine leichte Körperverletzung oder ein Sachschaden eintreten können, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Hinweis

ist eine wichtige Information über das Produkt, die Handhabung des Produktes oder den jeweiligen Teil der Dokumentation, auf den besonders aufmerksam gemacht werden soll.

Qualifiziertes Personal

Inbetriebsetzung und Betrieb eines Gerätes dürfen nur von **qualifiziertem Personal** vorgenommen werden. Qualifiziertes Personal im Sinne der sicherheitstechnischen Hinweise dieses Handbuchs sind Personen, die die Berechtigung haben, Geräte, Systeme und Stromkreise gemäß den Standards der Sicherheitstechnik in Betrieb zu nehmen, zu erden und zu kennzeichnen.

Bestimmungsgemäßer Gebrauch

Beachten Sie folgendes:

Warnung

Dieses Produkt darf nur für die im Katalog und in der technischen Beschreibung vorgesehenen Einsatzfälle und nur in Verbindung mit von Siemens empfohlenen bzw. zugelassenen Fremdgeräten und -komponenten verwendet werden.

Warenzeichen

SIMATIC®, SIMATIC HMI® und SIMATIC NET® sind eingetragene Warenzeichen der SIEMENS AG.

Die übrigen Bezeichnungen in dieser Schrift könnenWarenzeichen sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen können.

Siemens AG
Bereich Automatisierungs-und Antriebstechnik
Geschäftsgebiet Industrie-Automatisierungssysteme
Postfach 4848, D-90327 Nürnberg

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, und notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Vorwort

Zweck des Handbuchs

Dieses Handbuch unterstützt Sie bei der Erstellung von Anwenderprogrammen in der Programmiersprache SCL. Die prinzipiellen Vorgehensweisen für die Programmerstellung mit dem SCL-Editor, SCL-Compiler und SCL-Debugger werden erläutert.

Außerdem enthält das Handbuch einen Referenzteil über die Sprachelemente der Programmiersprache SCL. Dabei werden Syntax und Funktionsweise der einzelnen Sprachelemente beschrieben.

Leserkreis

Dieses Handbuch richtet sich an Programmierer von S7-Programmen, Inbetriebsetzer und Servicepersonal. Allgemeine Kenntnisse auf dem Gebiet der Automatisierungstechnik sind dabei hilfreich.

Gültigkeitsbereich des Handbuchs

Dieses Handbuch ist gültig für die Programmiersoftware STEP 7 ab Version 3.0. Es gilt für das Softwarepaket STEP 7 Basissoftware.

Normerfüllung nach IEC 1131-3

SCL entspricht der in der Norm DIN EN-61131-3 (int. IEC 1131-3) festgelegten Sprache ”Structured Control Language”, wobei hinsichtlich der Operationen wesentliche Unterschiede bestehen. Genaue Aussagen zur Normerfüllung finden Sie in der Normerfüllungstabelle in der NORM.TAB-Datei von STEP 7.

Einordnung in die Informationsland-schaft

Zur Unterstützung Ihrer Konfigurierung und Programmierung eines S7-Automatisierungssystems gibt es eine umfangreiche Anwenderdokumentation, die für eine selektive Benutzung vorgesehen ist. Die folgenden Erläuterungen sollen Ihnen gemeinsam mit dem Bild 1-1 die Nutzung der Anwenderdokumentation erleichtern.

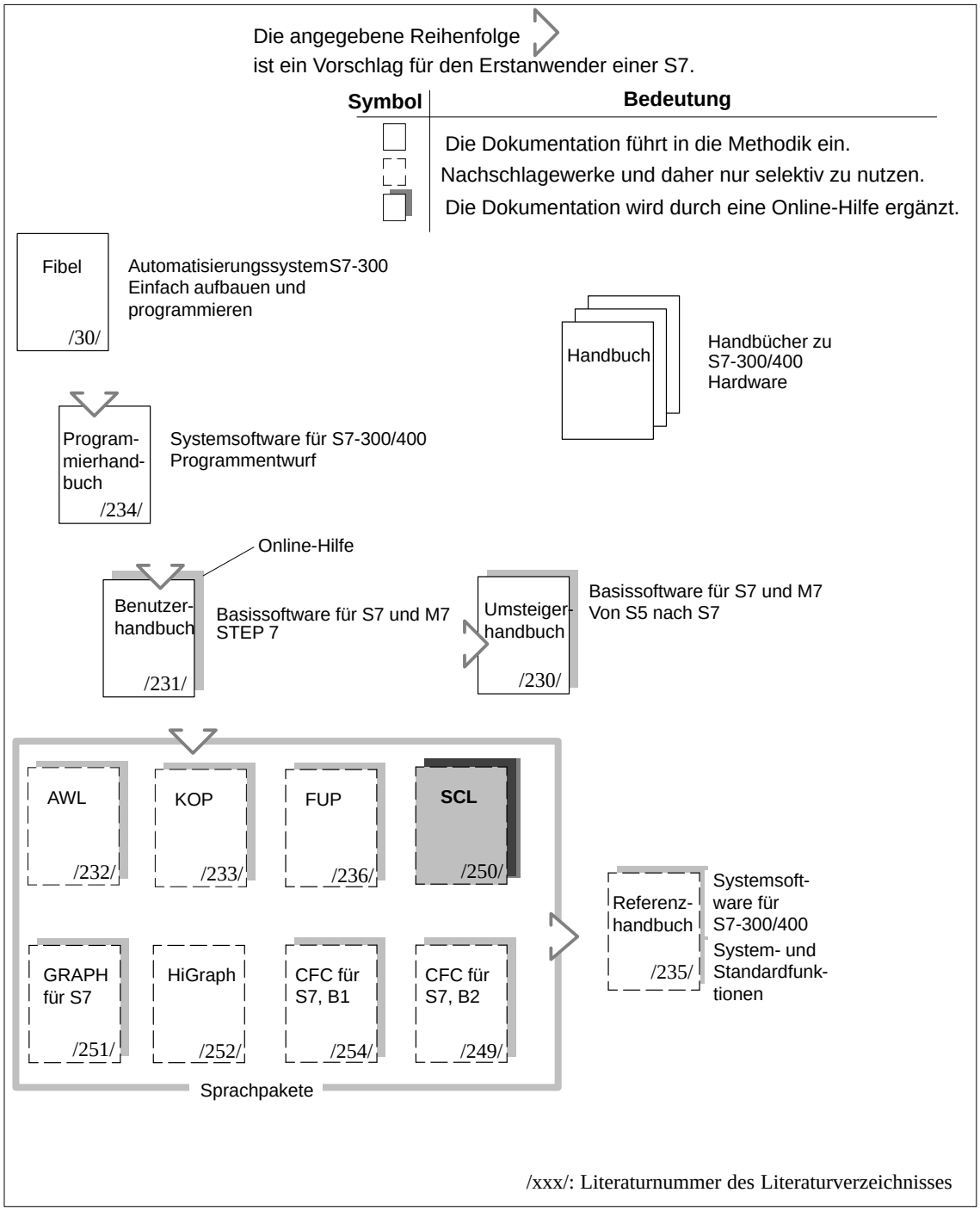


Bild 1-1 Informationslandschaft

Tabelle 1-1 Informationsinhalte

Titel	Inhalt
Fibel S7-300 Einfach aufbauen und programmieren	Die Fibel bietet einen sehr einfachen Einstieg in die Methodik des Aufbaus und der Programmierung einer S7-300/400. Sie ist insbesondere für den Erstanwender eines S7-Automatisierungssystems geeignet.
Programmierhandbuch Programmentwurf S7-300/400	Das Programmierhandbuch "Programmentwurf S7-300/400" vermittelt das grundlegende Wissen über den Aufbau des Betriebssystems und eines Anwenderprogramms einer S7-CPU. Es sollte vom Erstanwender einer S7-300/400 dazu genutzt werden, sich einen Überblick über die Programmiermethodik zu verschaffen und darauf das Design seines Anwenderprogramms aufzubauen.
Referenzhandbuch System- und Standardfunktionen S7-300/400	Die S7-CPUs enthalten in das Betriebssystem integrierte Systemfunktionen und Organisationsbausteine, die Sie bei der Programmierung nutzen können. Das Handbuch gibt Ihnen einen Überblick über die bei S7 verfügbaren Systemfunktionen, Organisationsbausteine und ladbare Standardfunktionen sowie – als Nachschlageinformation – detaillierte Schnittstellenbeschreibungen für deren Nutzung in Ihrem Anwenderprogramm.
Benutzerhandbuch STEP 7	Das Benutzerhandbuch "STEP 7" erläutert Ihnen die prinzipielle Nutzung und die Funktionen der Automatisierungssoftware STEP 7. Als Erstanwender von STEP 7 ebenso wie als Kenner von STEP 5 verschafft Ihnen das Handbuch einen Überblick über die Vorgehensweise bei der Konfigurierung, Programmierung und Inbetriebnahme einer S7-300/400. Beim Arbeiten mit der SW können Sie gezielt auf die Online-Hilfe zugreifen, die Ihnen Unterstützung zu den Detailfragen der SW-Nutzung bietet.
Umsteigerhandbuch Von S5 nach S7	Das Benutzerhandbuch "Von S5 nach S7" benötigen Sie, wenn Sie vorhandene S5-Programme konvertieren wollen, um diese dann in S7-CPUs zum Ablauf zu bringen. Das Handbuch gibt Ihnen einen Überblick über die Vorgehensweise und die Nutzung des Konverters; die detaillierte Bedienung der Konverterfunktionen entnehmen Sie bitte der Online-Hilfe. Ebenso erhalten Sie über die Online-Hilfe die Schnittstellenbeschreibung der verfügbaren konvertierten S7-Funktionen.
Handbücher AWL, KOP, FUP, SCL[1]	Die Handbücher zu den Sprachpaketen AWL, KOP, FUP und SCL enthalten sowohl die Benutzeranleitungen als auch die Sprachbeschreibung. Sie benötigen für die Programmierung einer S7-300/400 nur eine der Sprachen, können aber auch bei Bedarf die Sprachen innerhalb eines Projektes mischen. Für den erstmaligen Einsatz der Sprachen ist es empfehlenswert, sich anhand des Programmierhandbuchs "Programmentwurf S7-300/400" mit der Methodik der Programmerstellung vertraut zu machen. Bei Arbeiten mit der jeweiligen Software können Sie die Online-Hilfe nutzen, die Ihnen alle Detailfragen zu der Nutzung der zugehörigen Editoren/Compiler beantwortet.
Handbücher GRAPH[1] , HiGraph[1], CFC[1]	Die Sprachen GRAPH, HiGraph, CFC bieten zusätzlich Möglichkeiten, Ablaufsteuerungen, Zustandssteuerungen oder graphische Verschaltungen von Bausteinen zu realisieren. Die Handbücher enthalten sowohl die Benutzeranleitung als auch die Sprachbeschreibung. Für einen erstmaligen Einsatz der Sprache ist es empfehlenswert, sich mit der Methodik der Programmerstellung anhand des Programmierhandbuchs "Programmentwurf S7-300/400" vertraut zu machen. Beim Arbeiten mit der jeweiligen Software können Sie zudem die Online-Hilfe nutzen (Ausnahme HiGraph), die Ihnen die Detailfragen zu der Nutzung der Editoren/Compiler beantwortet.

1 Optionspakete zu Systemsoftware für S7-300/400

Wegweiser

Dieses Handbuch zu SCL setzt theoretische Kenntnisse über S7-Programme voraus, die Sie im Programmierhandbuch **/234/** nachlesen können. Da die Sprachpakete auf der Basissoftware STEP 7 aufsetzen, sollten Sie bereits Kenntnisse im Umgang mit der Basissoftware haben, die im Benutzerhandbuch **/234/** vermittelt werden.

Das SCL-Handbuch ist in folgende Themenbereiche gegliedert:

- Kapitel 1 liefert Ihnen allgemeine Informationen zum Programmieren mit SCL.
- In Kapitel 2 wird ein Entwurfsprozeß anhand eines Beispiels behandelt, das Sie auch zum Ablauf bringen können.
- Kapitel 3-6 zeigen Ihnen den Umgang mit der Entwicklungsumgebung von SCL. Sie lernen Editor, Compiler und Debugger von SCL kennen.
- Kapitel 7-19 bilden den Referenzteil, der Ihnen genaue Informationen zur Funktionsweise der einzelnen SCL-Anweisungen geben soll.

Im Anhang finden Sie:

- Die komplette Syntaxdarstellung von SCL.
- Ein Glossar, in dem wichtige Begriffe erklärt sind.
- Ein Stichwortverzeichnis, das Ihnen hilft, die Textstellen zu wichtigen Stichworten schnell zu finden.

Konventionen

Hinweise auf weitere Dokumentation sind mit Hilfe von Literaturnummern in Schrägstrichen /.../ angegeben. Damit können Sie dem Literaturverzeichnis am Ende des Handbuchs den genauen Titel der Dokumentation entnehmen.

Weitere Unterstützung

Bei Fragen zur Nutzung der beschriebenen Software, die Sie weder in der Papierdokumentation noch in der Online-Hilfe beantwortet finden, wenden Sie sich bitte an Ihre Siemens-Ansprechpartner in den für Sie zuständigen Vertretungen und Geschäftsstellen. Die Adressen finden Sie im Anhang von **/70/** bzw. **/100/** oder in Katalogen und in Compuserve (`go autforum`).

Darüber hinaus steht Ihnen unsere Hotline zur Verfügung:

Tel. +49(911) 895–7000 (Fax 7001)

Bei Fragen oder Anmerkungen zum vorliegenden Handbuch füllen Sie bitte den Fragebogen am Ende des Handbuchs aus und schicken Sie ihn an die dort angegebene Adresse. Bitte tragen Sie dort auch Ihre persönliche Bewertung des Handbuchs ein.

Um Ihnen den Einstieg in das Automatisierungssystem SIMATIC S7 zu erleichtern, bieten wir entsprechende Kurse an. Wenden Sie sich bitte an Ihr regionales Trainingscenter oder an das zentrale Trainingscenter in:

D-90327 Nürnberg, Tel. 0911 / 895 3154.

Besonderer Hinweis

Der Benutzerteil dieses Handbuchs enthält keine ausführlichen Anleitungen mit einzelnen Schrittfolgen, sondern soll grundsätzliche Vorgehensweisen verdeutlichen. Genauere Informationen zu den Dialogen der Software und deren Bearbeitung finden Sie jeweils in der Online-Hilfe.

Inhaltsverzeichnis

Teil 1: Entwerfen

Produktübersicht

1

Programmiersprache SCL

Immer häufiger müssen Automatisierungssysteme neben den klassischen Steuerungsaufgaben heute auch Datenverwaltungsaufgaben und komplexere mathematische Aufgaben übernehmen. Speziell für diese Aufgaben bieten wir Ihnen SCL für S7300/400 (Structured Control Language), die Programmiersprache, die das Programmieren leichter macht - genormt nach IEC 113-3.

SCL unterstützt Sie außer bei ”normalen” Steuerungsaufgaben auch bei umfangreichen Anwendungen und ist damit den ”klassischen” Programmiersprachen in den folgenden Anwendungsbereichen überlegen:

- Datenverwaltung
- Prozeßoptimierung
- Rezepturverwaltung
- mathematische/statistische Aufgaben.

Technische Daten

Um mit SCL arbeiten zu können, brauchen Sie ein SIMATIC-Programmiergerät oder einen PC (ab 80486 Prozessor, 16 MByte Hauptspeicher).

Sprachvorrat

Operatoren	`Potenz/Arithmetik`
	`Vergleiche/`
	`Verknüpfungen`

Funktionen	`Zeiten/Zähler/` `Funktionsbausteinaufrufe`
Kontrollstrukturen	`Schleifen (FOR/WHILE/REPEAT)` `Alternativen (IF THEN/CASE/GOTO)`

Datentypen

elementare	`BOOL/BYTE/WORD/DWORD/`
	`INT/DINT/REAL/`
	`DATE/TIME/TIME_OF_DAY/S5TIME`

zusammengesetzte	`Strings/Felder/Strukturen/anwenderdefinierte Strukturen/DATE_AND_TIME`

Kapitelübersicht

Im Kapitel	finden Sie	auf Seite
1.1	Was ist SCL?	1-2
1.2	Welche Vorteile bietet Ihnen SCL?	1-3
1.3	Leistungsmerkmale der Entwicklungsumgebung	1-5

1.1 Was ist SCL?

Höhere Programmiersprache

SCL (*Structured Control Language*) ist eine höhere Programmiersprache, die sich an PASCAL orientiert. Die Sprache basiert auf einer Norm für SPS (=speicherprogrammierbare Steuerungen).

Die Norm DIN EN-61131-3 (int. IEC 1131-3) standardisiert die Programmiersprachen für speicherprogrammierbare Steuerungen. Basis für SCL ist der Teil *"strukturierter Text"*. Genaue Aussagen zur Normerfüllung finden Sie in der "Compliance List" in der NORM.TAB-Datei von STEP 7.

SCL enthält neben Hochsprachenelementen auch typische Elemente der SPS wie Eingänge, Ausgänge, Zeiten, Merker, Bausteinaufrufe usw. als Sprachelemente. D.h. SCL ergänzt und erweitert die Programmiersoftware STEP 7 mit ihren Programmiersprachen KOP, FUP und AWL.

Entwicklungsumgebung

Zur optimalen Verwendung und zum praktischen Einsatz von SCL gibt es eine leistungsfähige Entwicklungsumgebung, die sowohl auf spezifische Eigenschaften von SCL als auch auf STEP 7 abgestimmt ist. Diese Entwicklungsumgebung besteht aus folgenden Komponenten:

- einem **Editor**, um Programme bestehend aus Funktionen (FC), Funktionsbausteinen (FB), Organisationsbausteinen (OB), Datenbausteinen (DB) und anwenderdefinierten Datentypen (UDT) zu programmieren. Der Programmierer wird dabei durch leistungsfähige Funktionen unterstützt.
- einem **Batch-Compiler** um das zuvor editierte Programm in MC7-Maschinencode zu übersetzen. Der erzeugte MC7-Code ist ab der CPU 314 auf allen CPUs des Automatisierungssystem S7-300/400 ablauffähig.
- einem **Debugger**, um eine Suche nach logischen Programmfehlern in einer fehlerfreien Übersetzung zu ermöglichen. Die Fehlersuche erfolgt dabei in der Quellsprache.

Die einzelnen Komponenten sind einfach und komfortabel zu handhaben, da sie unter Windows 95 ablaufen und so auch alle Vorteile dieses Betriebssystems nutzen.

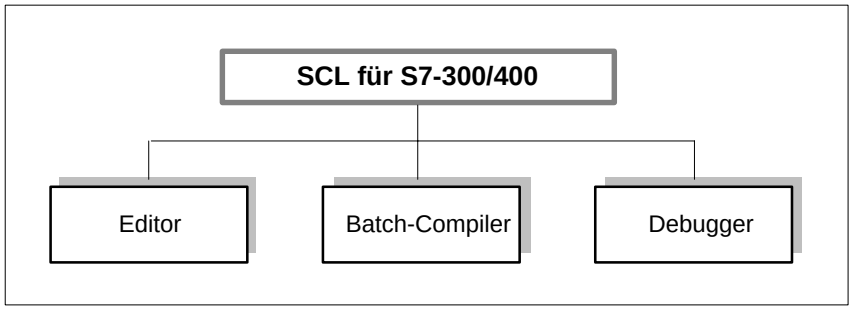


Bild 1-1 *Entwicklungsumgebung von SCL*

1.2 Welche Vorteile bietet Ihnen SCL?

Höhere Programmiersprache

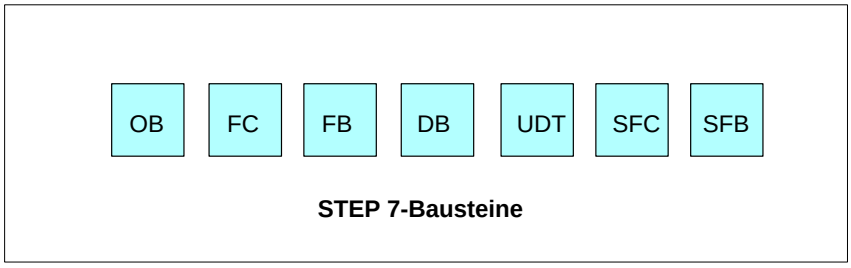
Mit SCL werden alle Vorteile einer höheren Programmiersprache angeboten. SCL enthält aber auch einige Eigenschaften, die speziell zur Unterstützung von strukturierter Programmierung entworfen wurden, z. B.:

- die Bausteinstruktur von STEP 7
- vorgefertigte Bausteine
- Kompatibilität zu STEP 5

Bewährte Bausteinstruktur von STEP 7

SCL ist optimal auf die Lösung aller Aufgaben zugeschnitten, die bei Automatisierungsprojekten anfallen, so daß Sie zusammen mit STEP 7 in allen Phasen des Projekts effektiv arbeiten können.

SCL unterstützt insbesondere das Bausteinkonzept von STEP 7 und ermöglicht daher neben AWL, KOP und FUP die normkonforme Programmierung von Bausteinen.



STEP 7-Bausteine

Bausteinarten

Die STEP 7-Bausteine sind durch ihre Funktion, ihre Struktur oder ihren Verwendungszweck abgegrenzte Teile eines Anwenderprogramms. Mit SCL können Sie die folgenden Bausteine erstellen:

Abkürzung	Bausteinart	Funktion
OB	Organisationsbaustein	Schnittstelle zwischen Betriebssystem und Anwenderprogramm.
FC	Funktion	Baustein mit der Möglichkeit zur Parameterübergabe ohne Gedächtnis.
FB	Funktionsbaustein	Baustein mit der Möglichkeit zur Parameterübergabe und mit Gedächtnis (= Speicher).
DB	Datenbaustein	Baustein zur Ablage von Anwenderdaten.
UDT	Anwenderdefinierter Datentyp	Baustein zur Ablage eines anwenderdefinierten Datentyps

Vorgefertigte Bausteine

Nicht jede Funktion müssen Sie selbst programmieren. Sie können auch auf vorgefertigte Bausteine zurückgreifen. Sie sind im Betriebssystem der Zentralbaugruppen oder in Bibliotheken *(S7lib)* des STEP 7-Basispakets vorhanden und können z. B. für die Programmierung von Kommunikationsfunktionen genutzt werden. Im einzelnen sind das folgende Bausteinarten:

Abkürzung	Bausteinart	Funktion
SFC	Systemfunktion	Eigenschaften wie eine Funktion (FC)
SFB	Systemfunktionsbaustein	Eigenschaften wie ein Funktionsbaustein (FB)

Mischbarkeit der Bausteine

Bausteine, die mit SCL programmiert sind, können Sie mit AWL-, KOP- und FUP-Bausteinen mischen. Das bedeutet, daß ein mit SCL programmierter Baustein einen anderen Baustein, der in AWL, KOP oder FUP programmiert ist, aufrufen kann. Entsprechend können SCL Bausteine auch in AWL-, KOP- und FUP-Programmen aufgerufen werden. Die Programmiersprachen von STEP 7 und SCL (als Optionspaket) ergänzen sich somit optimal.

Rückübersetzbarkeit

SCL-Bausteine können im konkreten Anwendungsfall in die STEP 7-Programmiersprache AWL (Anweisungsliste) rückübersetzt werden. Die Rückübersetzung nach SCL ist nicht möglich.

Kompatibilität mit STEP 5

Bausteine, die Sie mit SCL in STEP 5 programmiert haben, sind von einigen Ausnahmen abgesehen, aufwärtskompatibel, d.h. diese Bausteine können auch mit SCL für STEP 7 editiert, übersetzt und getestet werden.

Programmiermethoden

SCL unterstützt durch moderne Methoden des Software-Engineerings die strukturierte Programmierung.

Erlernbarkeit

SCL ist mit etwas Erfahrung in einer höheren Programmiersprache leicht erlernbar, denn das Repertoire der in SCL zur Verfügung stehenden Sprachkonstrukte orientiert sich an höheren Programmiersprachen.

1.3 Leistungsmerkmale der Entwicklungsumgebung

Editor

Der SCL-Editor ist ein Texteditor, mit dem beliebige Texte bearbeitet werden können. Die zentrale Aufgabe, die Sie mit ihm durchführen, ist das Erzeugen und Bearbeiten von Quelldateien für STEP 7-Programme. In einer Quelldatei können Sie einen oder mehrere Bausteine programmieren:

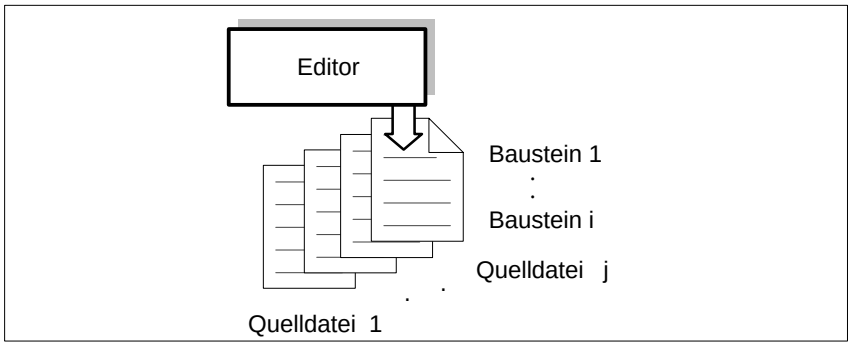


Bild 1-2 *SCL-Editor*

Der SCL-Editor erlaubt:

- das Editieren einer kompletten Quelldatei mit einem oder mehreren Bausteinen.
- das Editieren einer Übersetzungssteuerdatei, mit der Sie das Übersetzen mehrerer Quelldateien automatisieren können.
- das Benutzen zusätzlicher Funktionen, die Ihnen eine komfortable Bearbeitung des Quelltextes ermöglicht, z. B. Suchen und Ersetzen.
- die optimale Einstellung des Editors nach Ihren Anforderungen.

Während der Eingabe erfolgt keine Syntaxprüfung.

Compiler

Nachdem Sie ihre Quelldateien mit dem SCL-Editor erstellt haben, müssen Sie diese in MC7-Code übersetzen.

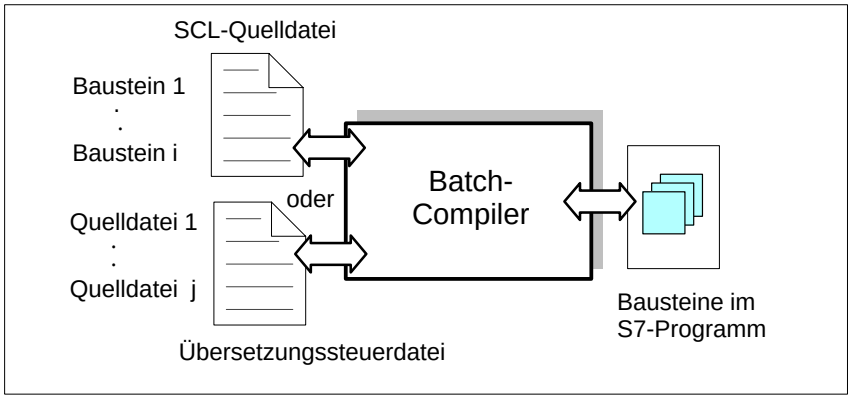


Bild 1-3 *SCL-Compiler*

Der SCL-Compiler erlaubt:

- das Übersetzen einer SCL-Quelldatei mit mehreren Bausteinen in einem Übersetzungslauf.
- das Übersetzen mehrerer SCL-Quelldateien mittels einer Übersetzungssteuerdatei, die die Namen der Quelldateien enthält.
- das optimale Einstellen des Compilers nach Ihren Anforderungen.
- das Anzeigen aller Fehler und Warnungen, die beim Übersetzen auftreten.
- ein einfaches Lokalisieren der fehlerhaften Stelle im Quelltext, optionell mit Fehlerbeschreibung und Angaben zur Fehlerbeseitigung.

Debugger

Der SCL-Debugger bietet die Möglichkeit, ein Programm in seinem Ablauf im AS zu kontrollieren und damit mögliche logische Fehler zu finden.

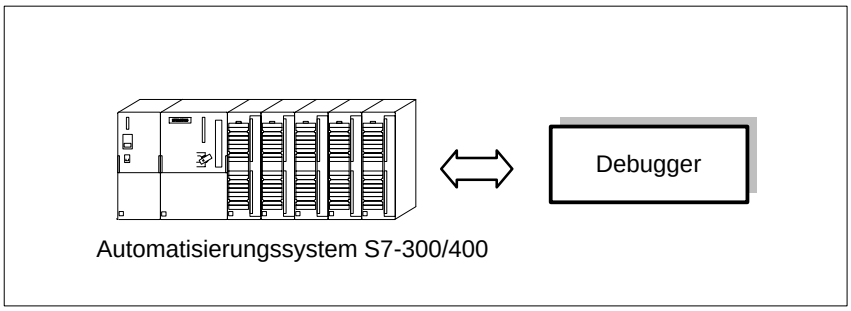


Bild 1-4 *SCL-Debugger*

SCL bietet dazu zwei verschiedene Testmodi an:

- das **schrittweise** Beobachten. Hier wird der logische Programmablauf nachvollzogen. Sie können den Programmalgorithmus Anweisung für Anweisung ausführen und in einem Ergebnisfenster beobachten, wie sich die dabei bearbeiteten Variableninhalte ändern
- das **kontinuierliche** Beobachten. Damit können Sie eine Gruppe von Anweisungen innerhalb eines Bausteins der Quelldatei testen. Während des Testlaufs werden die Werte der Variablen und Parameter in chronologischer Abfolge angezeigt und - sofern möglich - zyklisch aktualisiert.

STEP 7-Basispaket

Die SCL-Entwicklungsumgebung bietet die Möglichkeit, Funktionen des STEP 7-Basispakets wie Anzeigen und Ändern des Betriebszustandes der CPU und Einstellen der Uhrzeit direkt von SCL aus zu verwenden.

Entwerfen eines SCL-Programms

2

Entwerfen

Die Praxis zeigt: Am einfachsten und schnellsten programmieren Sie, wenn Sie die Aufgabe strukturieren, also in einzelne, eigenständige Teile zerlegen. Dabei unterstützt Sie SCL. Denn mit SCL können Sie Einzelbausteine rationell und effizient entwerfen.

Dieses Kapitel beschreibt anhand eines Beispiels wie Sie ein Anwenderprogramm in SCL entwerfen und implementieren können. Mit den mitgelieferten Testdaten und Ihren Ein- und Ausgabebaugruppen können Sie das Beispiel selbst zum Ablauf bringen.

Kapitelübersicht

2.1 Überblick

Zielsetzung

Der Entwurfteil soll Ihnen aufzeigen, wie Sie SCL effektiv einsetzen können. Fragen, die am Anfang häufig auftreten sind z. B.:

- Wie kann ich bei der Programmerstellung mit SCL vorgehen?
- Welche Sprachmittel von SCL bieten sich zur Lösung der Aufgabe an?
- Welche Testfunktionen stehen mir zur Verfügung?

Diese und andere Fragen sollen in diesem Kapitel beantwortet werden.

SCL-Sprachmittel

Im Beispiel werden u.a. folgende SCL-Sprachelemente vorgestellt:

- Aufbau und Einsatz der verschiedenen Bausteinarten von SCL
- Aufruf der Bausteine mit Parameter-Übergabe und -Auswertung
- Verschiedene Ein- und Ausgabeformate
- Programmierung mit elementaren Datentypen und Feldern
- Initialisierung von Variablen
- Programmstrukturen mit Verwendung von Verzweigungen und Schleifen.

Hardware für das Beispiel

Sie können das Beispielprogramm mit SIMATIC S7-300 oder SIMATIC S7-400 zum Ablauf bringen und benötigen dazu folgende Peripherie:

- eine Eingabebaugruppe mit 16 Kanälen
- eine Ausgabebaugruppe mit 16 Kanälen

Testfunktionen

Das Programm ist so aufgebaut, daß Sie es schnell über die Schalter an der Eingabe und die Anzeigen an der Ausgabe testen können. Für einen ausführlichen Test benutzen Sie am besten die Testfunktionen von SCL, siehe Kapitel 6.

Weiterhin stehen Ihnen sprachübergreifend alle Möglichkeiten des STEP 7-Basispaket zur Verfügung.

2.2 Aufgabenstellung

Übersicht

Die Meßwerte sollen über eine Eingabebaugruppe erfaßt, sortiert und verarbeitet werden. Der geforderte Wertebereich der Meßwerte beträgt 0 bis 255. Für die Eingabe wird daher ein Byte benötigt.

Als Verarbeitungsfunktionen sind Wurzel und Quadrat zu benutzen. Die Ergebnisse sollen über einer Ausgabebaugruppe angezeigt werden. Dafür wird ein Wort benötigt wird. Die Programmsteuerung soll über ein Eingabebyte erfolgen.

Meßwerte erfassen

Ein Meßwert, der über die 8 Eingabeschalter eingestellt wird, soll genau dann in das Meßwertefeld im Speicher übernommen werden, wenn am Eingabeschalter eine Flanke erkannt wird (siehe auch Bild 2-1). Das Meßwertefeld soll als Ringpuffer mit maximal 8 Einträgen organisiert sein.

Meßwerte verarbeiten

Wenn am Sortierschalter eine Flanke erkannt wird, so sind die im Meßwertfeld gespeicherten Werte aufsteigend zu sortieren. Danach sollen für jeden Wert Berechnungen von Wurzel und Quadrat durchgeführt werden.

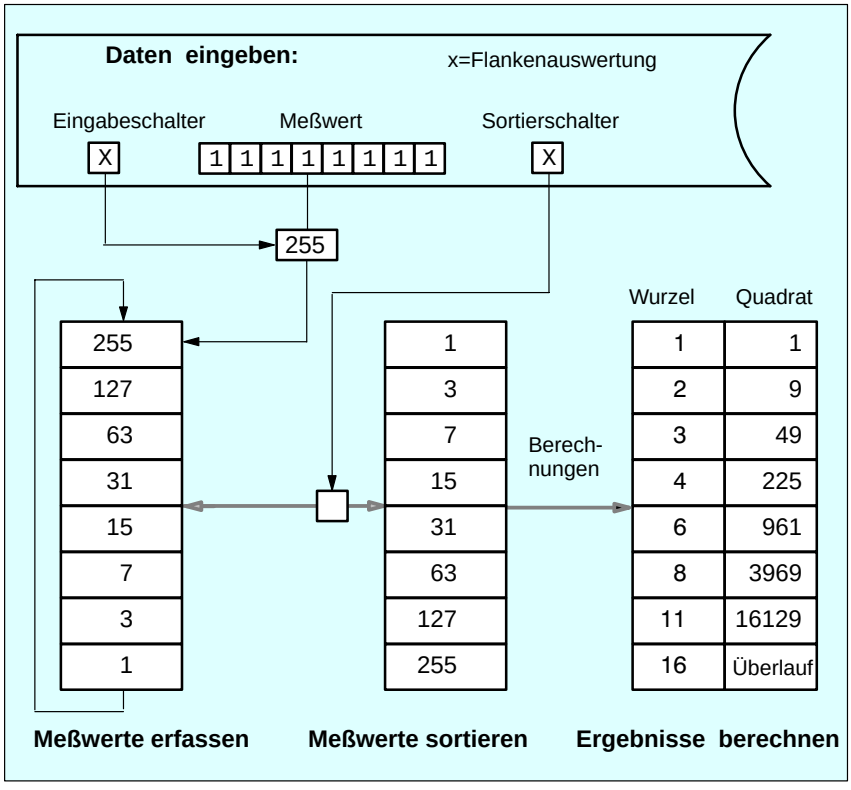


Bild 2-1 *Erfassen und Verarbeiten der Meßwerte*

Einstellbare Ausgabe

Da immer nur ein Wert angezeigt werden kann, soll es folgende Auswahlmöglichkeiten geben:

- Auswahl eines Elements innerhalb einer Liste
- Auswahl zwischen Meßwert, Wurzel und Quadrat

Die Auswahl eines Elements innerhalb einer Liste soll so realisiert werden, daß ein Listenelement durch folgende Einstellung über Schalter adressiert wird:

- Mit drei Schaltern wird eine Codierung eingestellt, die übernommen wird, wenn am vierten Schalter, dem Codierschalter, eine Flanke erkannt wird. Daraus wird die Adresse berechnet, mit der die Ausgabe adressiert wird.
- Mit der gleichen Adresse werden drei Werte, Meßwert, Wurzel und Quadrat, für die Ausgabe bereitgestellt. Um daraus einen Wert auszuwählen, sind zwei Umschalter vorzusehen (siehe Bild 2-2).

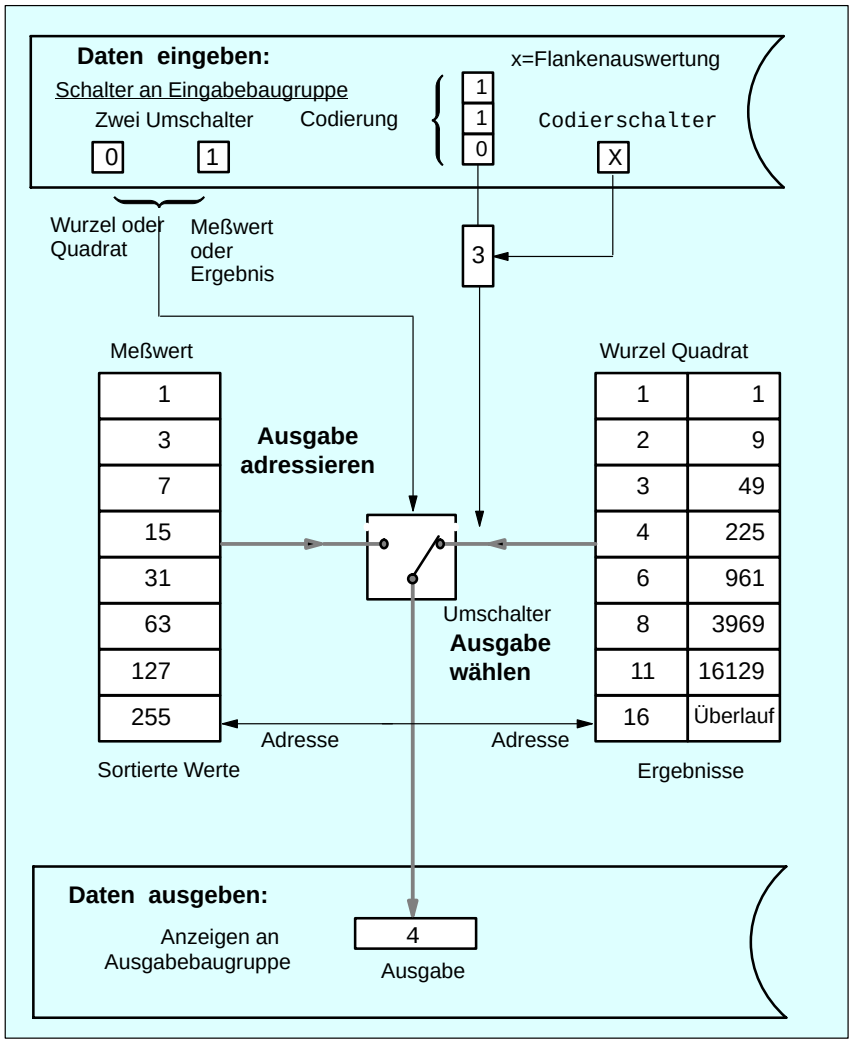


Bild 2-2 *Einstellbare Ausgabe*

2.3 Lösung mit SCL-Bausteinen

Übersicht

Die beschriebene Aufgabe lösen Sie am besten in Form eines **strukturierten SCL-Programms**. Ein solches Programm ist modular aufgebaut, d.h. in Bausteine gegliedert, die jeweils eine bestimmte Teilaufgabe übernehmen. Bei SCL stehen Ihnen wie bei den Programmiersprachen von STEP 7 eine Reihe von Bausteinarten zur Verfügung. Informationen hierzu finden Sie in den Kapiteln 1, 7 und 8).

Schritte zur Lösung

Bei der Lösung können Sie in folgenden Schritten vorgehen:

1. Festlegen der Teilaufgaben
2. Auswahl und Zuordnung der möglichen Bausteinarten
3. Festlegen der Schnittstellen zwischen den Bausteinen
4. Festlegen der Ein-/Ausgabeschnittstelle
5. Bausteine programmieren.

2.3.1 Festlegen der Teilaufgaben

Übersicht

Die Teilaufgaben sind im Bild 2-3 als Kästen dargestellt. Die grau unterlegten, rechteckigen Bereiche repräsentieren die Bausteine. Die Anordnung der Codebausteine von links nach rechts entspricht der Aufrufreihenfolge.

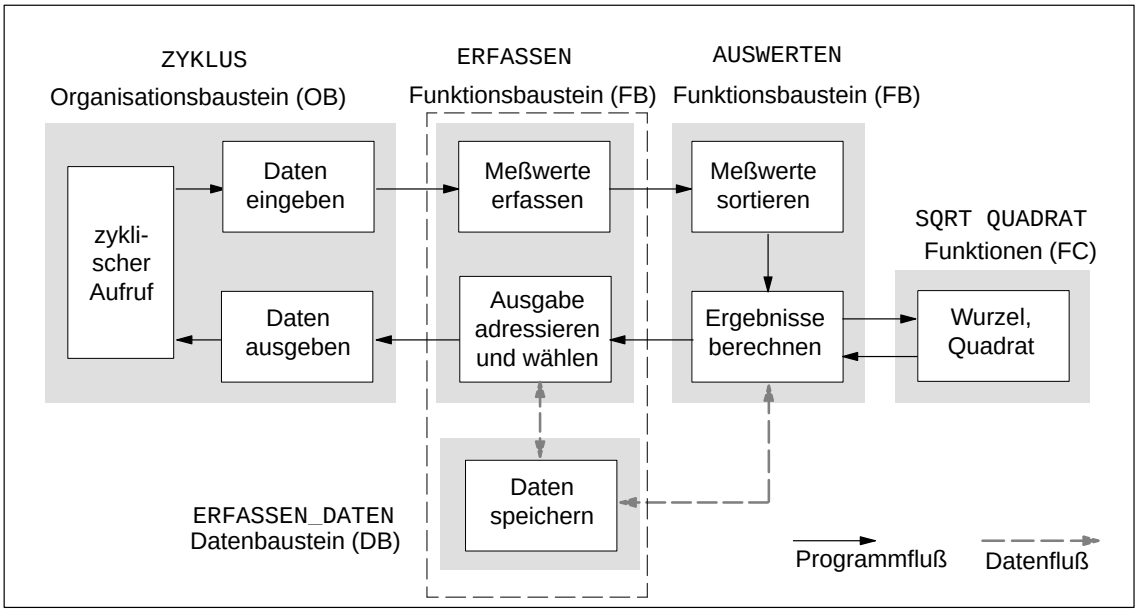


Bild 2-3 *Bildung von Bausteinen aus den Teilaufgaben*

2.3.2 Auswahl und Zuordnung der möglichen Bausteinarten

Übersicht

Die einzelnen Bausteine wurden nach folgenden Kriterien ausgewählt:

ZYKLUS

Anwenderprogramme können nur in einem OB angestoßen werden. Da die Meßwerte zyklisch erfaßt werden sollen, ist ein OB für *zyklischen Aufruf* (OB1) erforderlich. Ein Teil der Bearbeitung - *Daten eingeben* und *Daten ausgeben* - wird dabei im OB programmiert.

ERFASSEN

Für die Teilaufgabe *Meßwerte erfassen* ist ein Baustein mit Gedächtnis, also ein FB erforderlich, da bestimmte bausteinlokale Daten (z.B. der Ringpuffer) von einem Programmzyklus zum nächsten erhalten bleiben müssen. Der Ort zum *Daten speichern* (Gedächtnis) ist der Instanz-Datenbaustein ERFASSEN_DATEN.

Derselbe FB kann auch die Teilaufgabe *Ausgabe adressieren und wählen* übernehmen, da die benötigten Daten hier zur Verfügung stehen.

AUSWERTEN

Bei der Wahl der Bausteinart zur Lösung der Teilaufgaben *Meßwerte sortieren* und *Ergebnisse berechnen* ist zu berücksichtigen, daß ein Ausgabepuffer angelegt werden muß, der zu jedem Meßwert die Berechnungsergebnisse Wurzel und Quadrat enthält.

Deshalb kommt als Baustein nur ein FB in Frage. Da der FB von einem übergeordneten FB aufgerufen wird, benötigt er keinen eigenen DB. Seine Instanzdaten können im Instanz-Datenbaustein des aufrufenden FB abgelegt werden.

SQRT (Wurzel) und QUADRAT

Zur Lösung der Teilaufgabe *Wurzel oder Quadrat berechnen* eignet sich ein FC am besten, weil die Rückgabe des Ergebnisses als Funktionswert erfolgen kann. Außerdem werden zur Berechnung keine Daten benötigt, die länger als einen Zyklus der Programmbearbeitung erhalten bleiben müssen.

Zur Wurzelberechnung kann die SCL-Standardfunktion SQRT benutzt werden. Zur Quadratberechnung soll eine Funktion QUADRAT erstellt werden, die auch eine Grenzprüfung des Wertebereichs durchführt.

2.3.3 Festlegung der Schnittstellen zwischen den Bausteinen

Übersicht

Die Schnittstelle eines Bausteins zu einem anderen Baustein wird durch die Deklaration seiner **Formalparameter** festgelegt. Es gibt bei SCL die folgenden Möglichkeiten:

- Eingangsparameter: Deklaration mit VAR_INPUT
- Ausgangsparameter: Deklaration mit VAR_OUTPUT
- Durchgangsparameter: Deklaration mit VAR_IN_OUT

Wenn ein Baustein aufgerufen wird, werden ihm Eingabedaten als **Aktualparameter** übergeben. Nach der Rückkehr zum aufrufenden Baustein werden die Ausgabedaten zur Übernahme bereit gestellt. Ein FC kann sein Ergebnis als **Funktionswert** übergeben (Weitere Informationen hierzu finden Sie im Kapitel 16).

ERFASSEN

Der OB ZYKLUS hat selbst keine Formalparameter. Er ruft den FB ERFASSEN auf und übergibt ihm den Meßwert und die Steuerungsdaten für dessen Formalparameter (Tabelle 2-1):

Tabelle 2-1 Formalparameter von ERFASSEN

Parametername	Datentyp	Deklarationstyp	Beschreibung
messwert_ein	INT	VAR_INPUT	Meßwert
neuwert	BOOL	VAR_INPUT	Schalter, um Messwert in Ringpuffer zu übernehmen
neusort	BOOL	VAR_INPUT	Schalter, um Meßwerte zu sortieren und auszuwerten
funktions-wahl	BOOL	VAR_INPUT	Umschalter, um Wurzel oder Quadrat zu wählen
auswahl	WORD	VAR_INPUT	Codierung, um Ausgabewert auszuwählen
neuwahl	BOOL	VAR_INPUT	Schalter, um Codierung zu-übernehmen
ergebnis_aus	DWORD	VAR_OUTPUT	Ausgabe des berechneten Ergebnisses
messwert_aus	DWORD	VAR_OUTPUT	Ausgabe des zugehörigen Meßwerts

AUSWERTEN

Der FB ERFASSEN ruft den FB AUSWERTEN auf. Als gemeinsame Daten haben die beiden FBs das zu sortierende Meßwertefeld. Deshalb wird dieses als Durchgangsparameter deklariert. Für die Rechenergebnisse Wurzel und Quadrat wird ein strukturiertes Feld als Ausgangsparameter angelegt. Formalparameter siehe Tabelle 2-2:

Tabelle 2-2 Formalparameter von AUSWERTEN

Name	Datentyp	Deklarations-typ	Beschreibung
sortier-puffer	ARRAY[..] OF REAL	VAR_IN_OUT	Meßwertefeld, entspricht dem Ringpuffer
rechen-puffer	ARRAY[..] OF STRUCT	VAR_OUTPUT	Feld für Ergebnisse: Struktur mit den Komponenten ”wurzel” und ”quadrat” vom Typ INT

SQRT und QUADRAT

Die Funktionen werden von AUSWERTEN aufgerufen. Sie benötigen einen Eingabewert und liefern ihr Ergebnis als Funktionswert, siehe Tabelle 2-3.

Tabelle 2-3 Formalparameter und Funktionswerte von SQRT und QUADRAT

Name	Datentyp	Deklarations-typ	Beschreibung
wert	REAL	VAR_INPUT	Eingabe für SQRT
SQRT	REAL	Funktions-wert	Wurzel des Eingabewerts
wert	INT	VAR_INPUT	Eingabe für QUADRAT
QUADRAT	INT	Funktions-wert	Quadrat des Eingabewerts

2.3.4 Festlegung der Ein-/Ausgabe Schnittstelle

Übersicht

Bild 2-4 zeigt die Ein-/Ausgabeschnittstelle. Beachten Sie bitte, daß bei der byteweisen Ein-/Ausgabe oben das niederwertige Byte und unten das höherwertige Byte ist. Bei der wortweisen Ein-/Ausgabe dagegen ist es umgekehrt:

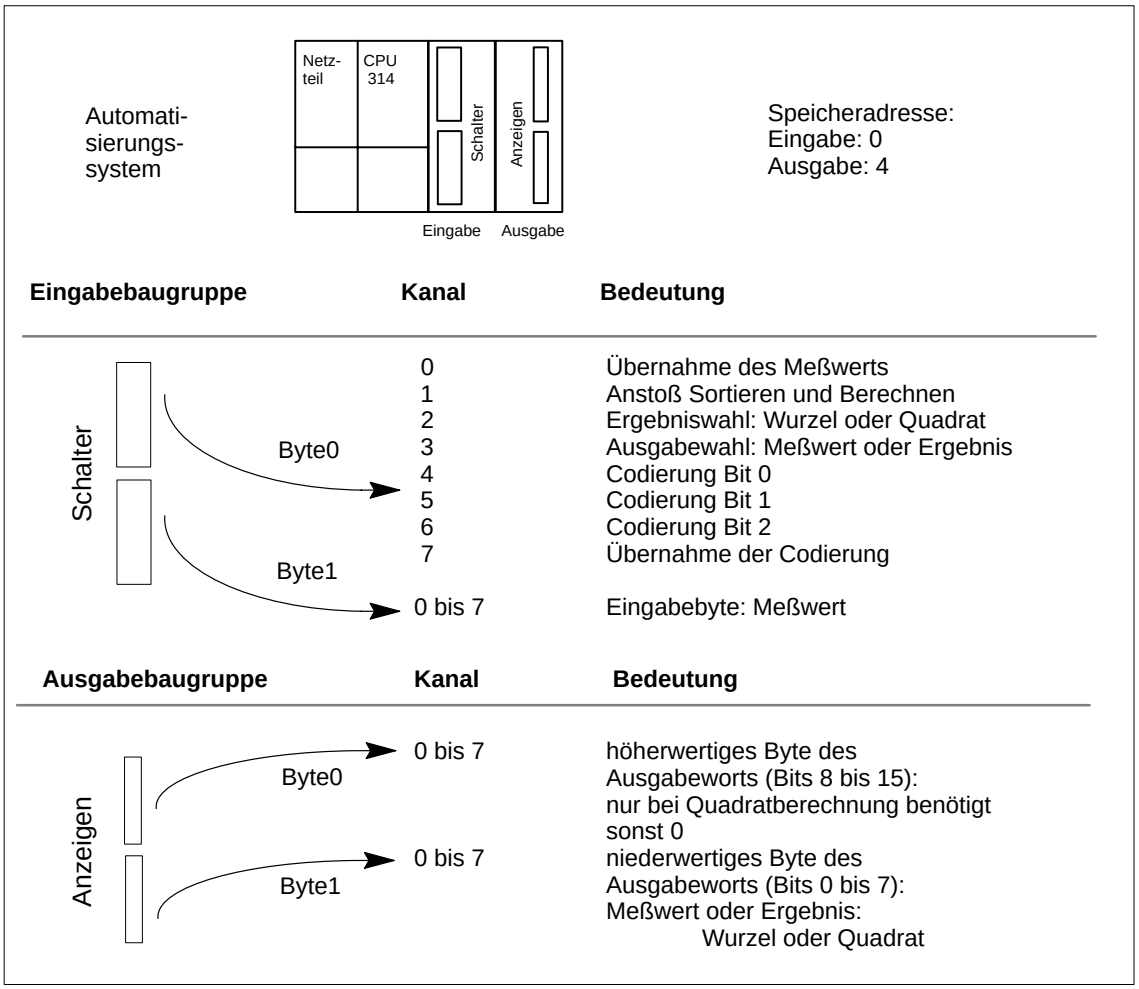


Bild 2-4 *Anzeigen und Bedienelemente*

2.3.5 Bausteine programmieren

Bausteine programmieren

Wenn die Schnittstellen definiert sind, können Sie die Bausteine unabhängig voneinander entwerfen. Das geschieht am besten ”top down”, also in der Reihenfolge ZYKLUS, ERFASSEN, AUSWERTEN und QUADRAT. In dieser Reihenfolge werden die Bausteine im folgenden auch beschrieben.

Beim Übersetzen der Bausteine müssen Sie berücksichtigen, daß ein Baustein vorhanden sein muß, bevor sie ihn benutzen, d.h. von einem anderen Baustein aus aufrufen. Das ergibt die notwendige Reihenfolge der Bausteine in der SCL-Quelle: QUADRAT, AUSWERTEN, ERFASSEN und ZYKLUS. (Informationen hierzu finden Sie in Kapitel 8).

Symbolische Programmierung

Die Verständlichkeit des Programms verbessert sich, wenn Sie **symbolische Namen** für Baugruppenadressen und Bausteine vergeben. Dazu müssen Sie Einträge in der Symboltabelle nach Bild 2-5 durchführen (siehe Kapitel 7). Die Namen können Sie entweder nach der Namenskonvention für BEZEICHNER oder nach der für Symbole (z.B. ”Eingang 0.0”) bilden, siehe Anhang A.

Einleitungs-kommentar mit Symboltabelle

In Bild 2-5 sehen Sie den Einleitungskommentar der SCL-Quelle. Er beschreibt die symbolischen Namen, die Sie in der Symboltabelle vereinbaren müssen, damit die Quelle fehlerfrei übersetzt werden kann:

```
(*#########################################################################################

SCL-Programm zur Erfassung und Weiterverarbeitung von Meßwerten:

      -  Über Eingang 0.0 (Eingabeschalter) wird ein Meßwert, der an der
         Eingangsbaugruppe anliegt, übernommen.
      -  Über verschiedene Schalter kann die Weiterverarbeitung der
         Meßwerte gesteuert werden.
      -  Alle Werte werden im Arbeitsbereich des Funktionsbausteins
         ERFASSEN, dem Instanzdatenbaustein ERFASSEN_DATEN, abgespeichert.

Das Programm ist symbolisch programmiert. Damit es übersetzt werden kann,muß die
Zuordnung der symbolischen Namen zu den Baugruppenadressen und den in der CPU ab-
laufenden Bausteinen vorhanden sein. Dazu wird folgendeSymboltabelle benötigt:

Eingabe              EB1      BYTE  // Meßwert
Eingang 0.0          E0.0     BOOL  // Eingabeschalter zur Übernahme des Meßwerts
Sortierschalter      E0.1     BOOL  // Anstoß zum Sortieren und Berechnen
Funktionsschalter    E0.2     BOOL  // Ergebniswahl Wurzel oder Quadrat
Ausgabeschalter      E0.3     BOOL  // Ausgabewahl Meßwert oder Ergebnis
Codierung            EW0      WORD  // Codierung,relevante Bits 12,13 und 14
Codierschalter       E0.7     BOOL  // Übernahme der Codierung
Ausgabe              AW4      INT   // Meßwert oder Ergebnis: Wurzel oder Quadrat

ERFASSEN             FB10     FB10  // Meßwerte erfassen,
                                    // Ausgabe adressieren und wählen
ERFASSEN_DATEN       DB10     FB10  // Instanzdatenbaustein zu ERFASSEN
AUSWERTEN            FB20     FB20  // Meßwerte auswerten, Ergebnisse berechnen
QUADRAT              FC41     FC41  // Funktion zur Quadratberechnung
ZYKLUS               OB1      OB1   // Zyklischer Aufruf und Ein-/Ausgabe

#########################################################################################*)
```

Bild 2-5 *Einleitungskommentar mit Symboltabelle*

2.4 Erstellen des Organisationsbausteins ZYKLUS

Lösungsweg

Ein OB1 wurde gewählt, weil er **zyklisch** aufgerufen wird. Mit ihm werden folgende Aufgaben für das Programm realisiert:

- Aufruf und Versorgung des Funktionsbausteins ERFASSEN mit Eingabedaten und Steuerungsdaten
- Übernahme der Ergebnisdaten des Funktionsbausteins ERFASSEN
- Ausgabe der Werte zur Anzeige

Am Anfang des Vereinbarungsteils steht das temporäre Datenfeld mit 20 byte "`systemdaten`" (siehe auch Kapitel 8).

```
ORGANIZATION_BLOCK ZYKLUS

(*****************************************************************************
      ZYKLUS entspricht OB1, d.h. er wird vom S7-System zyklisch aufgerufen.
      Teil 1   :  Aufruf des Funktionsbausteins und Übergabe der Eingabewerte
      Teil 2   :  Übernahme der Ausgabenwerte und Ausgabe mit Ausgabeumschaltung
*****************************************************************************)

VAR_TEMP
      systemdaten    :  ARRAY[0..20] OF BYTE;   // Bereich für OB1
END_VAR

BEGIN

(*    Teil 1   :  ************************************************************)

ERFASSEN.ERFASSEN_DATEN(
         messwert_ein   := WORD_TO_INT(Eingabe),
         neuwert        := "Eingang 0.0", //Eingabeschalter als Symbol
         neusort        := Sortierschalter,
         funktionswahl  := Funktionsschalter,
         neuwahl        := Codierschalter,
         auswahl        := Codierung);

(*    Teil 2   :  ************************************************************)

IF Ausgabeschalter THEN                              //Ausgabeumschaltung
      Ausgabe     := ERFASSEN_DATEN.ergebnis_aus;    //Wurzel oder Quadrat
ELSE
      Ausgabe     := ERFASSEN_DATEN.messwert_aus;    //Meßwert
END_IF;

END_ORGANIZATION_BLOCK
```

Bild 2-6 *Organisationsbaustein ZYKLUS (OB1)*

Datentyp-Konvertierungen

Der Meßwert liegt in der Eingabe als Typ BYTE vor. Er muß nach INT konvertiert werden: Dazu müssen Sie ihn von WORD nach INT konvertieren (die vorherige Konvertierung von BYTE nach WORD erfolgt implizit durch den Compiler, siehe Kapitel 18). Keine Konvertierung wird dagegen für die Ausgabe benötigt, da diese in der Symboltabelle als INT deklariert wurde, siehe Bild 2-5.

2.5 Erstellen des Funktionsbausteins ERFASSEN

Lösungsweg

Der Bausteintyp FB wurde gewählt, weil es Daten gibt, die von einem Progammzyklus zum nächsten gespeichert werden müssen. Dieses sind die statischen Variablen, die im Vereinbarungsblock ”VAR, END_VAR” deklariert werden, siehe Tabelle 2-4.

Statische Variablen sind lokale Variablen, deren Werte über alle Bausteindurchläufe hinweg erhalten bleiben. Sie dienen der Speicherung von Werten eines **Funktionsbausteins** und werden im Instanz-Datenbaustein abgelegt.

```
FUNCTION_BLOCK ERFASSEN
(*******************************************************************************
      Teil 1 :    Erfassung der Meßwerte
      Teil 2 :    Sortierung und Berechnung anstoßen
      Teil 3 :    Codierung auswerten und Ausgabe vorbereiten
*******************************************************************************)
```

Bild 2-7 *Bausteinkopf des Funktionsbausteins ERFASSEN*

Statische Variablen

Tabelle 2-4 Statische Variablen von ERFASSEN

Name	Datentyp	Deklarationstyp	Vorbesetzung	Beschreibung
messwerte	ARRAY [..] OF INT	VAR	8(0)	Ringpuffer für Meßwerte
ergebnis-puffer	ARRAY [..] OF STRUCT	VAR	–	Feld für Strukturen mit den Komponenten ”wurzel” und ”quadrat” vom Typ INT
zeiger	INT	VAR	0	Index für Ringpuffer, dort Eintrag des nächsten Meßwerts
altwert	BOOL	VAR	FALSE	Vorgängerwert für Meßwertübernahme mit ”neuwert”
altsort	BOOL	VAR	FALSE	Vorgängerwert für Sortieren mit ”neusort”
altwahl	BOOL	VAR	FALSE	Vorgängerwert für Übernahme der Codierung mit ”neuwahl”
adresse	INT	VAR	0	Adresse für Meßwert- oder Ergebnisausgabe
auswerte_instanz	AUSWERTEN, = FB 20	VAR	–	Lokale Instanz für den FB AUSWERTEN

Beachten Sie die Vorbesetzungswerte, die bei der Initialisierung des Bausteins (nach dem Laden in die CPU) in die Variablen eingetragen werden. Im Vereinbarungsblock "VAR, END_VAR" wird auch die lokale Instanz für den FB AUSWERTEN deklariert. Der Name wird später zum Aufruf und Zugriff auf die Ausgangsparameter verwendet. Als Datenspeicher wird die globale Instanz ERFASSEN_DATEN benutzt.

Vereinbarungsteil von ERFASSEN

Der Vereinbarungsteil in diesem Baustein besteht aus folgenden Vereinbarungsblöcken:

- Konstantendeklaration: Zwischen CONST und END_CONST.
- Eingangsparameter: Zwischen VAR_INPUT und END_VAR.
- Ausgangsparameter: Zwischen VAR_OUTPUT und END_VAR.
- statischen Variablen: Zwischen VAR und END_VAR. Hierzu zählt auch die Deklaration der lokalen Instanz für den Baustein AUSWERTEN.

```
CONST
      GRENZE         := 7;
      ANZAHL         := GRENZE + 1;
END_CONST

VAR_INPUT
      messwert_ein   :  INT;  //    Neuer Meßwert
      neuwert        :  BOOL; //    Meßwert in Ringpuffer "meßwerte" übernehmen
      neusort        :  BOOL; //    Meßwerte sortieren
      funktionswahl  :  BOOL; //    Wahl der Berechnungsfunktion Wurzel/Quadrat
      neuwahl        :  BOOL; //    Ausgabeadresse übernehmen
      auswahl        :  WORD; //    Ausgabeadresse
END_VAR

VAR_OUTPUT
      ergebnis_aus   :  INT;  //    berechneter Wert
      messwert_aus   :  INT;  //    zugehöriger Meßwert
END_VAR

VAR
      messwerte      :  ARRAY[0..GRENZE] OF INT := 8(0);
      ergebnispuffer :  ARRAY[0..GRENZE] OF
         STRUCT
            wurzel   :  INT;
            quadrat  :  INT;
         END_STRUCT;
      zeiger         :  INT   :=    0;
      altwert        :  BOOL  :=    TRUE;
      altsort        :  BOOL  :=    TRUE;
      altwahl        :  BOOL  :=    TRUE;
      adresse        :  INT   :=    0;    //Konvertierte Ausgabeadresse
      auswerte_instanz: AUSWERTEN;       //Lokale Instanz deklarieren
END_VAR
```

Bild 2-8 *Vereinbarungsteil des Funktionsbausteins ERFASSEN*

Entwurf Anweisungsteil

Er gliedert sich in 3 Teile:

Meßwerte erfassen

Wenn sich der Eingangsparameter ”neuwert” gegenüber ”altwert” verändert hat, wird ein neuer Meßwert in den Ringpuffer eingelesen.

Sortierung und Berechnung anstoßen

Sortierung und Berechnung werden durch Aufruf des Funktionsbausteins `AUSWERTEN` angestoßen, wenn sich der Eingangsparameter ”neusort” gegenüber ”altsort” verändert hat.

Codierung auswerten und Ausgabe vorbereiten

Die Codierung wird wortweise eingelesen: Nach der SIMATIC-Konvention bedeutet dies, daß die obere Schaltergruppe (Byte0) die höherwertigen 8 Bit des Eingabeworts enthält und die untere Schaltergruppe (Byte1) die niederwertigen. Das Bild 2-9 zeigt, wo die Schalter liegen, an denen Sie die Codierung einstellen:

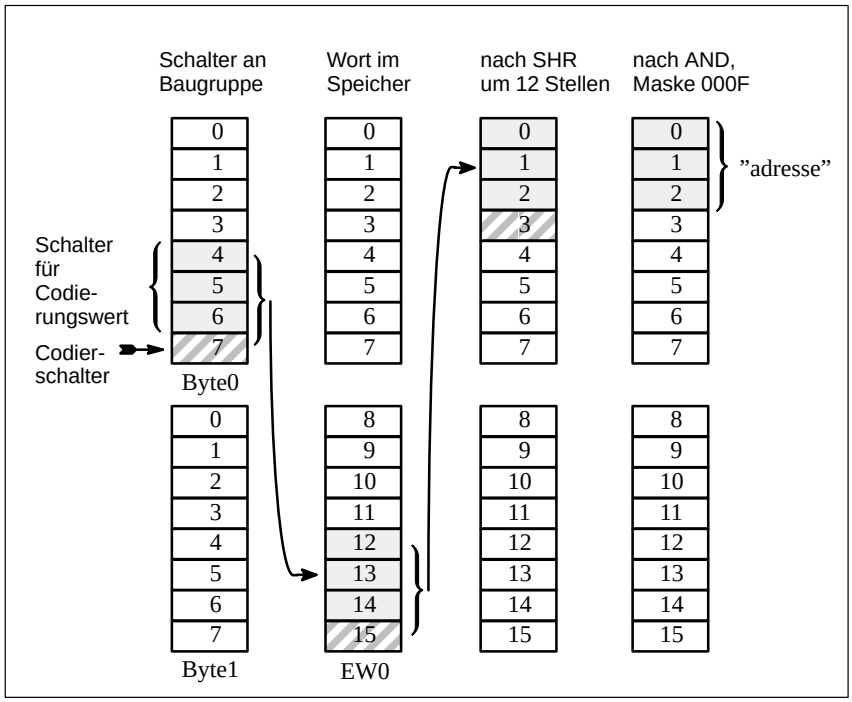


Bild 2-9 *Auswertung der Codierung*

Berechnung der Adresse

Bild 2-9 zeigt die Berechnung der Adresse: Das Eingabewort EW0 enthält im 12. bis 14. Bit die Codierung, die übernommen wird, wenn am Codierschalter (15. Bit) eine Flanke erkannt wird. Durch Verschieben nach rechts mit der Standardfunktion SHR und Ausblenden der relevanten Bits mit einer AND-Maske wird ”adresse” ermittelt.

Mit dieser Adresse werden die Feldelemente (Rechenergebnis und zugehöriger Meßwert) in die Ausgangsparameter geschrieben. Ob Wurzel oder Quadrat ausgegeben wird, hängt von ”funktionswahl” ab.

Eine Flanke am Codierschalter wird dadurch erkannt, daß sich ”neuwahl” gegenüber ”altwahl” verändert hat.

Flußdiagramm von ERFASSEN

Bild 2-10 stellt den Algorithmus in Form eines Flußdiagramms dar:

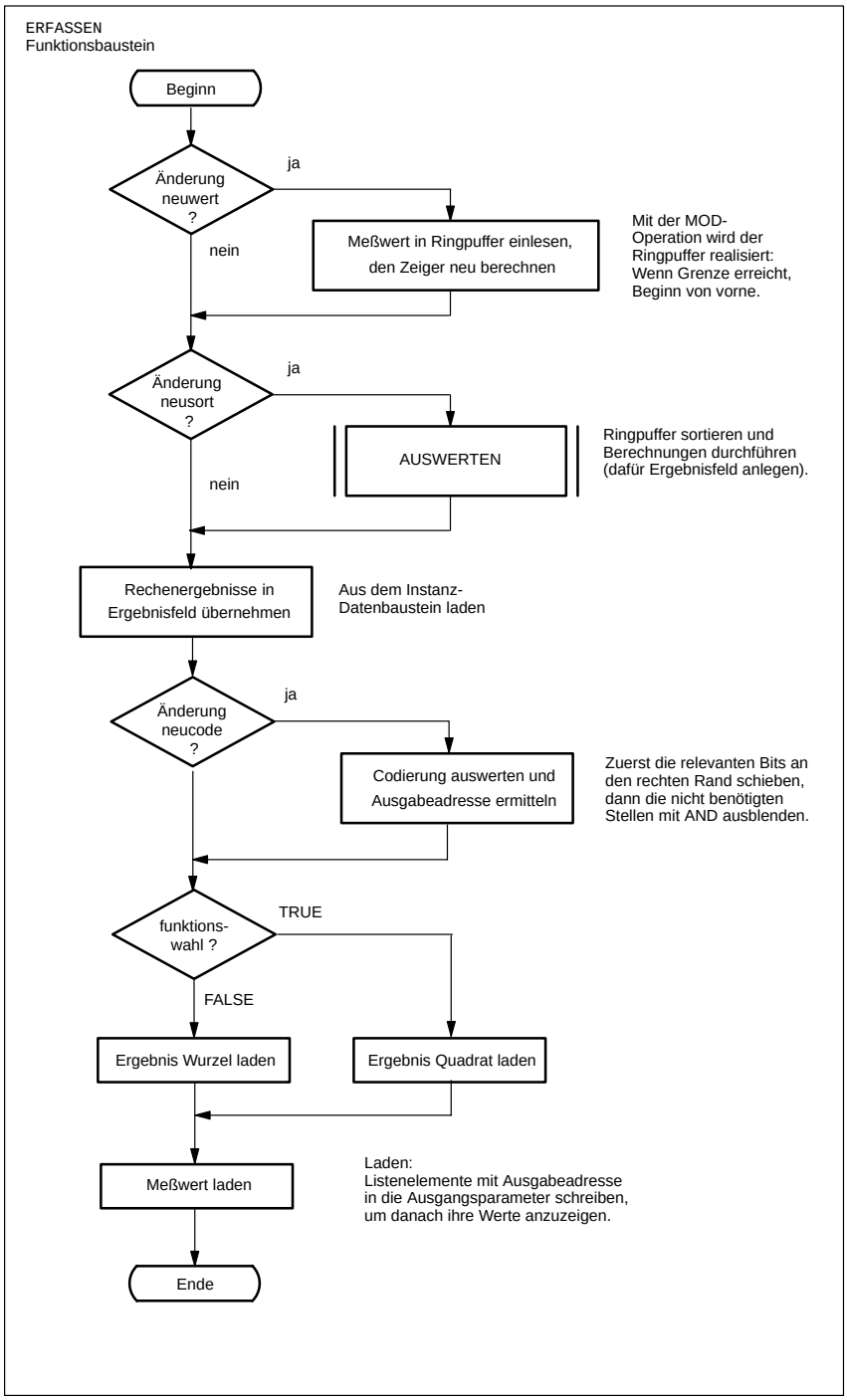


Bild 2-10 *Algorithmus zum* `Erfassen` *der Meßwerte*

Anweisungsteil von ERFASSEN

Bild 2-11 zeigt die Formulierung des in Bild 2-10 dargestellten Flußdiagrammes in SCL, d.h. den **Anweisungsteil** des Codebausteins.

```
BEGIN

(* Teil 1 :    Erfassung der Meßwerte ***************************************
               Bei Änderung von ”neuwert” erfolgt Eingabe des Meßwerts.
               Mit Operation MOD wird ein Ringpuffer für Meßwerte realisiert.*)

IF neuwert <> altwert THEN
      zeiger            :=    zeiger MOD ANZAHL;
      messwerte[zeiger] :=    messwert_ein;
      zeiger            :=    zeiger + 1;
END_IF;
altwert  :=    neuwert;

(* Teil 2 :    Sortierung und Berechnung anstoßen ******************************
               Bei Änderung von ”neusort” Anstoß zum Sortieren des Ringpuffers
               und zur Ausführung von Berechnungen mit den Meßwerten. Ergebnisse
               werden in einem neuen Feld, ”rechenpuffer”, gespeichert. *)

IF neusort <> altsort THEN
      zeiger := 0;   //Ringpufferzeiger rücksetzen
      auswerte_instanz( sortierpuffer := messwerte); //AUSWERTEN aufrufen
END_IF;
altsort  :=    neusort;

ergebnispuffer :=    auswerte_instanz.rechenpuffer; //Quadrat und Wurzel

(* Teil 3 :    Codierung auswerten und Ausgabe vorbereiten: *********************
               Bei Änderung von ”neuwahl” wird die Codierung für die Adressierung
               des Feldelements für die Ausgabe neu ermittelt: Die relevanten
               Bits von ”auswahl” werden ausgeblendet und in Integer gewandelt.
               Je nach Schalterstellung von ”funktionswahl” wird ”wurzel” oder
               ”quadrat” für die Ausgabe bereitgestellt. *)

IF neuwahl <> altwahl THEN
      adresse  := WORD_TO_INT(SHR(IN := auswahl, N := 12) AND 16#0007);
END_IF;
altwahl  :=    neuwahl;

IF funktionswahl THEN
      ergebnis_aus   :=    ergebnispuffer[adresse].quadrat;
ELSE
      ergebnis_aus   :=    ergebnispuffer[adresse].wurzel;
END_IF;


messwert_aus   :=    messwerte[adresse]; //Meßwertanzeige

END_FUNCTION_BLOCK
```

Bild 2-11 *Anweisungsteil des Funktionsbausteins ERFASSEN*

2.6 Erstellen des Funktionsbausteins AUSWERTEN

Vereinbarungsteil von AUSWERTEN

Der Vereinbarungsteil dieses Bausteins besteht aus folgenden Teilen:

- Konstantendeklaration: Zwischen CONST und END_CONST
- Durchgangsparameter: Zwischen VAR_IN_OUT und END_VAR,
- Ausgangsparameter: Zwischen VAR_OUTPUT und END_VAR
- Deklaration der temporären Variablen: Zwischen VAR_TEMP und END_VAR

```
FUNCTION_BLOCK AUSWERTEN

(*******************************************************************************
      Teil 1 :    Ringpuffer mit Meßwerten sortieren
      Teil 2 :    Anstoß Berechnungen der Ergebnisse
*******************************************************************************)
```

Bild 2-12 *Bausteinkopf des Funktionsbausteins AUSWERTEN*

```
CONST
      GRENZE         := 7;
END_CONST

VAR_IN_OUT
      sortierpuffer  :  ARRAY[0..GRENZE] OF INT;
END_VAR

VAR_OUTPUT
      rechenpuffer   :  ARRAY[0..GRENZE] OF
         STRUCT
            wurzel   :  INT;
            quadrat  :  INT;
         END_STRUCT;
END_VAR

VAR_TEMP
      tauschen       :  BOOL;
      index, hilf    :  INT;
      wertr, ergebnr :  REAL;
END_VAR
```

Bild 2-13 *Vereinbarungsteil des Funktionsbausteins AUSWERTEN*

Fuktionsablauf

Der Durchgangsparameter ”sortierpuffer” wird mit dem Ringpuffer ”messwerte” verknüpft, d.h. der ursprüngliche Pufferinhalt wird mit den sortierten Meßwerten überschrieben.

Für die Rechenergebnisse wird das neue Feld ”rechenpuffer” als Ausgangsparameter angelegt. Seine Elemente sind so strukturiert, daß sie zu jedem Meßwert die Wurzel und das Quadrat enthalten.

Im Bild 2-14 sehen Sie den Zusammenhang zwischen den beschriebenen Feldern:

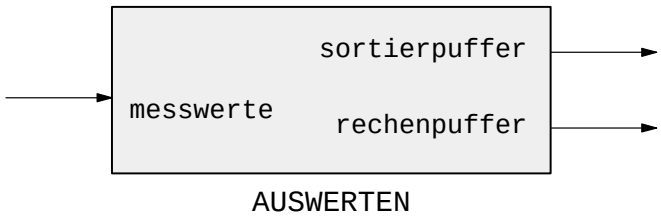


Bild 2-14 *Schnittstelle des FB AUSWERTEN*

Diese Schnittstelle zeigt den Kern des Datenaustauschs zur Verarbeitung der Meßwerte. Die Werte werden im Instanz-Datenbaustein ERFASSEN_DATEN gespeichert, da im aufrufenden FB ERFASSEN eine lokale Instanz für den FB AUSWERTEN angelegt wurde

Entwurf Anweisungsteil

Zuerst werden die Meßwerte im Ringpuffer sortiert und danach die Berechnungen durchgeführt:

- Methode des Algorithmus zum Sortieren

Hier wird die Methode des permanenten Tauschens von Werten zur Sortierung des Meßwertepuffers verwendet, d.h. zwei aufeinanderfolgende Werte werden miteinander verglichen und solange getauscht bis die gewünschte Sortierreihenfolge erreicht ist. Der verwendete Puffer ist der Durchgangsparameter ”sortierpuffer”.

- Anstoß der Berechnung

Wenn die Sortierung abgeschlossen ist, wird eine Schleife zur Berechnung durchlaufen, in der die Funktionen QUADRAT zur Quadrierung und SQRT zur Wurzelberechnung aufgerufen werden. Ihre Ergebnisse werden in dem strukturierten Feld ”rechenpuffer” gespeichert.

Flußdiagramm von AUSWERTEN

Bild 2-15 stellt den Algorithmus in Form eines Flußdiagrammes dar:

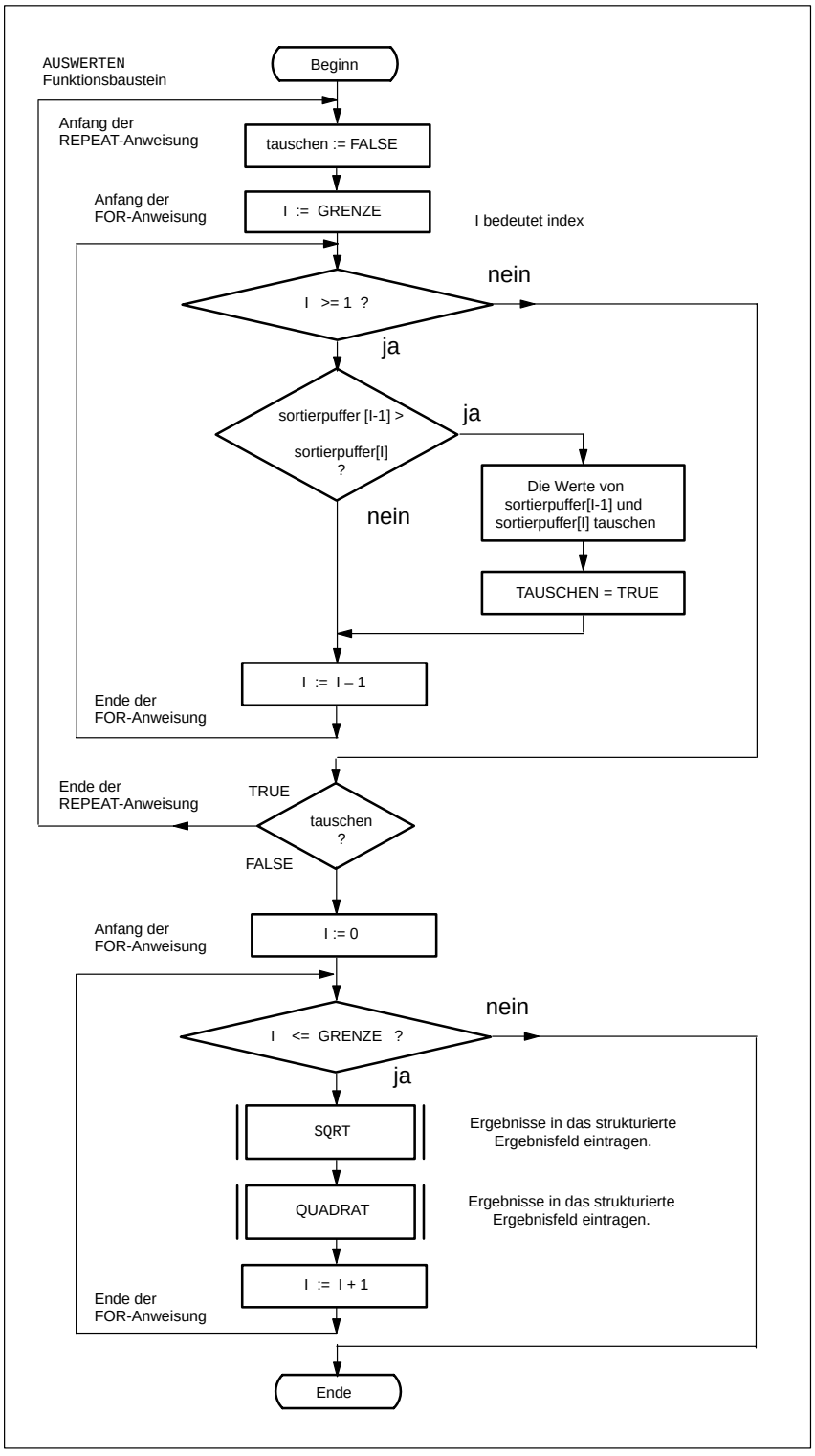


Bild 2-15 *Algorithmus zum Auswerten der Meßwerte*

Anweisungsteil von Auswerten

Bild 2-16 zeigt die Formulierung des in Bild 2-15 dargestellten Flußdiagrammes in SCL, d.h. den Anweisungsteil des Codebausteins:

```
BEGIN

(* Teil 1   Sortierung : ************************************************
            nach dem "Bubble Sort" Verfahren: Werte solange
            paarweise tauschen, bis Meßwertpuffer sortiert ist. *)

REPEAT
      tauschen    := FALSE;

      FOR index   := GRENZE TO 1 BY –1 DO
         IF sortierpuffer[index–1] > sortierpuffer[index] THEN
            hilf                   :=    sortierpuffer[index];
            sortierpuffer[index]   :=    sortierpuffer[index–1];
            sortierpuffer[index–1] :=    hilf;
            tauschen               :=    TRUE;
         END_IF;
      END_FOR;

UNTIL NOT tauschen
END_REPEAT;


(* Teil 2   Berechnung : ************************************************
            Wurzelberechnung mit Standardfunktion SQRT und
            Quadrierung mit Funktion QUADRAT durchführen. *)

FOR index   := 0 TO GRENZE BY 1 DO
      wertr    := INT_TO_REAL(sortierpuffer[index]);
      ergebnr  :=        SQRT(wertr);
      rechenpuffer[index].wurzel    :=    REAL_TO_INT(ergebnr);
      rechenpuffer[index].quadrat   :=    QUADRAT(sortierpuffer[index]);
END_FOR;

END_FUNCTION_BLOCK
```

Bild 2-16 *Anweisungsteil des Funktionsbausteins AUSWERTEN*

2.7 Erstellen der Funktion QUADRAT

Entwurf Anweisungsteil

Zuerst wird geprüft, ob der Eingabewert die Grenze überschreitet, bei der das Ergebnis über den Zahlenbereich für Integer hinaus geht. Ist dies der Fall, wird der Maximalwert für Integer eingetragen. Ansonsten wird die Quadrierung durchgeführt. Das Ergebnis wird als Funktionswert übergeben.

```
FUNCTION QUADRAT : INT

(*****************************************************************************
Diese Funktion liefert als Funktionswert das Quadrat des Eingangswertes
oder bei Überlauf den maximalen Wert, der mit Integer darstellbar ist.
*****************************************************************************)

VAR_INPUT
      wert    :   INT;
END_VAR

BEGIN
IF wert <= 181 THEN
      QUADRAT     :=    wert * wert;      // Berechnung des Funktionswerts
ELSE
      QUADRAT     :=    32_767;           // bei Überlauf Maximalwert setzen
END_IF;

END_FUNCTION
```

Bild 2-17 *Funktion QUADRAT*

2.8 Testdaten

Voraussetzungen

Für den Test benötigen Sie eine Eingabebaugruppe mit der Adresse 0 und eine Ausgabebaugruppe mit der Adresse 4 (siehe Bild 2-4).

Stellen Sie vor dem Test die oberen 8 Schalter der Eingabebaugruppe nach links (”0”) und die unteren 8 Schalter der Eingabebaugruppe nach rechts (”1”).

Laden Sie die Bausteine neu in die CPU, da auch die Initialwerte der Variablen getestet werden.

Testschritte

Führen Sie nun die Testschritte nach Tabelle 2-5 durch.

Tabelle 2-5 Testschritte

Test	Aktion	Folge
1	Stellen Sie die Codierung ”111” ein (E0.4, E0.5 und E0.6) und übernehmen Sie diese mit dem Codierschalter (E0.7).	Alle Ausgänge der Ausgabe (niederwertiges Byte) werden angesteuert und die Anzeigen leuchten.
2	Zeigen Sie die zugehörige Wurzel an, indem Sie den Ausgabeschalter (E0.3) auf ”1”schalten.	Die Anzeigen am Ausgang entsprechen dem Binärwert ”10000” (=16).
3	Zeigen Sie das zugehörige Quadrat an, indem Sie den Funktionsschalter (E0.2) auf ”1”schalten.	Am Ausgang leuchten 15 Anzeigen. Das bedeutet, daß Überlauf auftritt, da 255 x 255 einen zu großen Wert für den Integerbereich ergibt.
4a	Stellen Sie den Ausgabeschalter (E0.3) wieder auf ”0” zurück.	Der Meßwert wird wieder angezeigt: Alle Anzeigen an den Ausgängen des niederwertigen Ausgabebytes sind gesetzt.
4b	Stellen Sie als neuen Meßwert an der Eingabe den Wert 3, d.h. Binärwert ”11”, ein.	Die Ausgabe verändert sich noch nicht.
5a	Einlesen des Meßwerts beobachten: Stellen Sie die Codierung auf ”000” ein und übernehmen Sie diese mit dem Codierschalter (E0.7), so daß Sie später die Werteingabe beobachten können.	Am Ausgang wird 0 angezeigt, d.h. keine der Anzeigen leuchtet.
5b	Schalten Sie den Eingabeschalter ”Eingang 0.0” (E0.0) um. Dadurch wird der im 4. Testschritt eingestellte Wert eingelesen.	Am Ausgang wird der Meßwert 3 , Binärwert ”11”, angezeigt.
6	Stoßen Sie Sortieren und Berechnen an, indem Sie den Sortierschalter (E0.1) umschalten.	Am Ausgang erscheint wieder 0, da durch den Sortiervorgang der Meßwert weiter nach oben im Feld verschoben wurde.
7	Meßwert nach der Sortierung anzeigen: Stellen Sie die Codierung ”110” ein (E0.6 = 1, E0.5 = 1, E0.4 = 0 von EB0, entspricht Bit 14, Bit 13, Bit12 von EW0) und übernehmen Sie diese durch Umschalten des Codierschalters.	Am Ausgang wird nun wieder der Meßwert ”11” angezeigt, da er der zweithöchste Wert im Feld ist.
8a	Die zugehörigen Ergebnisse anzeigen: Durch Umschalten des Ausgabeschalters (E0.3) wird das Quadrat zum Meßwert aus dem 7. Testschritt angezeigt.	Der Ausgangswert 9 bzw. Binärwert ”1001” wird angezeigt.
8b	Durch Umschalten des Funktionsschalters (E0.2) erhalten Sie auch die Wurzel.	Der Ausgangswert 2 bzw. Binärwert ”10” wird angezeigt.

Zusatztest

Die Tabellen 2-6 und 2-7 erläutern die Schalter an der Eingabebaugruppe sowie Testmuster für Wurzel und Quadrat und helfen Ihnen eigene Testschritte zu definieren:

- Die Eingabe erfolgt über Schalter: Über die oberen 8 Schalter können Sie das Programm steuern, über die unteren 8 können Sie Meßwert einstellen.
- Die Ausgabe erfolgt über Anzeigen: An der oberen Gruppe erscheint das höherwertige Ausgabebyte, an der unteren Gruppe das niederwertige.

Tabelle 2-6 Bedienschalter

Bedien-schalter	Name	Erklärung
Kanal 0	Eingabeschalter	Umschalten zur Meßwertübernahme
Kanal 1	Sortierschalter	Umschalten zur Sortierung/Auswertung
Kanal 2	Funktionsschalter	Schalter nach links (”0”): Wurzel, Schalter nach rechts (”1”): Quadrat
Kanal 3	Ausgabeschalter	Schalter nach links (”0”): Meßwert, Schalter nach rechts (”1”): Ergebnis
Kanal 4	Codierung	Ausgabeadresse Bit 0
Kanal 5	Codierung	Ausgabeadresse Bit 1
Kanal 6	Codierung	Ausgabeadresse Bit 2
Kanal 7	Codierschalter	Umschalten zur Codierungsübernahme

Die Tabelle 2-7 enthält 8 beispielhafte Meßwerte in bereits sortierter Reihenfolge.

Geben Sie die Werte in beliebiger Reihenfolge ein. Stellen Sie dazu die jeweilige Bitkombination ein und übernehmen Sie den Wert, indem Sie den Eingabeschalter umschalten. Nachdem alle Werte eingegeben sind, stoßen Sie durch Umschalten des Sortierschalters die Sortierung und Auswertung an. Danach können Sie die sortierten Meßwerte oder die Ergebnisse – Wurzel oder Quadrat – anschauen.

Tabelle 2-7 Testmuster für Wurzel und Quadrat

Meßwert	Wurzel	Quadrat
0000 0001 = 1	0, 0000 0001 = 1	0000 0000, 0000 0001 = 1
0000 0011 = 3	0, 0000 0010 = 2	0000 0000, 0000 1001 = 9
0000 0111 = 7	0, 0000 0011 = 3	0000 0000, 0011 0001 = 49
0000 1111 = 15	0, 0000 0100 = 4	0000 0000, 1110 0001 = 225
0001 1111 = 31	0, 0000 0110 = 6	0000 0011, 1100 0001 = 961
0011 1111 = 63	0, 0000 1000 = 8	0000 1111, 1000 0001 = 3969
0111 1111 = 127	0, 0000 1011 = 11	0011 1111, 0000 0001 = 16129
1111 1111 = 255	0, 0001 0000 = 16	0111 111, 1111 1111 = Überlaufanzeige!

Teil 2: Bedienen und testen

Installieren der SCL-Software

3

Übersicht

Ein Setup-Programm ermöglicht Ihnen das menügeführte Installieren der SCL-Software. Das Setup-Programm muß bei der unter Windows 95 üblichen Standardprozedur zur Installation der Software aufgerufen werden.

Installations-voraussetzungen

Um die SCL-Software installieren zu können, benötigen Sie:

- ein Programmiergerät oder PC mit bereits installiertem STEP 7-Basispaket
  - Prozessor 80486 (oder höher) und
  - RAM-Speicherausbau 16 MB
- einen Farbmonitor, Tastatur und Maus, die von Microsoft Windows 95 unterstützt werden
- eine Festplatte mit einem freiem Speicherplatz von 78 MB (10 MB für Anwenderdaten, 60 MB für Swap-Dateien, 8 MB für das Optionspaket SCL)
- mindestens 1 MB freien Speicherplatz auf dem Laufwerk C: für das Setup (Setup-Dateien werden nach Abschluß der Installation gelöscht)
- das Betriebssystem Windows 95.
- eine MPI-Schnittstelle zwischen Programmiergerät oder PC und AS:
  - Entweder ein PC/MPI-Kabel, das an die Kommunikationsschnittstelle Ihres Geräts angeschlossen wird, oder
  - eine MPI-Baugruppe, die in Ihrem Gerät installiert wird. Bei einigen Programmiergeräten ist die MPI-Schnittstelle bereits vorhanden.

Kapitelübersicht

3.1 Autorisierung / Nutzungsberechtigung

Einleitung

Für die Nutzung des Softwarepaketes SCL wird eine produktspezifische Autorisierung (Nutzungsberechtigung) benötigt. Die so geschützte Software ist nur benutzbar, wenn auf der Festplatte des betreffenden PG/PC die für das Programm oder Softwarepaket erforderliche Autorisierung erkannt wird.

Autorisierungs-diskette

Für die Autorisierung benötigen Sie die zum Lieferumfang gehörende kopiergeschützte Autorisierungsdiskette. Sie enthält die Autorisierung und das zum Anzeigen, Installieren und Deinstallieren der Autorisierung erforderliche Programm AUTHORS.

Die Anzahl der möglichen Autorisierungen ist durch einen Autorisierungszähler auf der Autorisierungsdiskette festgelegt. Bei jeder Autorisierung wird der Zähler um 1 erniedrigt. Wenn er den Wert Null erreicht hat, ist mit dieser Diskette keine weitere Autorisierung mehr möglich.

Weitere Hinweise und Regeln für den Umgang mit Autorisierungen finden Sie im Benutzerhandbuch **/231/**.

Vorsicht

Beachten Sie die Hinweise in der Datei LIESMICH.TXT auf der Autorisierungsdiskette. Bei Nichtbeachtung besteht die Gefahr, daß die Autorisierung unwiderruflich verloren geht.

Autorisierung bei der Erstinstallation vornehmen

Sie sollten die Autorisierung vornehmen, wenn Sie im Rahmen der Erstinstallation durch eine entsprechende Meldung dazu aufgefordert werden. Gehen Sie so vor:

1. Legen Sie die Autorisierungsdiskette ein, wenn die entsprechende Aufforderung angezeigt wird.
2. Quittieren Sie anschließend die Aufforderung.

Die Berechtigung wird auf ein physikalisches Laufwerk übertragen. (d.h. Ihr Rechner ”merkt sich”, daß Sie eine Berechtigung haben).

Autorisierung später durchführen

Wenn Sie die SCL-Software starten und keine Autorisierung vorhanden ist, so erhalten Sie eine entsprechende Meldung. Um die Autorisierung nachträglich vorzunehmen, rufen Sie das Programm AUTHORS auf der Autorisierungsdiskette auf. Damit lassen sich Autorisierungen anzeigen, installieren und deinstallieren. Das Programm ist menügeführt.

Hinweis

Geben Sie bei der Installation der Autorisierung für SCL als Ziellaufwerk immer Laufwerk C: an.

Autorisierung deinstallieren

Wird eine erneute Autorisierung notwendig, z.B. wenn Sie das Laufwerk, auf dem sich die Berechtigung befindet, neu formatieren wollen, so müssen Sie vorher die Berechtigung ”retten”. Dazu benötigen Sie die Original-Autorisierungsdiskette.

Gehen Sie folgendermaßen vor, um die Autorisierung wieder zurück auf die Autorisierungsdiskette zu übertragen:

1. Legen Sie die Original-Autorisierungsdiskette in Ihr 3,5”-Laufwerk ein.
2. Rufen Sie das Programm AUTHORS.EXE von der Autorisierungsdiskette auf.
3. Wählen Sie den Menübefehl **Autorisierung ▸ Deinstallieren**.
4. Geben Sie im anschließend aufgeblendeten Dialogfeld das Laufwerk an, auf dem sich die Autorisierung befindet, und quittieren Sie den Dialog. Eine Liste mit allen Autorisierungen auf dem betreffenden Laufwerk wird angezeigt.
5. Markieren Sie die Autorisierung, die Sie deinstallieren wollen, und quittieren Sie den Dialog. Wenn der Vorgang ohne Fehler abgeschlossen worden ist, erhalten Sie die Meldung ”**Autorisierung** <Name> **erfolgreich von Laufwerk** <X:> **entfernt.**”
6. Quittieren Sie die Meldung.

   Danach wird wieder das Dialogfeld mit der Liste der restlichen Autorisierungen auf dem Laufwerk angezeigt. Schließen Sie das Dialogfeld, wenn Sie keine weiteren Autorisierungen deinstallieren wollen.

Sie können diese Diskette dann erneut zum Autorisieren benutzen.

Wenn Ihre Festplatte defekt ist ...

Tritt auf Ihrer Festplatte ein Defekt auf, bevor Sie die Berechtigung retten konnten, wenden Sie sich bitte an Ihre zuständige SIEMENS-Vertretung.

3.2 Installieren / Deinstallieren der SCL-Software

Übersicht

SCL enthält ein Setup-Programm, das die Installation automatisch durchführt. Eingabeaufforderungen auf dem Bildschirm führen Sie Schritt für Schritt durch den gesamten Installationsvorgang.

Vorbereitungen

Bevor Sie mit der Installation beginnen können, muß Windows 95 gestartet und das STEP 7-Basispaket geladen sein.

Installations-programm starten

Gehen Sie so vor:

1. Starten Sie unter Windows 95 den Dialog zur Installation von Software durch Doppelklick auf das Symbol ”Software” in ”Systemsteuerung”.
2. Klicken Sie auf ”Installieren”.
3. Legen Sie den Datenträger (Diskette 1) oder die CD-ROM ein und klicken Sie auf ”Weiter”. Windows 95 sucht nun selbständig nach dem Installationsprogramm SETUP.EXE.
4. Befolgen Sie Schritt für Schritt die Anweisungen, die Ihnen das Installationsprogramm anzeigt.

Das Programm führt Sie schrittweise durch den Installationsprozeß. Sie können jeweils zum nächsten oder vorherigen Schritt weiterschalten.

Falls schon eine SCL-Version installiert ist

Wenn das Installationsprogramm feststellt, daß sich bereits eine SCL-Installation auf dem Erstellsystem befindet, wird eine entsprechende Meldung angezeigt, und Sie haben folgende Wahlmöglichkeiten:

- Installation abbrechen (um danach die alte SCL-Version unter Windows 95 zu deinstallieren und anschließend die Installation erneut zu starten) oder
- Installation fortsetzen und damit die alte Version durch die neue Version überschreiben.

Sie sollten vor einer Installation auf jeden Fall eine eventuell vorhandene ältere Version deinstallieren. Das einfache Überschreiben einer älteren Version hat den Nachteil, daß bei einem anschließenden Deinstallieren eventuell noch vorhandene Teile aus einer älteren Installation nicht entfernt werden.

Während des Installationsvorgangs werden Ihnen in Dialogfeldern Fragen angezeigt oder Optionen zur Auswahl angeboten. Lesen Sie bitte die folgenden Hinweise, um die Dialoge schneller und leichter beantworten zu können.

Deinstallieren

Benutzen Sie das unter Windows 95 übliche Verfahren zur Deinstallation:

1. Starten Sie unter Windows 95 den Dialog zu Installation von Software durch Doppelklick auf das Symbol ”Software” in ”Systemsteuerung”.
2. Markieren Sie den STEP 7-Eintrag in der angezeigten Liste der installierten Software und betätigen Sie die Schaltfläche zum Entfernen der Software.
3. Falls Dialogfelder ”Freigegebene Datei entfernen” erscheinen, so klicken Sie im Zweifelsfall auf die Schaltfläche ”Nein”.

Zum Installations-umfang

Alle Sprachen der Benutzeroberfläche und alle Beispiele benötigen ca. 8 MB Speicherplatz.

Zur Autorisierung

Bei der Installation wird überprüft, ob eine Autorisierung auf der Festplatte vorhanden ist. Wird keine Autorisierung erkannt, so erscheint ein Hinweis, daß die Software nur mit Autorisierung benutzt werden kann. Wenn Sie es wünschen, können Sie gleich die Autorisierung vornehmen oder aber die Installation fortsetzen und die Autorisierung zu einem späteren Zeitpunkt nachholen.

Im erstgenannten Fall legen Sie nach Aufforderung die Autorisierungs-diskette ein und quittieren Sie. Hinweise zur Autorisierung finden Sie in Abschnitt 3.1.

Zum Abschluß der Installation

Wenn die Installation erfolgreich war, wird dies durch eine entsprechende Meldung auf dem Bildschirm angezeigt.

Fehler während der Installation

Folgende Fehler führen zum Abbruch der Installation:

- Wenn sofort nach dem Start von Setup ein Initialisierungsfehler auftritt, so wurde höchstwahrscheinlich das Programm SETUP.EXE nicht unter Windows 95 gestartet.
- Speicherplatz reicht nicht aus: Sie benötigen mindestens 8MB freien Speicherplatz auf Ihrer Festplatte.
- Defekte Diskette: Wenn Sie feststellen, daß eine Diskette defekt ist, wenden Sie sich bitte an Ihre Siemens-Vertretung.
- Bedienerfehler: Beginnen Sie die Installation erneut und beachten Sie die Anweisungen sorgfältig.

Bedienen von SCL

4

Übersicht

Dieses Kapitel macht Sie mit der allgemeinen Bedienung von SCL bekannt. Es informiert Sie über die Bedienoberfläche des SCL-Editors.

Kapitelübersicht

4.1 Starten der SCL-Software

Starten von der Windows-Oberfläche

Nachdem Sie die SCL-Software auf Ihrem PG installiert haben, können Sie SCL über die Schaltfläche ”Start” auf der Task-Leiste in Windows 95 starten (Eintrag unter ”SIMATIC/STEP 7”).

Starten aus dem SIMATIC Manager

Am schnellsten starten Sie SCL, indem Sie im SIMATIC Manager den Mauszeiger auf einer SCL-Quelle positionieren und doppelklicken. Informationen hierzu finden Sie im Benutzerhandbuch **/231/**.

Bild 4-1 zeigt das SCL-Fenster nach dem Aufrufen der Software.

Bild 4-1 *SCL-Fenster*

Hinweis

Nähere Informationen über Standardbedienungen und -optionen von Windows 95 finden Sie in Ihrem Windows-Benutzerhandbuch oder online im Windows 95-Lernprogramm.

4.2 Anpassen der Bedienoberfläche

Übersicht

Die SCL-Fenster bestehen, wie andere STEP 7-Fenster, aus den folgenden Standardkomponenten, siehe Bild 4-2:

- **Titelzeile:**
  enthält den Fenstertitel und Symbole für die Fenstersteuerung.
- **Menüleiste:**
  enthält alle Menüs, die im Fenster zur Verfügung stehen.
- **Funktionsleiste:**
  enthält Symbole, über die Sie häufig verwendete Befehle schnell ausführen können.
- **Arbeitsbereich:**
  enthält die verschiedenen Fenster, in denen Sie den Programmtext editieren oder Compiler- und Testinformationen ablesen können.
- **Statuszeile:**
  zeigt den Status und weitere Informationen zum ausgewählten Objekt an.

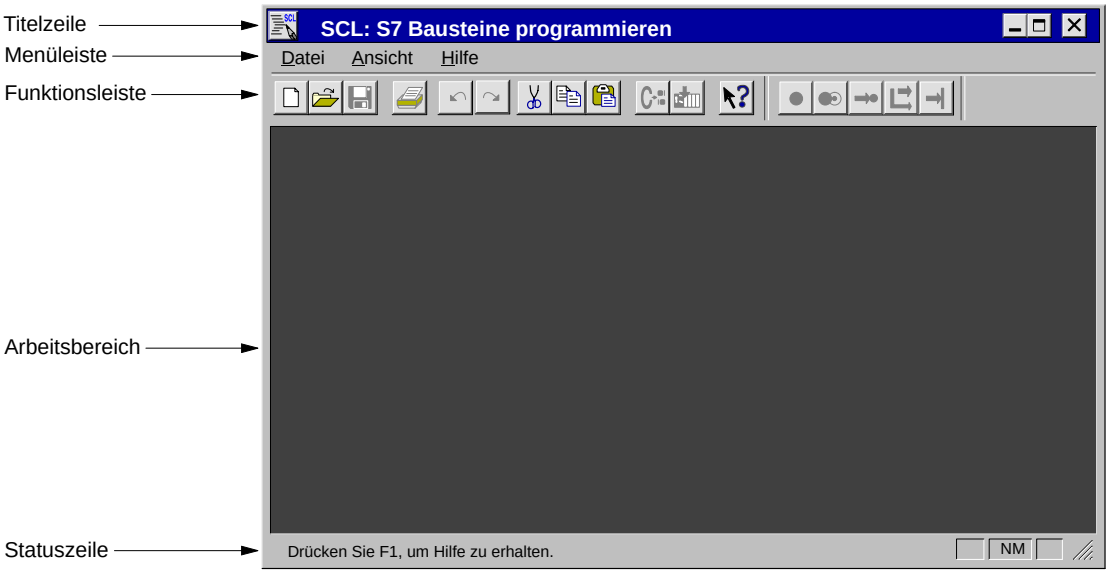


Bild 4-2 *Komponenten des SCL-Fensters*

Änderungen der Komponenten

Sie haben die Möglichkeit, folgende Komponenten an Ihre persönlichen Anforderungen anzupassen:

- Anzeige der Funktionsleiste
- Anzeige der Haltepunktleiste
- Anzeige der Statuszeile

Anpassen der Funktionsleiste

Die Anzeige der Funktionsleiste können Sie ein- oder ausschalten. Aktivieren bzw. deaktivieren Sie dazu den Menübefehl **Ansicht ▸ Funktionsleiste**. Ein Häkchen hinter dem Menübefehl zeigt an, ob der Befehl gerade aktiviert ist oder nicht.

Anpassen der Haltepunktleiste

Auch die Anzeige der Haltepunktleiste können Sie ein- oder ausschalten. Aktivieren bzw. deaktivieren Sie dazu den Menübefehl **Ansicht ▸ Haltepunktleiste**. Ein Häkchen hinter dem Menübefehl zeigt an, ob der Befehl gerade aktiviert ist oder nicht.

Anpassen der Statuszeile

Auch die Anzeige der Statuszeile können Sie ein- oder ausschalten. Aktivieren bzw. deaktivieren Sie dazu den Menübefehl **Ansicht ▸ Statuszeile**. Ein Häkchen hinter dem Menübefehl zeigt an, ob der Befehl gerade aktiviert ist oder nicht.

Anpassen der Entwicklungsumgebung

Der Editor und der Compiler erlauben Ihnen bestimmte Einstellungen vorzunehmen, die das Arbeiten erleichtern können.

- Einstellungen beim Erzeugen von Bausteinen
- Compilereinstellungen
- Editoreinstellungen

Erzeugen von Bausteinen

Sie können z. B. einstellen, ob bereits existierende Bausteine beim Übersetzen überschrieben werden sollen oder nicht. Wählen Sie dazu den Menübefehl **Extras ▸ Einstellungen** und klicken Sie im Dialogfeld ”Einstellungen” das Register ”Baustein erzeugen” an. Eine ausführliche Beschreibung der Optionen finden Sie im Abschnitt 5.5.

Anpassen des Compilers

Sie können auch den Übersetzungsvorgang an Ihre Anforderungen anpassen. Eine ausführliche Beschreibung der Optionen finden Sie im Abschnitt 5.5.

Wählen Sie dazu den Menübefehl **Extras ▸ Einstellungen** und klicken Sie im Dialogfeld ”Einstellungen” das Register ”Compiler” an.

Anpassen des Editors

Tabulatorweite, Sichern vor dem Übersetzen, die Anzeige von Zeilennummern und andere Optionen können Sie einstellen. Wählen Sie dazu den Menübefehl **Extras ▸ Einstellungen** und klicken Sie im Dialogfeld ”Einstellungen” das Register ”Editor” an.

4.3 Arbeiten mit dem SCL-Editor

Übersicht

Die SCL-Quelle besteht prinzipiell aus fortlaufendem Text. Bei der Texteingabe werden Sie durch den SCL-Editor mit Textverarbeitungsfunktionen unterstützt, die speziell auf SCL abgestimmt sind.

Editorfenster

Das Quellobjekt für Ihr Anwenderprogramm geben Sie über die Tastatur in das Arbeitsfenster ein. Sie haben dabei die Möglichkeit, mehrere Fenster mit unterschiedlichen Quellobjekten zu öffnen. Die Anordnung der Fenster können Sie über das Menü ”Fenster” steuern.

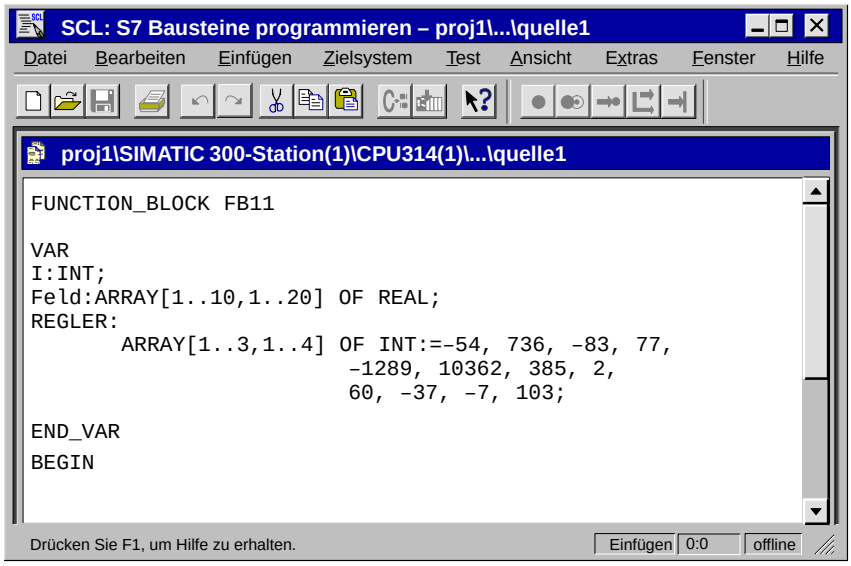


Bild 4-3 *SCL-Editierfenster*

Markieren von Text

In SCL können Sie Text markieren, indem Sie die Einfügemarke am Anfang des zu markierenden Bereichs positionieren und die linke Maustaste gedrückt halten, während Sie die Einfügemarke über den gewünschten Bereich ziehen.

Außerdem können Sie:

- den gesamten Text einer Quelle markieren, indem Sie den Menübefehl **Bearbeiten ▸ Alles Markieren** auswählen,
- ein Wort markieren, indem Sie es doppelt anklicken oder
- eine ganze Zeile markieren, indem Sie sie dreifach anklicken.

Suchen und Ersetzen

Mit dem Menübefehl **Bearbeiten ▸ Suchen /Ersetzen** wird ein Dialogfeld geöffnet, über das Sie eine Zeichenkette suchen und durch andere Textobjekte ersetzen können.

Einfügen von Vorlagen

Das Einfügen von Vorlagen ermöglicht Ihnen ein effizientes Programmieren und erleichtert die Einhaltung der Syntax. In SCL können Sie:

- Vorlagen für Bausteine einfügen, indem Sie den Menübefehl **Einfügen ▸ Bausteinvorlage** wählen.
- Vorlagen für Kontrollstrukturen einfügen, indem Sie den Menübefehl **Einfügen ▸ Kontrollstruktur** wählen.

Ausschneiden, Kopieren, Einfügen, Löschen

Textobjekte können Sie wie gewohnt ausschneiden, kopieren, einfügen und löschen. Die zugehörigen Menübefehle befinden sich im Menü **Bearbeiten**.

In der Regel können Sie die Objekte auch durch ”ziehen und loslassen” (Drag&Drop) verschieben oder kopieren.

Gehe zu

Mit dem Menübefehl **Bearbeiten ▸ Gehe zu ▸ Zeile** wird ein Dialogfeld geöffnet, über das Sie die Einfügemarke am Anfang einer gewünschten Zeile positionieren können, indem Sie die Zeilennummer eintragen und mit ”OK” bestätigen.

Rückgängig, Wiederherstellen

Mit dem Menübefehl **Bearbeiten ▸ Rückgänig** können Sie eine Aktion z.B. das Löschen einer Zeile wieder rückgänig machen.
Der Menübefehl **Bearbeiten ▸ Wiederherstellen** ermöglicht das Wiederherstellen einer rückgänig gemachten Aktion.

Programmieren mit SCL

5

Übersicht

Beim Programmieren sind eine Reihe von Arbeitsschritten notwendig. Dieses Kapitel beschreibt diese Arbeitsschritte und gibt Ihnen eine mögliche Reihenfolge vor.

Kapitelübersicht

5.1 Erstellen von Anwenderprogammen in SCL

Voraussetzungen der Programm-erstellung

Bevor Sie mit SCL ein Programm erstellen, sollten Sie folgende Arbeiten durchführen:

1. Richten Sie mit dem SIMATIC Manager ein Projekt ein.
2. Ordnen Sie mit dem SIMATIC Manager jeder CPU die Kommunikationsadresse im Netz zu.
3. Konfigurieren und parametrieren Sie die Zentral- und Signalbaugruppen.
4. Legen sie eine Symboltabelle an, falls Sie für Speicherbereiche der CPU oder für Bausteinbezeichnungen symbolische Adressen verwenden möchten.

Erstellen der Symboltabelle

Wenn Sie in Ihrem SCL-Programm symbolische Adressen für Speicher-bereiche der CPU oder Bausteinbezeichnungen verwenden wollen, müssen Sie eine Symboltabelle erstellen. SCL greift bei der Übersetzung auf diese Tabelle zu. Das Erstellen der Symboltabelle und die Werteingabe in die Symboltabelle erfolgt mit STEP 7.

Sie können die Symboltabelle mit dem SIMATIC Manager oder direkt mit SCL über den Menübefehl **Extras ▸ Symboltabelle** öffnen.

Darüberhinaus ist es möglich, als Textdatei vorliegende Symboltabellen, die mit beliebigen Texteditoren erstellt sein können, zu importieren und weiterzubearbeiten (Informationen hierzu finden Sie in **/231/**).

Wie gehen Sie vor?

Um ein Anwenderprogramm mit SCL zu erstellen, legen Sie zunächst eine SCL-Quelle an. In dieser Quelle können Sie entweder einen oder mehrere Bausteine (OB, FB, FC, DB und UDT) editieren und anschließend in einem Batchlauf übersetzen. Durch das Übersetzen der Quelle werden die enthaltenen Bausteine im Behälter ”Anwenderprogramm” (<AP-off>, siehe Bild 5-1) desselben S7-Programms aufgenommen, in dem auch die Quelle gespeichert ist.

Das Anlegen und Editieren der SCL-Quelle können Sie mit dem integrierten Editor oder mit einem Standard-Editor vornehmen. Quellen, die Sie mit einem Standardeditor erzeugt haben, müssen Sie mit dem SIMATIC Manager in das Projekt importieren. Danach können Sie sie öffnen, um sie weiterzubearbeiten oder zu übersetzen.

5.2 Anlegen und Öffnen einer SCL-Quelle

Übersicht

Quellen, die Sie mit SCL anlegen, lassen sich folgendermaßen in die Struktur eines S7-Programms einordnen:

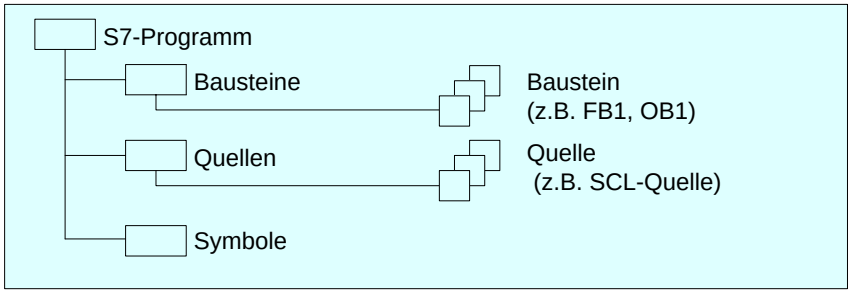


Bild 5-1 *Struktur eines S7-Programms im SIMATIC Manager*

Anlegen der SCL-Quelle

Zum Anlegen einer Quelle für SCL gehen Sie folgendermaßen vor:

1. Wählen Sie den Menübefehl **Datei ▸ Neu** oder klicken Sie das Symbol ”Neu” in der Funktionsleiste an.
2. Wählen Sie im Dialogfeld ”Neu” das gewünschte Projekt und in einem zugehörigen S7-Programm den Quellbehälter aus.
3. Öffnen Sie den Quellbehälter und wählen Sie den Menübefehl **Einfügen ▸ S7-Software ▸ SCL-Quelle**.
4. Markieren Sie die Quelle und wählen Sie den Menübefehl **Bearbeiten ▸ Objekteigenschaften**. Tragen Sie im Register ”Allgemein” den Namen des Quellobjekts ein. Der Name kann bis zu 24 Zeichen lang sein. Groß- und Kleinschreibung wird bei Quellennamen beachtet.
5. Doppelklicken Sie auf die Quelle. Es wird ein leeres Fenster geöffnet, in dem Sie die SCL-Quelle editieren können.

Öffnen einer SCL-Quelle

Sie können ein Quellobjekt öffnen, das Sie bereits mit SCL erzeugt und abgespeichert haben, um es weiterzubearbeiten oder zu übersetzen.

Gehen Sie folgendermaßen vor:

1. Wählen Sie den Menübefehl **Datei ▸ Öffnen**, oder klicken Sie das Symbol ”Öffnen” an.
2. Wählen Sie im Dialogfeld ”Öffnen” das gewünschte Projekt und das zugehörige S7-Programm aus.
3. Stellen Sie sicher, daß der Filter ”SCL-Quelle” eingestellt ist und wählen Sie den Quellbehälter aus.
4. Im Dialogfeld werden alle SCL-Quellen des S7-Programms angezeigt. Wählen Sie das gewünschte Objekt aus und bestätigen Sie mit ”OK” oder mit einem Doppelklick auf die Quelle.

Quellen, die Sie mit einem Standardeditor erzeugt haben, können Sie auf die gleiche Weise öffnen, nachdem Sie sie mit dem SIMATIC Manager in das Projekt importiert haben.

5.3 Eingeben von Vereinbarungen, Anweisungen und Kommentaren

Übersicht

Eine SCL-Quelle müssen Sie nach fest vereinbarten syntaktischen Regeln eingeben. Diese Regeln sind Bestandteil der *Sprachbeschreibung* und sind komplett im Anhang aufgeführt.

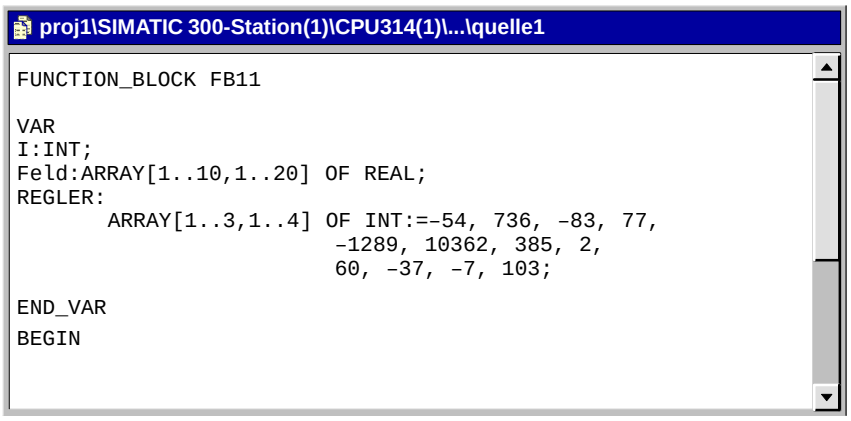


Bild 5-2 *SCL-Quelle*

Regeln

Beachten Sie folgende Konventionen bei der Eingabe:

- In einer SCL-Quelle können beliebig viele Codebausteine (FB, FC, OB), Datenbausteine (DB) und anwenderdefinierte Datentypen (UDT) formuliert werden, wobei jede Bausteinart typisch aufgebaut ist (siehe Kapitel 8.).
- Groß- und Kleinschreibung wird nur bei symbolischen Bezeichnern (z.B. bei Variablennamen und Zeichenliteralen) berücksichtigt.
- Aufgerufene Bausteine stehen vor den Bausteinen, in denen sie aufgerufen werden.
- Anwenderdefinierte Datentypen (UDT) stehen vor den Bausteinen, in denen sie verwendet werden.
- Globale Datenbausteine stehen vor allen Bausteinen, die auf sie zugreifen.
- Halten Sie sich an die Richtlinien für Format und Syntax, die im Teil *Sprachbeschreibung* dieses Handbuches angeführt sind.

5.4 Speichern und Drucken einer SCL-Quelle

Speichern einer SCL-Quelle

Unter dem Begriff ”Speichern” wird in SCL immer das Speichern der Quelldateien verstanden. Bausteine werden in SCL beim Übersetzen der Quelldatei erzeugt und automatisch im zugehörigen Programmverzeichnis abgelegt.

Für das Speichern einer SCL-Quelle haben Sie mehrere Möglichkeiten:

- Wählen Sie den Menübefehl **Datei ▸ Speichern**, oder klicken Sie das Symbol ”Speichern” in der Funktionsleiste an.
  Die SCL-Quelle wird aktualisiert.
- Wenn Sie eine Kopie der aktuellen SCL-Quelle erstellen möchten, wählen Sie den Menübefehl **Datei ▸ Speichern unter**. Es erscheint das Dialogfeld ”Speichern unter”, in dem Sie den Namen und den Pfad des Duplikats angeben können.
- Wenn Sie den Menübefehl **Datei ▸ Schließen** ausführen und eine geänderte SCL-Quelle noch nicht gespeichert haben, werden Sie gefragt, ob Sie die Änderungen speichern, verwerfen oder den Befehl **Schließen** abbrechen möchten.

  Anstatt des Menübefehls **Datei ▸ Schließen** können Sie auch das Symbol ”Schließen” in der Titelzeile anklicken.

  Auch wenn Sie SCL über den Menübefehl **Datei ▸ Beenden** verlassen wollen, und geöffnete Quellen nicht in ihrem aktuellen Stand gespeichert sind, erscheint für jede Quelle ein Dialog, über den Sie die Änderungen speichern oder verwerfen können.

Quellobjekt drucken

Die in Ihrer SCL-Quelle befindlichen Bausteine, Vereinbarungen und Anweisungen können Sie jederzeit auf einem Drucker ausgeben. Dazu müssen Sie den Drucker über die Windows-Systemsteuerung installiert und eingerichtet haben. Gehen Sie folgendermaßen vor:

- Klicken Sie in der Funktionsleiste das Symbol ”Drucken” an, oder wählen Sie den Menübefehl **Datei ▸ Drucken**. Es erscheint ein Dialogfeld, in dem Sie verschiedene Druckoptionen, wie z.B. den Druckbereich oder die Anzahl der Kopien, auswählen können.
  Bestätigen Sie mit ”OK”, um das Dokument an den Drucker zu senden.

Einrichten der Seite

Über den Menübefehl **Datei ▸ Seite einrichten** können Sie das Seitenformat Ihres Ausdrucks festlegen.

Kopf- und Fußzeilen erstellen

Kopf- und Fußzeilen für Ihre zu druckenden Dokumente können Sie im SIMATIC Manager mit dem Menübefehl **Datei ▸ Schriftfeld** einstellen.

Druckvorschau

Über den Menübefehl **Datei ▸ Druckvorschau** können Sie die vorgenommenen Einstellungen in der Seitenansicht überprüfen, bevor Sie das Dokument zum Drucker senden. Eine Bearbeitung ist hier nicht möglich.

5.5 Der Übersetzungsvorgang

Übersicht

Bevor Sie Ihr Programm ablaufen lassen oder testen können, müssen Sie es übersetzen. Durch Anstoßen des Übersetzungsvorgangs (siehe unten) aktivieren Sie den Compiler. Der Compiler hat folgende Eigenschaften:

- Der Compiler arbeitet im Batch-Modus, d.h. er bearbeitet eine SCL-Quelle als Einheit. Partielle Übersetzungen (z.B. zeilenweise) sind nicht möglich.
- Der Compiler überprüft die Syntax einer SCL-Quelle und zeigt anschließend alle Fehler an, die er während der Übersetzung gefunden hat.
- Er erzeugt Bausteine mit Test-Informationen, wenn die SCL-Quelle fehlerfrei und die entsprechende Option (siehe unten) gesetzt ist. Die Option Test-Informationen müssen Sie bei jedem Programm wählen, das Sie mit SCL auf Hochsprachen-Niveau testen möchten.
- Er generiert bei jedem Aufruf eines Funktionsbausteins einen zugehörigen Instanz-Datenbaustein, sofern dieser nicht schon existiert.

Einstellen des Compilers

Sie haben die Möglichkeit, den Übersetzungsvorgang an Ihre individuellen Anforderungen anzupassen. Wählen sie dazu den Menübefehl **Extras** ▸ **Einstellungen**, und klicken Sie im Dialogfeld ”Einstellungen” das Register ”Compiler” an. Die Optionen können Sie durch Mausklick ein- bzw. ausschalten.

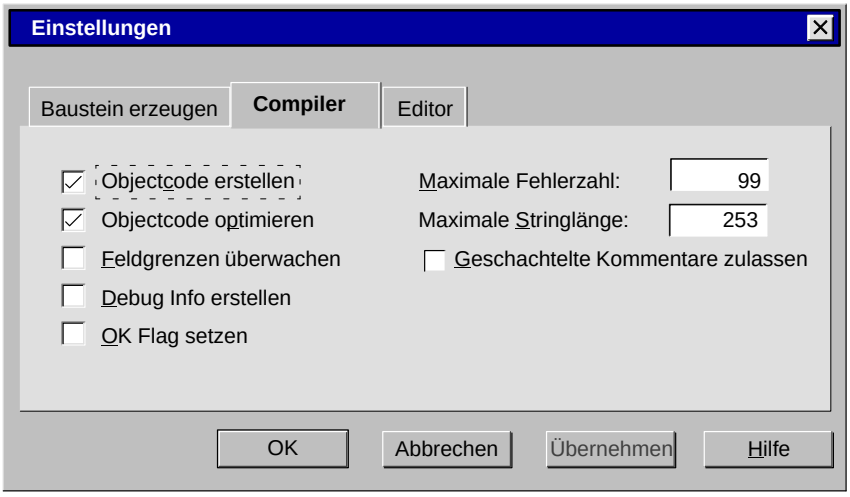


Bild 5-3 *Dialogfeld ”Einstellungen”, Register ”Compiler”*

Optionen

Die Optionen haben die folgenden Bedeutungen:

- **Maximale Fehleranzahl**: Der Compiler bricht die Übersetzung einer SCL-Quelle ab, wenn die angegebene Anzahl von Fehlern erreicht ist.
- **Objectcode erstellen**: Auf einem AS ablauffähigen Code erzeugen ja/ nein.
- **Objectcode optimieren**: Kürzeren Code erzeugen. Wenn die Option Debug Info erstellen gewählt ist, sind nicht alle Optimierungen möglich.
- **Feldgrenzen überwachen**: Zur Laufzeit überprüfen, ob Feldindizes in dem - laut Vereinbarung - für das Feld zulässigen Bereich liegen. Wenn ein Feldindex den zulässigen Bereich überschreitet, wird das OK-Flag auf FALSE gesetzt (vorausgesetzt die Option OK–Flag setzen ist aktiviert).
- **Debug Info erstellen**: Test-Informationen erzeugen ja/nein. Test-Informationen sind zum Testen mit dem Hochsprachen-Debugger notwendig.
- **OK-Flag setzen**: Zur Laufzeit soll jede fehlerhafte Operation die OK-Variable auf FALSE setzen.
- **Maximale Stringlänge:** Standardlänge des Datentyps “STRING” reduzieren. Die Standardlänge beträgt in der Grundeinstellung 254 Zeichen. Um die Ressourcen Ihrer CPU besser zu nutzen, können Sie die Standardlänge hier reduzieren.
- **Geschachtelte Kommentare zulassen:** In der SCL-Quelle dürfen mehrere Kommentare ineinander geschachtelt werden ja/nein.

Baustein erzeugen

Im Register ”Baustein erzeugen” haben Sie weitere Möglichkeiten, den Übersetzungsvorgang zu beeinflussen:

- Sie können einstellen, ob bereits existierende Bausteine beim Übersetzen überschrieben werden sollen oder nicht.
- Sie können automatisch beim Übersetzen einer Quelle Referenzdaten erzeugen lassen. Ist diese Option angeklickt, verlängert sich allerdings der Übersetzungsvorgang.
- Die Option ”Systemattribut S7_server berücksichtigen” klicken Sie an, wenn der Baustein für die Meldungs- oder Verbindungsprojektierung relevant ist. Auch diese Option verlängert den Übersetzungsvorgang.

Den Übersetzungsvorgang anstoßen

Sie haben zwei Möglichkeiten, den Übersetzungsvorgang anzustoßen:

- Wählen Sie den Menübefehl **Datei ▸ Übersetzen.**
- Klicken Sie das Symbol ”Übersetzen” in der Funktionsleiste an.

Um sicherzugehen, daß Sie immer die neuste Version Ihrer SCL-Quelle übersetzen, ist es ratsam, den Menübefehl **Extras ▸ Einstellungen** zu wählen und im Register ”Editor” die Option ”Sichern vor Übersetzen” anzuklicken. Der Menübefehl **Datei ▸ Übersetzen** speichert die SCL-Quelle dadurch implizit.

Nach dem Übersetzen

Nach dem Übersetzungsvorgang wird Ihnen eine fehlerfreie Übersetzung gemeldet, oder Fehler und Warnungen werden in einem Fenster nach Bild 5-4 angezeigt:

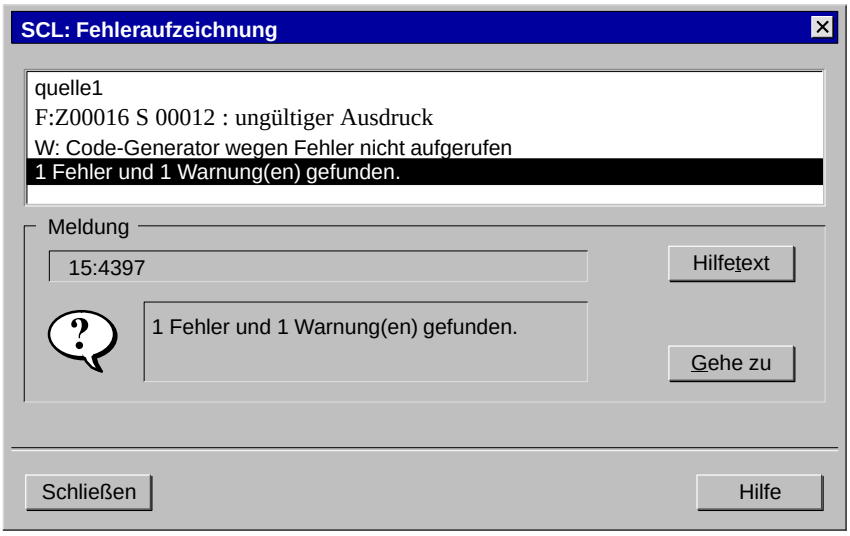


Bild 5-4 *Fenster für Fehlermeldungen und Warnungen*

Ursachen für Fehler und Warnungen finden

Jede Meldung wird mit zugehöriger Zeilen- und Spaltenposition, sowie einer Kurzbeschreibung angegeben. Eine ausführliche Beschreibung des Fehlers bzw. der Warnung erhalten Sie, indem Sie die gewünschte Meldung markieren und die Schaltfläche ”Hilfe” anklicken.

Mit einem Doppelklick auf eine Meldung können Sie die Schreibmarke an der betreffenden Stelle der SCL-Quelle positionieren.

Durch diese beiden Möglichkeiten können Sie Fehler und Warnungen rasch und einfach lokalisieren und korrigieren.

5.6 Übertragen des erstellten Anwenderprogramms in das AS

Übersicht

Bei der Übersetzung einer SCL-Quelle werden aus der Quelle die Bausteine erzeugt und im Behälter ”Bausteine” des S7-Programms gespeichert. In SCL können Sie danach nur diese Bausteine aus dem Programmiergerät in die CPU laden.

Verwenden Sie den SIMATIC Manager, wenn Sie weitere Bausteine des S7-Programms in das Automatisierungssytem übertragen möchten.

Voraussetzungen

Um das Anwenderprogramm in das AS zu laden, müssen folgende Voraussetzungen erfüllt sein:

- Zwischen dem Programmiergerät und dem Automatisierungssystem besteht eine Verbindung.
- Die Bausteine, die geladen werden sollen, müssen fehlerfrei übersetzt worden sein.

Urlöschen des CPU-Speichers

Mit der Funktion Urlöschen können Sie online das gesamte Anwenderprogramm in einer CPU löschen. Beachten Sie, daß dabei außerdem die CPU zurückgesetzt wird, alle bestehenden Verbindungen zur CPU abgebrochen werden und, sofern eine Memory Card gesteckt ist, der Inhalt der Memory Card in den internen Ladespeicher kopiert wird. Gehen Sie folgendermaßen vor:

1. Wählen Sie den Menübefehl **Zielsystem ▸ Betriebszustand** und schalten Sie die CPU auf STOP.
2. Wählen Sie den Menübefehl **Zielsystem ▸ Urlöschen**.
3. Bestätigen Sie die Aktion im darauf erscheinenden Dialogfeld.

Laden in das Zielsystem

Das Laden der Bausteine im Betriebszustand STOP ist vorteilhaft, da beim Überschreiben eines alten Programms im Betriebszustand RUN Fehler entstehen können. Gehen Sie folgendermaßen vor:

1. Wählen Sie den Menübefehl **Zielsystem ▸ Laden**.
2. Ist der Baustein schon im RAM der CPU vorhanden, bestätigen Sie nach Rückfrage, ob der Baustein überschrieben werden soll.

5.7 Erstellen einer Übersetzungssteuerdatei

Übersicht

Die Übersetzung von mehreren SCL-Quellen können Sie automatisieren, indem Sie eine Übersetzungssteuerdatei anlegen.

Übersetzungs-steuerdatei

Für Ihr STEP 7-Projekt können Sie eine Übersetzungssteuerdatei anlegen. Dort tragen Sie die Namen von SCL-Quellen ein, die sich im Projekt befinden. Diese SCL-Quellen sollen in einem Batchlauf übersetzt werden.

Erstellen

Das Erstellen führen Sie in folgenden Schritten durch:

- Beim Erzeugen einer Datei mit ”Neu” oder beim ”Öffnen” müssen Sie den Filter für ”Übersetzungssteuerdatei” anlegen.
- Die Datei, mit der Sie jetzt arbeiten, hat als spezielles Kennzeichen die Extension ”.inp”.
- Wenn Sie diese Datei übersetzen, werden die über ihre Namen angegebenen Dateien der Reihe nach übersetzt.

Übersetzen

Beim Übersetzen werden die erzeugten Bausteine im Behälter ”Bausteine” des S7-Programms abgelegt.

Testen eines Programmes

6

Übersicht

Die Testfunktionen von SCL bieten die Möglichkeit, ein Programm in seinem Ablauf im AS zu kontrollieren und damit mögliche Fehler zu finden.

Syntaxfehler werden von SCL beim Übersetzungsvorgang erkannt und angezeigt. Die Laufzeitfehler in der Ausführung des Programms werden durch Systemalarme angezeigt; logische Programmierfehler können Sie mit den Testfunktionen von SCL finden.

Wo finden Sie weitere Informationen?

Ausführliche Informationen zum Testen mit SCL finden Sie in der Online-Hilfe. Durch die Online-Hilfe erhalten Sie während der Arbeit mit SCL Antworten auf gezielte Fragen.

Kapitelübersicht

6.1 Übersicht

Hochsprachen-Niveau

Mit den Testfunktionen von SCL können Sie in SCL programmierte Anwenderprogramme auf Hochsprachen-Niveau testen. Durch diese Art des Testens können Sie:

- Programmierfehler entdecken
- Auswirkungen eines Anwenderprogrammes auf den Ablauf in der CPU beobachten und kontrollieren

Voraussetzungen

Bevor Sie ein SCL-Programm testen können, müssen Sie folgende Schritte erfolgreich durchgeführt haben:

1. Das Programm müssen Sie mit den Übersetzungsoptionen ”Objectcode erstellen” und ”Debug Info erstellen” fehlerfrei übersetzen. Die Einstellungen können Sie mit dem Menübefehl **Extras ▸ Einstellungen** im Register ”Compiler” wählen.
2. Zwischen PG/PC und CPU müssen Sie eine Online-Verbindung sicherstellen.
3. Darüber hinaus müssen Sie das Programm in die CPU laden. Das können Sie über den Menübefehl **Zielsystem ▸ Laden** durchführen.

Testfunktionen von SCL

Tabelle 6-1 zeigt die Namen und kurze Charakterisierungen der wesentlichen Testfunktionen, die Sie in SCL aufrufen können.

Tabelle 6-1 Die Testfunktionen im Überblick

Funktion	Charakterisierung
Kontinuierlich beobachten (S7-300/400-CPUs)	Namen und aktuelle Werte von Variablen eines Beobachtungsbereichs ausgeben
Haltepunkte aktiv (nur S7-400-CPUs)	Haltepunkte setzen, löschen und bearbeiten: Test in Einzelschritten
Variablen beobachten / steuern	Aktuelle Werte von globalen Daten beobachten/festlegen
Referenzdaten erzeugen	Übersicht über das Anwenderprogramm anlegen
Testfunktionen des STEP 7-Basispakets	CPU-Betriebszustand abfragen/ändern

Hinweis

Ein Test bei laufendem Anlagenbetrieb kann bei Funktionsstörungen oder Programmfehlern schwere Sach- und Personenschäden verursachen! Vergewissern Sie sich, daß keine gefährlichen Zustände eintreten können, bevor Sie die Testfunktionen ausführen.

6.2 Testfunktion ”Kontinuierlich beobachten”

Übersicht

Beim kontinuierlichen Beobachten eines Programms können Sie eine Gruppe von Anweisungen testen. Diese Gruppe von Anweisungen nennt man auch Beobachtungsbereich.

Während des Testlaufs werden die Werte der Variablen und Parameter dieses Bereichs in chronologischer Abfolge angezeigt und zyklisch aktualisiert. Liegt der Beobachtungsbereich in einem Programmteil, der in jedem Zyklus durchlaufen wird, können die Werte der Variablen in der Regel nicht aus aufeinanderfolgenden Zyklen erfaßt werden.

Werte, die sich im aktuellen Durchlauf geändert haben, werden in schwarzer Schrift angezeigt, Werte, die sich nicht geändert haben, werden in hellgrauer Schrift dargestellt.

Der Umfang der zu testenden Anweisungen ist durch die Leistungsfähigkeit der angeschlossenen CPUs begrenzt. Da die SCL-Anweisungen des Quellcodes in unterschiedlich viele Anweisungen im Anwenderprogramm abgebildet werden, ist die Länge des Beobachtungsbereichs variabel. Sie wird von SCL ermittelt und markiert, wenn Sie die erste Anweisung des gewünschten Beobachtungsbereichs auswählen.

Testmodus

Beim Testen im Modus ”Kontinuierlich beobachten” werden die aktuellen Werte der Daten im Beobachtungsbereich abgefragt und angezeigt. Das Abfragen geschieht, während der Beobachtungsbereich durchlaufen wird, und bewirkt meist eine Verlängerung der Zykluszeiten.

Damit Sie diese Verlängerung beeinflussen können, bietet SCL zwei verschiedene Testumgebungen an.

- **Testumgebung ”Prozeß”:**

  In der Testumgebung ”Prozeß” schränkt der SCL-Debugger den maximalen Beobachtungsbereich so ein, daß die Zykluszeiten beim Testvorgang die realen Ablaufzeiten des Prozesses nicht oder nur unwesentlich überschreiten.

- **Testumgebung ”Labor”:**

  In der Testumgebung ”Labor” wird der Beobachtungsbereich lediglich durch die Leistungsfähigkeit der angeschlossenen CPU begrenzt. Die Zykluszeiten können sich jedoch gegenüber dem realen Prozeß verlängern, wobei der maximale Beobachtungsbereich größer ist als in der Testumgebung ”Prozeß”.

Wie verwenden Sie "Kontinuierlich beobachten"?

Gehen Sie folgendermaßen vor, um die Funktion "Kontinuierlich beobachten" auszuführen:

1. Stellen Sie sicher, daß die in Kapitel 6.1 genannten Voraussetzungen erfüllt sind.
2. Wählen Sie das Fenster aus, das die Quelle des zu testenden Programms enthält.
3. Wenn Sie die voreingestellte Testumgebung (Prozeß) ändern wollen, wählen Sie den Menübefehl **Test ▸ Testumgebung ▸ Labor**.
4. Positionieren Sie die Einfügemarke in der Zeile des Quelltextes, die die erste Anweisung des zu testenden Bereichs enthält.
5. Wählen Sie den Menübefehl **Test ▸ Kontinuierlich beobachten**.

   Ergebnis: Der größtmögliche Beobachtungsbereich wird ermittelt und durch einen grauen Balken am linken Rand des Fensters angezeigt. Das Fenster teilt sich, und in der rechten Hälfte des Fensters werden die Namen und aktuellen Werte der im Beobachtungsbereich liegenden Variablen zeilengerecht angezeigt.
6. Wählen Sie den Menübefehl **Test ▸ Kontinuierlich beobachten**, um den Testlauf zu unterbrechen und später fortzusetzen.
7. Wählen Sie den Menübefehl **Test ▸ Test beenden**, um den Testlauf zu beenden.

6.3 Testfunktion ”Haltepunkte aktiv”

Übersicht

Beim Testen mit der Funktion ”Haltepunkte aktiv” erfolgt der Testlauf in Einzelschritten. Sie können das Programm Anweisung für Anweisung ausführen und beobachten, wie sich die Werte der bearbeiteten Variablen verändern.

Nach dem Setzen der Haltepunkte können Sie das Programm zunächst bis zum ersten Haltepunkt laufen lassen und von dort aus mit dem schrittweisen Beobachten beginnen.

Haltepunkte

Haltepunkte können Sie an beliebigen Stellen im Anweisungsteil des Quelltextes definieren.

Erst wenn Sie den Menübefehl **Test ▸ Haltepunkte aktiv** wählen, werden die Haltepunkte in das Automatisierungssystem übertragen und aktiv geschaltet. Das Programm wird dann ausgeführt bis der erste Haltepunkt erreicht wird.

Die Anzahl der aktiven Haltepunkte ist CPU-abhängig:

- CPU 416: maximal 4 aktive Haltepunkte möglich.
- CPU 414: maximal 2 aktive Haltepunkte möglich.
- CPU 314: keine aktiven Haltepunkte möglich

Einzelschritt-funktionen

Wenn die Testfunktion **Haltepunkte aktiv** gestartet ist, können Sie folgende Funktionen durchführen:

- **Nächste Anweisung ausführen**

  Fortsetzung um eine Anweisung, dient zur Ausgabe der Variablenwerte.

- **Fortsetzen**

  Fortsetzung bis zum nächsten aktivierten Haltepunkt.

- **Ausführen bis Markierung**

  Fortsetzung bis zu einer Textmarke in der Quelle, die Sie definieren.

Hinweis

Beachten Sie, daß die maximale Anzahl der aktiven Haltepunkte nicht überschritten wird, wenn Sie die Menübefehle **Nächste Anweisung ausführen** oder **Ausführen bis Markierung** benutzen, da diese implizit einen Haltepunkt setzen und auch aktivieren.

Wie benutzen Sie "Haltepunkte aktiv"?

Stellen Sie vor Testbeginn sicher, daß die in Kapitel 6.1 genannten Voraussetzungen erfüllt sind. Nun können Sie mit der Funktion "Haltepunkte aktiv" Ihr Programm in der CPU in Einzelschritten testen. Folgende schrittweise Beschreibung und die Darstellung im Bild 6-1 zeigt Ihnen die Vorgehensweise:

1. Wählen Sie das Fenster aus, das die Quelle des zu testenden Bausteins enthält.
2. Setzen Sie Haltepunkte, indem Sie die Schreibmarke an die gewünschte Stelle in der Programmquelle positionieren und den Menübefehl **Test ▸ Haltepunkt setzen** wählen. Die Haltepunkte werden am linken Fensterrand als roter Kreis dargestellt.
3. Starten Sie den Einzelschrittablauf, indem Sie den Menübefehl **Test ▸ Haltepunkte aktiv** wählen.

   Ergebnis: Das Fenster wird vertikal in zwei Hälften geteilt und der nächste Haltepunkt wird gesucht. Wird er erreicht, geht die CPU in den Betriebszustand HALT, und die erreichte Stelle wird durch einen gelben Pfeil markiert.
4. Die folgenden Einzelschrittfunktionen stehen nun alternativ zur Verfügung:
   - Wählen Sie den Menübefehl **Test ▸ Nächste Anweisung ausführen** (4a).
     Ergebnis: Die CPU geht kurz in den Betriebszustand RUN über. Beim Erreichen der nächsten Anweisung hält sie erneut an und zeigt die Werte der Variablen, die in der Vorgängeranweisung bearbeitet wurden, in der linken Fensterhälfte zeilengerecht an.
   - Wählen Sie den Menübefehl **Test ▸ Fortsetzen** (4b).
     Ergebnis: Die CPU geht in den Betriebszustand RUN. Beim Erreichen des nächsten aktiven Haltepunkts hält sie erneut und markiert den Haltepunkt am linken Fensterrand. Um die Variableninhalte anzuzeigen, müssen Sie erneut den Menübefehl **Test ▸ Nächste Anweisung ausführen** wählen.
   - Wählen Sie den Menübefehl **Test ▸ Ausführen bis Markierung** (4c). An der aktuellen Position der Markierung wird implizit ein Haltepunkt gesetzt und aktiviert. Die CPU geht in den Betriebszustand RUN über. Beim Erreichen der Markierung hält sie erneut und markiert diesen Haltepunkt. Um die Variableninhalte anzuzeigen, müssen Sie wieder den Menübefehl **Test ▸ Nächste Anweisung ausführen** wählen.
5. Gehen Sie zurück nach 2., wenn Sie mit veränderten Haltepunkten weiter testen wollen. Bei 2. können sie neue Haltepunkte setzen oder auch vorhandene Haltepunkte löschen.
6. Wählen Sie den Menübefehl **Test ▸ Haltepunkte aktiv** an, um die Testschleife zu unterbrechen.
7. Wenn Sie keine weiteren Anweisungen in der Quelle testen wollen, beenden Sie den Test, indem Sie den Menübefehl **Test ▸ Test beenden** wählen.

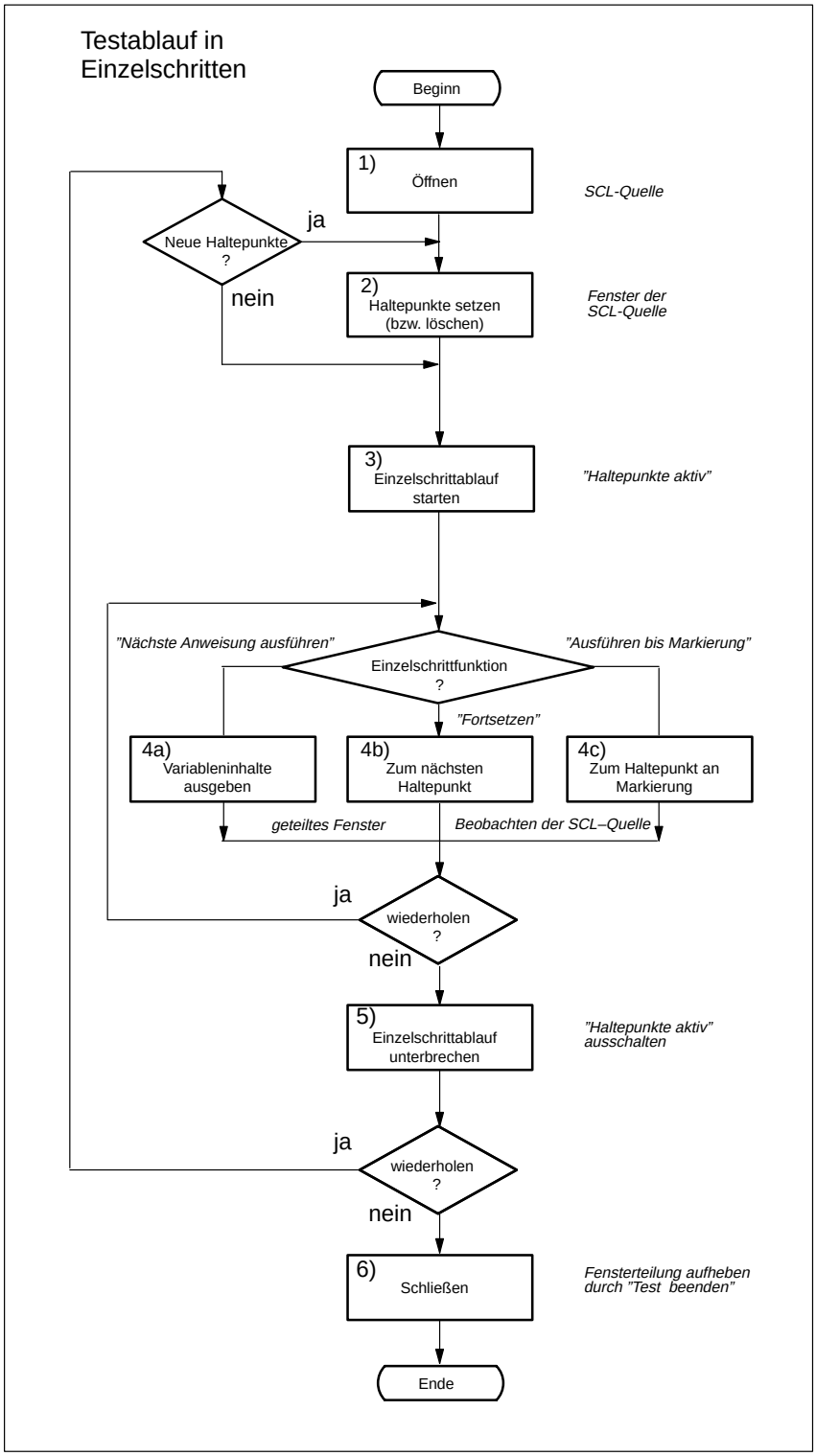


Bild 6-1 *Algorithmus des Testablaufs*

6.4 Testfunktion ”Variablen beobachten/steuern”

Übersicht

Beim Testen mit der Funktion ”Variablen beobachten/steuern” können Sie

- die aktuellen Werte von globalen Daten aus Ihrem Anwenderprogramm anzeigen lassen (beobachten)
- den Variablen eines Anwenderprogramms feste Werte zuweisen (steuern).

Variablen beobachten und steuern

Über den Menübefehl **Zielsystem ▸ Variablen beobachten/steuern** können Sie:

- Triggerpunkte und -bedingungen festlegen.
- Werte für die Variablen eines Anwenderprogramms angeben.

In beiden Fällen müssen Sie eine Variablentabelle erstellen, in der Sie angeben, welche Variablen bearbeitet werden sollen. Im Fall von ”Steuern” geben Sie zusätzlich die gewünschten Werte an.

Eine ausführliche Beschreibung der Testfunktion finden Sie im STEP 7-Benutzerhandbuch **/231/**.

6.5 Testfunktion ”Referenzdaten”

Übersicht

Sie können Referenzdaten erzeugen und auswerten, um sich das Testen und Ändern Ihres Anwenderprogramms zu erleichtern.

Referenzdaten umfassen: Programmstruktur, Querverweisliste, Belegungsplan, Liste nicht verwendeter Operanden, Liste der Operanden ohne Symbol.

Referenzdaten können Sie nutzen als

- Übersicht über ein gesamtes Anwenderprogramm
- Grundlage für Änderungen und Tests
- Ergänzung der Programmdokumentation.

Erzeugen von Referenzdaten

Zur Erzeugung von Referenzdaten haben Sie folgende Möglichkeiten:

- Über den Menübefehl **Extras ▸ Referenzdaten** können Sie die Referenzdaten bei Bedarf erzeugen, aktualisieren und anzeigen lassen.
- Über den Menübefehl **Extras ▸ Einstellungen** können Sie festlegen, daß die Referenzdaten automatisch beim Übersetzen einer Quelle erzeugt werden. Markieren Sie dazu im Register ”Baustein erzeugen” den Eintrag ”Referenzdaten erzeugen”.
  Das automatische Erzeugen der Referenzdaten verlängert jedoch den Übersetzungsvorgang.

Eine ausführliche Beschreibung der Testfunktion finden Sie im STEP 7-Benutzerhandbuch **/231/**.

6.6 Verwenden der STEP 7-Testfunktionen

AWL-Editor

Sie können Bausteine, die mit SCL übersetzt worden sind, in AWL öffnen und sie dann mit dem AWL-Editor testen.

CPU-Betriebszustand abfragen und ändern

Wählen Sie den Menübefehl **Zielsystem ▸ Betriebszustand**, um den aktuellen Betriebszustand der CPU abzufragen oder zu ändern.

Eigenschaften der CPU anzeigen

Der Menübefehl **Zielsystem ▸ Baugruppenzustand** öffnet ein Dialogfeld, in dem Sie:

- durch Auslesen des Diagnosepuffers die Ursache für den Betriebszustand STOP feststellen können.
- den Inhalt der Stacks der CPU abfragen können. Insbesondere der Unterbrechungsstack ist eine wichtige Hilfe bei der Fehlersuche.
- sich über die technischen Daten der CPU informieren können.
- die Uhrzeit und das Datum der CPU auslesen können.
- die Zykluszeit der CPU feststellen können.
- sich über die in der CPU enthaltenen Bausteine informieren können.
- Informationen zur Kommunikation der CPU abfragen können.

Bei diesen Funktionen muß die CPU online sein.

Teil 3: Sprachbeschreibung

Allgemeine SCL-Grundbegriffe

7

Übersicht

Sie erfahren in diesem Kapitel, welche Sprachmittel Ihnen SCL zur Verfügung stellt und wie Sie mit diesen Sprachmitteln umgehen können. Beachten Sie, daß hier nur die Grundkonzepte und notwendigen Definitionen vorausgeschickt werden, die in den nächsten Kapiteln vertieft werden.

Kapitelübersicht

7.1 Hilfen für die Sprachbeschreibung

Sprachbeschreibung von SCL

Basis für die Sprachbeschreibung in den einzelnen Kapiteln sind Syntaxdiagramme. Sie geben Ihnen einen guten Einblick in den syntaktischen (d.h. grammatikalischen) Aufbau von SCL. Eine vollständige Zusammenstellung aller Diagramme mit den Sprachelementen finden Sie in Anhang B.

Was ist ein 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 eingeführten Regeln aufbauen.

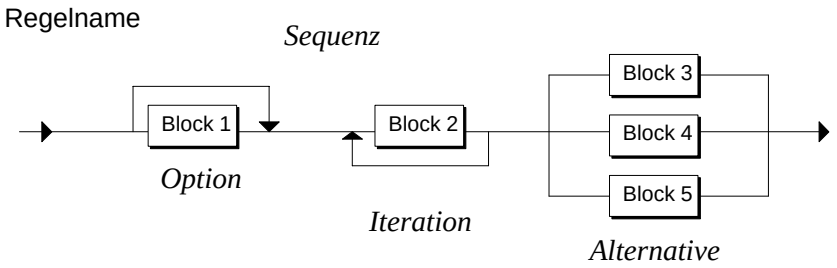


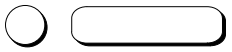
Bild 7-1 *Syntaxdiagramm*

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

- Sequenz: Folge von Blöcken
- Option: Überspringbarer Zweig
- Iteration: Wiederholung von Zweigen
- Alternative: Verzweigung

Welche Arten von Blöcken gibt es?

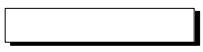
Ein Block ist ein Grundelement oder ein Element, das wiederum aus Blöcken zusammengesetzt ist. Folgendes Bild zeigt die Symbolarten, die den Blöcken entsprechen:



Grundelement, das nicht weiter erklärt werden muß.

Hier handelt es sich um druckbare Zeichen und Spezialzeichen, Schlüsselwörter und vordefinierte Bezeichner.

Die Angaben zu diesen Blöcken sind unverändert zu übernehmen.



Zusammengesetztes Element, das durch weitere Syntaxdiagramme beschrieben wird.

Was bedeutet Formatfreiheit?

Bei der Eingabe von Quelltexten sind sowohl die **syntaktischen Regeln** als auch **lexikalischen Regeln** zu beachten.

Die lexikalischen und syntaktischen Regeln sind in den Anhängen B und C detailliert beschrieben. Formatfreiheit bedeutet, daß Sie zwischen den Regelblöcken Formatierungszeichen wie Leerzeichen, Tabulatoren und Seitenwechsel sowie Kommentare einfügen können.

Lexikalische Regel

Bei den lexikalischen Regeln z. B. nach Bild 7-2 haben Sie **keine** Formatfreiheit. Wenn Sie die eine lexikalische Regel anwenden, müssen Sie die Angaben unverändert übernehmen.

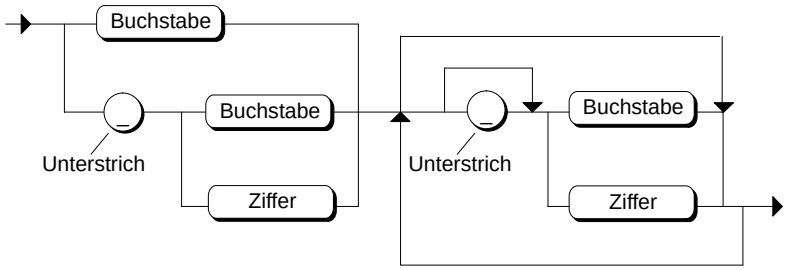


Bild 7-2 Beispiel für eine **lexikalische** Regel

Gültige Beispiele nach dieser dargestellten Regel wären:

```
R_REGLER3
_A_FELD
_100_3_3_10
```

Ungültige Beispiele aus den oben genannten Gründen sind:

```
1_1AB
RR__20
*#AB
```

Syntaktische Regel

Bei syntaktischen Regeln (z.B. Bild 7-3) haben Sie Formatfreiheit.

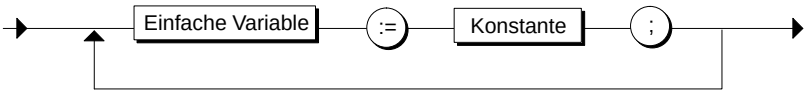


Bild 7-3 Beispiel für eine syntaktische Regel

Gültige Beispiele nach dieser dargestellten Regel wären:

```
VARIABLE_1  := 100;     SCHALTER:=FALSE;
VARIABLE_2  := 3.2;
```

7.2 Der SCL-Zeichensatz

Buchstaben, Ziffern

SCL benutzt aus dem Teilbereich des ASCII-Zeichensatzes:

- die Klein- und Großbuchstaben von A bis Z,
- die arabischen Ziffern 0 bis 9,
- Leerzeichen (ASCII-Wert 32) und alle Steuerzeichen (ASCII 0-31) einschließlich des Zeilenendezeichens (ASCII 13).

Sonstige Zeichen

Diese Zeichen haben in SCL eine festgelegte Bedeutung:

+	–	*	/	=	<	>	[	]	(	)
.	,	:	;	$	#	”	’	{	}	

Weitere Informationen

Sie finden im Anhang A eine detaillierte Auflistung aller verwendbaren Zeichen sowie eine Zuordnung, wie SCL diese Zeichen interpretiert.

7.3 Reservierte Wörter

Bedeutung

Reservierte Wörter sind Schlüsselwörter, die Sie nur wie vorbestimmt benutzen dürfen. Eine Unterscheidung zwischen Groß- und Kleinschreibung findet nicht statt.

Schlüsselwörter

AND
ANY
ARRAY
BEGIN
BLOCK_DB
BLOCK_FB
BLOCK_FC
BLOCK_SDB
BLOCK_SFB
BLOCK_SFC
BOOL
BY
BYTE

CASE
CHAR
CONST
CONTINUE
COUNTER
DATA_BLOCK
DATE
DATE_AND_TIME
DINT
DIV
DO
DT
DWORD
ELSE
ELSIF
END_CASE
END_CONST
END_DATA_BLOCK
END_FOR
END_STRUCT
END_VAR
END_WHILE
EXIT
FOR
FUNCTION
FUNCTION_BLOCK
GOTO
IF
INT
LABEL
MOD
NIL
NOT
OF
OR
ORGANIZATION_BLOCK
POINTER
REAL
REPEAT
RETURN
S5TIME
STRING
STRUCT
THEN
TIME
TIMER
TIME_OF_DAY
TO
TOD
TYPE
UNTIL
VAR

Schlüsselwörter, Fortsetzung

END_FOR
END_FUNCTION
END_FUNCTION_BLOCK
END_IF
END_LABEL
END_ORGANIZATION_BLOCK
END_REPEAT
END_TYPE
VAR
VAR_IN_OUT
VAR_INPUT
VAR_OUTPUT
VAR_TEMP
VOID
WHILE
WORD
XOR

Sonstige reservierte Namen

EN
ENO
OK
TRUE
FALSE
Namen der Standardfunktionen

7.4 Bezeichner in SCL

Definition

Ein Bezeichner ist ein Name, den Sie für ein Sprachobjekt von SCL, also eine Konstante, eine Variable, eine Funktion oder einen Baustein selbst vergeben können.

Regeln

Bezeichner können aus Buchstaben oder Ziffern in jeder beliebigen Reihenfolge zusammengesetzt sein, wobei nur das erste Zeichen ein Buchstabe oder ein Unterstrich sein muß. Sowohl Groß- wie Kleinbuchstaben sind erlaubt. Eine Unterscheidung zwischen Groß- und Kleinschreibung findet nicht statt (Anna und AnNa sind also zum Beispiel identisch).

Ein Bezeichner kann durch das folgende Syntaxdiagramm formal dargestellt werden:

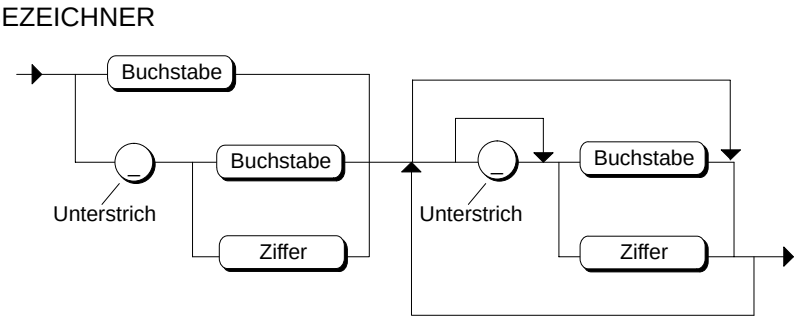


Bild 7-4 *Syntax: Bezeichner*

Beachten Sie bitte folgende Hinweise:

- Bei der Namensvergabe wählen Sie am besten eindeutige und aussagekräftige Namen, die zur Verständlichkeit des Programms beitragen.
- Achten Sie darauf, ob der Name nicht schon durch Schlüsselwörter (z.B. nach Tabelle 7-1) oder Standardbezeichner belegt ist.
- Die maximale Länge eines Bezeichners beträgt 24 Zeichen.
- Symbolische Namen für Bausteine (also andere Bezeichner wie nach Tabelle 7-1) müssen Sie in der Symboltabelle von STEP 7 definieren (Informationen hierzu finden Sie in **/231/**)

Beispiele

Die folgenden Namen sind gültige Bezeichner:

```
x          y12        Summe      Temperatur
Namen      Flaeche    Regler     Tabelle
```

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

`4ter`	Das erste Zeichen muß ein Buchstabe oder ein Unterstrich sein.
`Array`	ARRAY ist ein Schlüsselwort und nicht zugelassen.
`S Wert`	Leerstellen sind Zeichen und nicht erlaubt.

7.5 Standardbezeichner

Definition

SCL definiert bereits eine Reihe von Bezeichnern, für die deshalb der Name *Standardbezeichner* verwendet wird. Diese Standardbezeichner sind:

- die Bausteinschlüsselwörter und
- die Operandenkennzeichen für das Ansprechen von Speicherbereichen der CPU.

Baustein-schlüsselwörter

Diese Standardbezeichner werden für die absolute Adressierung von Bausteinen verwendet.

Die Tabelle 7-1 ist nach der SIMATIC-Mnemonik sortiert, die entsprechende internationale IEC-Mnemonik wird ebenfalls angegeben.

Tabelle 7-1 Bausteinschlüsselwörter

Mnemonik SIMATIC	Mnemonik IEC	kennzeichnet
DBx	DBx	Datenbaustein (Data Block)
FBx	FBx	Funktionsbaustein (Function Block)
FCx	FCx	Funktion (Function)
OBx	OBx	Organisationsbaustein (Organization Block)
SDBx	SDBx	Systemdatenbaustein (System Data Block)
SFCx	SFCx	Systemfunktion (System Function)
SFBx	SFBx	Systemfunktionsbaustein (System Function Block)
Tx	Tx	Zeitglied (Timer)
UDTx	UDTx	Anwenderdefinierter Datentyp (User defined Data Type)
Zx	Cx	Zähler (Counter)
x = Zahl zwischen 0 und 65533 DB0 = ist reserviert !		

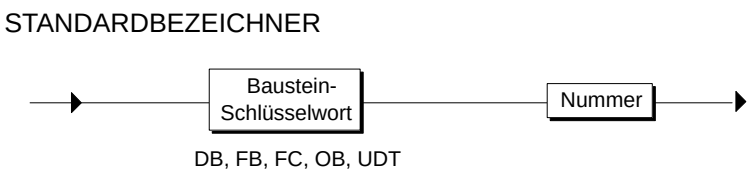


Bild 7-5 *Syntax: Standardbezeichner*

Die Tabelle 7-1 ist nach der SIMATIC-Mnemonik sortiert, die entsprechende internationale IEC-Mnemonik wird ebenfalls angegeben.

Gültige Bezeichnungen sind:

```
FB10
DB100
T141
```

Operanden-kennzeichen

Die Speicherbereiche einer CPU, können Sie von jeder Programmstelle aus mit ihrem Operandenkennzeichen ansprechen.

Die folgende Tabelle ist nach der SIMATIC-Mnemonik sortiert, die entsprechende internationale IEC-Mnemonik wird dazu angegeben.

Mnemonik (SIMATIC)	Mnemonik (IEC)	addressiert	Datentyp
Ax.y	Qx.y	Ausgang (über Prozeßabbild)	Bit
ABx	QBx	Ausgang (über Prozeßabbild)	Byte
ADx	QDx	Ausgang (über Prozeßabbild)	Doppelwort
AWx	QWx	Ausgang (über Prozeßabbild)	Wort
AXx.y	QXx.y	Ausgang (über Prozeßabbild)	Bit
Dx.y 1)	Dx.y 1)	Datenbaustein	Bit
DBx 1)	DBx 1)	Datenbaustein	Byte
DDx 1)	DDx 1)	Datenbaustein	Doppelwort
DWx 1)	DWx 1)	Datenbaustein	Wort
DXx.y 1)	DXx.y1)	Datenbaustein	Bit
Ex.y	Ix.y	Eingang (über Prozeßabbild)	Bit
EBx	IBx	Eingang (über Prozeßabbild)	Byte
EDx	IDx	Eingang (über Prozeßabbild)	Doppelwort
EWx	IWx	Eingang (über Prozeßabbild)	Wort
EXx.y	IXx.y	Eingang (über Prozeßabbild)	Bit
Mx.y	Mx.y	Merker	Bit
MBx	MBx	Merker	Byte
MDx	MDx	Merker	Doppelwort
MWx	MWx	Merker	Wort
MXx.y	MXx.y	Merker	Bit
PABx	PQBx	Ausgang (Peripherie direkt)	Byte
PADx	PQDx	Ausgang (Peripherie direkt)	Doppelwort
PAWx	PQWx	Ausgang (Peripherie direkt)	Wort
PEBx	PIBx	Eingang (Peripherie direkt)	Byte
PEDx	PIDx	Eingang (Peripherie direkt)	Doppelwort
PEWx	PIWx	Eingang (Peripherie direkt)	Wort
PEWx	PIWx	Eingang (Peripherie direkt)	Wort
PEWx	PIWx	Eingang (Peripherie direkt)	Wort
x = Zahl zwischen 0 und 65535 (absolute Byteadresse) y = Zahl zwischen 0 und 7 (Bitnummer)			

Gültige Beispiele sind.

```
E1.0        MW10        PAW5        DB20.DW3
```

1) *Diese Operandenkennzeichen gelten nur zusammen mit der Angabe des Datenbausteins.*

7.6 Zahlen

Übersicht

Zahlen können in SCL 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.

Integerzahlen

Eine Integerzahl enthält weder einen Dezimalpunkt noch einen Exponenten. Somit ist eine Integerzahl einfach eine Folge von Ziffern, die wahlweise mit einem Vorzeichen beginnt. In SCL sind 2 Integer-Typen realisiert, die jeweils unterschiedliche Wertebereiche haben, `INT` und `DINT` (siehe Kapitel 9).

Einige gültige Integerzahlen :

```
0          1          +1         -1
743        -5280      600_00     -32_211
```

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

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

Integerzahlen als binär, oktal- oder hexadezimale Zahl

In SCL können Sie Integerzahlen 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 Integerzahlen für Dezimal 15:

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

Realzahlen

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

Einige gültige Realzahlen :

```
0.0        1.0        -0.2       827.602
50000.0    -0.000743  12.3       -315.0066
```

Die folgenden Realzahlen sind **falsch**:

`1.`	Auf beiden Seiten des Dezimalpunktes muß eine Ziffer stehen.
`1,000.0`	Kommata sind nicht erlaubt.
`.3333`	Auf beiden Seiten des Dezimalpunktes muß eine Ziffer stehen.

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 Integerzahl sein. Die Basis 10 wird durch den Buchstaben E ersetzt.

Die Größe 3 x 10^{10} kann in SCL durch folgende Realzahlen dargestellt werden:

```
3.0E+10   3.0E10        3e+10       3E10
0.3E+11   0.3e11        30.0E+9     30e9
```

Die folgenden Realzahlen sind **falsch**:

`3.E+10`	Auf beiden Seiten des Dezimalpunkts muß eine Ziffer stehen.
`8e2.3`	Der Exponent muß eine Integerzahl sein.
`.333e–3`	Auf beiden Seiten des Dezimalpunkts muß eine Ziffer stehen.
`30 E10`	Leerzeichen sind nicht erlaubt.

Zeichenkette

Eine Zeichenkette ist eine Folge von Zeichen (d.h. Buchstaben, Ziffern und Sonderzeichen), die in Anführungszeichen stehen. Sowohl Klein- wie Großbuchstaben können benutzt werden.

Einige gültige Zeichenketten:

```
'ROT'     '76181 Karlsruhe'         '270–32–3456'
'DM19.95' 'Die richtige Antwort ist:'
```

Spezielle Formatierungszeichen, das Anführungzeichen (') oder ein $-Zeichen können Sie mit Fluchtsymbol $ eingeben.

Quelltext	nach Kompilierung
`'SIGNAL$'ROT$''`	`SIGNAL'ROT'`
`'50.0$$'`	`50.0$`
`'WERT$P'`	`WERT` *Seitenumbruch*
`'REG-$L'`	`REG-`*Zeilenumbruch*
`'REGEL$R`	`REGLER`*Wagenrücklauf*
`'SCHRITT$T'`	`SCHRITT` *Tabulator*

Für die nicht druckbaren Zeichen geben Sie die Ersatzdarstellung im Hexacode mit $*hh* an, wobei *hh* stellvertretend für den hexadezimal ausgedrückten Wert des ASCII-Zeichens steht.

Zur Unterbrechung einer Zeichenkette (für Kommentare die nicht ausgedruckt oder angezeigt werden sollen) stehen Ihnen in SCL die Zeichen $> und $< für eine Stringunterbrechung zur Verfügung.

7.7 Datentypen

Übersicht

Jede Vereinbarung einer Variablen muß den Typ dieser Variablen angeben. Der Typ legt den Wertebereich der Variablen fest und bestimmt die Operationen, die mit ihr ausgeführt werden können.

Ein bestimmter Datentyp bestimmt :

- die Art und Bedeutung eines Datenelements
- den zulässigen Bereich des Datenelements
- die zulässige Menge der Operationen, die mit einem Operanden eines Datentyps ausgeführt werden können
- die Schreibweise der Daten dieses Datentyps.

Arten der Datentypen

Man unterscheidet zwischen den folgenden Datentypen.

Tabelle 7-2 Elementare Datentypen

Datentypen	Bedeutung
elementare	Stellt Ihnen SCL standardmäßig zur Verfügung
zusammengesetzte	Können Sie erzeugen, indem Sie elementare Datentypen verknüpfen.
anwenderdefinierte	Definieren Sie speziell für Ihren Anwendungsfall unter einem frei wählbaren Namen.
Parametertypen	Können Sie nur für die Vereinbarung von Parametern verwendet werden.

Elementare Datentypen

Elementare Datentypen definieren die Struktur von Daten, die nicht in kleinere Einheiten zerlegt werden können. Sie entsprechen der Definition der Norm DIN EN 1131-3.

In SCL sind zwölf elementare Datentypen vordefiniert:

```
BOOL        CHAR        INT         TIME
BYTE                    DINT        DATE
WORD                    REAL        TIME_OF_DAY
DWORD                               S5TIME
```

Zusammengesetzte Datentypen

Zusammengesetzte Datentypen definieren Strukturen von Daten, die sich aus anderen Datentypen zusammensetzen. SCL läßt folgende zusammengesetzte Datentypen zu:

```
DATE_AND_TIME
STRING
ARRAY
STRUCT
```

Anwenderdefinierte Datentypen

Hierbei handelt es sich um globale Datentypen (UDT), die Sie in SCL für Ihren Anwendungsfall erstellen können. Sie können diesen Datentyp mit seiner UDT-Bezeichnung UDTx (x steht für Nummer) oder unter einem zugeordneten symbolischen Namen im Vereinbarungsteil eines Bausteins oder Datenbausteins verwenden.

Parametertypen

Zusätzlich zu elementaren, zusammengesetzten und anwenderdefinierten Datentypen können Sie Parametertypen zur Definition von Parametern verwenden. SCL bietet dazu die folgenden Parametertypen:

TIMER	BLOCK_FB	POINTER	ANY
COUNTER	BLOCK_FC		
	BLOCK_DB		
	BLOCK_SDB		

7.8 Variablen

Variablen-vereinbarung

Ein Bezeichner, dessen Wert während der Programmdurchführung geändert werden kann, wird *Variable* genannt. Jede Variable muß einzeln erklärt (d.h. vereinbart) werden, bevor sie innerhalb eines Codebausteins oder Datenbausteins benutzt werden kann. Die Variablenvereinbarung legt fest, daß ein Bezeichner eine Variable ist (und keine Konstante etc.) und spezifiziert durch die Zuordnung zum Datentyp den Variablentyp.

Je nach Gültigkeit der Variablen wird unterschieden zwischen:

- Lokaldaten
- Globalen Anwenderdaten
- erlaubten vordefinierten Variablen (Speicherbereiche einer CPU)

Lokaldaten

Lokaldaten sind Daten, die innerhalb eines Codebausteins (FC, FB, OB) vereinbart werden und nur für diesen Codebaustein Gültigkeit haben. Im einzelnen sind das:

Tabelle 7-3 Die Lokaldaten eines Bausteines

Variable	Bedeutung
Statische Variablen	Eine statische Variable ist eine lokale Variable, deren Wert über alle Bausteindurchläufe hinweg erhalten bleibt (Bausteingedächtnis). Sie dient der Speicherung von Werten eines **Funktionsbausteins**.
Temporäre Variablen	Temporäre Variablen gehören lokal zu einem Code-Baustein und belegen **keinen** statischen Speicherbereich. Ihr Wert bleibt nur während eines Bausteinablaufs erhalten. Auf temporäre Variablen kann außerhalb des Bausteins, in dem die Variablen deklariert wurden, **nicht** zugegriffen werden.
Bausteinparameter	Bausteinparameter sind formale Parameter eines Funktionsbausteins oder einer Funktion. Es sind lokale Variablen die dazu dienen, die beim Aufruf angegebenen aktuellen Parameter zu übergeben.

Globale Anwenderdaten

Globale Anwenderdaten sind Daten bzw. Datenbereiche die Sie von jeder Programmstelle aus nutzen können. Dazu müssen Sie Datenbausteine (DB) erstellen.

Wenn Sie einen DB erstellen, legen Sie in einer Strukturvereinbarung seinen Aufbau fest. Anstelle einer Strukturvereinbarung kann auch ein anwenderdefinierter Datentyp (UDT) verwendet werden. Die Reihenfolge, in der Sie die Strukturkomponente angeben, bestimmt die Reihenfolge der Daten in dem DB.

Speicherbereiche einer CPU

Auf die Speicherbereiche einer CPU können Sie über die Operandenkennzeichen (siehe Kapitel 7.5) direkt von jeder Programmstelle aus zugreifen, ohne daß Sie diese Variablen vereinbaren müssen.

Sie haben darüber hinaus immer die Möglichkeit, diese Datenbereiche auch symbolisch anzusprechen. Die Symbolzuordnung erfolgt in diesem Fall global über die Symboltabelle in STEP 7. Informationen hierzu finden sie in **/231/** .

7.9 Ausdrücke

Übersicht

Ein Ausdruck steht für einen Wert, der entweder bei der Übersetzung oder zur Laufzeit des Programms berechnet wird. Er besteht aus einem oder mehreren Operanden die durch Operatoren verknüpft sind. Die Auswertungsreihenfolge der Operatoren ist durch deren Priorität vorgegeben und kann außerdem durch Klammerung gesteuert werden.

- Arithmetische Ausdrücke
- Logische Ausdrücke
- Vergleichsausdrücke

Arithmetischer Ausdruck

Ein typischer Ausdruck ist z.B.:

```
(b*b-4*a*c)/(2*a)
```

Die Bezeichner a und b und die Zahlen 4 und 2 sind die Operanden; die Symbole *, – und / sind die entsprechenden Operatoren (Multiplikation, Subtraktion und Division). Der gesamte Ausdruck stellt eine Zahl dar.

Vergleichs-ausdrücke

Ein Vergleichsausdruck ist ein logischer Ausdruck, der entweder wahr oder falsch sein kann. Hier ein Beispiel eines Vergleichsausdrucks:

```
Sollwert < 100.0
```

In diesem Ausdruck ist `SOLLWERT` eine Realvariable, `100.0` eine Realzahl und das Symbol < ein Vergleichs-Operator. Der Ausdruck hat den Wert wahr, falls Sollwert einen Wert kleiner als 100.0 darstellt, andernfalls hat der Ausdruck den Wert **falsch**.

Logischer Ausdruck

Ein typischer Ausdruck dafür ist:

```
a AND NOT b
```

Die Bezeichner a und b sind die Operanden; die Schlüsselwörter AND und NOT logische Operatoren. Der gesamte Ausdruck stellt ein Bitmuster dar.

7.10 Anweisungen

Übersicht

Eine SCL-Anweisung ist eine ausführbare Aktion im Anweisungsteil eines Codebausteins. Es gibt in SCL drei grundlegende Anweisungen:

1. Wertzuweisungen (Zuordnung eines Ausdrucks zu einer Variablen)
2. Kontrollanweisungen (Wiederholung oder Verzweigung von Anweisungen)
3. Unterprogrammbearbeitung (Aufrufen oder Verzweigung von Codebausteinen)

Wertzuweisungen

Ein typische Wertzuweisung ist z. B.:

```
SOLLWERT := 0.99*SOLLWERT_ALT
```

In diesem Beispiel wird vorausgesetzt, daß `SOLLWERT` und `SOLLWERT_ALT` Realvariablen sind. Der Zuweisungsbefehl multipliziert den Wert `SOLLWERT_ALT` mit 0.99 und ordnet das Produkt der Variablen `SOLLWERT` zu. Beachten Sie, daß das Symbol für die Zuweisung `:=` ist.

Kontrollanweisungen

Eine typische Kontrollanweisung ist:

```
FOR Zaehler :=1 TO 20 DO
            LISTE[Zaehler]   := WERT+Zaehler;

END_FOR;
```

In diesem Beispiel wird die Zuweisung 20mal ausgeführt. Jedesmal wird im Feld `LISTE` der neu errechnete Wert in einen nächsthöheren Listenplatz eingetragen.

Unterprogrammbearbeitung

Mit der Angabe einer Bausteinbezeichnung für eine Funktion (FC) oder einen Funktionsbaustein (FB) wird der Codebaustein aufgerufen, der mit diesem Bezeichner deklariert wurde.[1)] Wenn die Vereinbarung des Codebausteins Formalparameter enthält, können den Formalparametern beim Aufruf aktuelle Operanden zugeordnet werden.

Alle Parameter, die im Vereinbarungsblock

```
VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT
```

eines Codebausteins aufgelistet sind, bezeichnet man als Formalparameter - die entsprechenden Parameter in den Aufrufen innerhalb des Anweisungsteils werden dagegen Aktualparameter genannt.

Die Übergabe der Aktualparameter an die Formalparameter ist Bestandteil des Aufrufs.

Eine typische Unterprogrammbearbeitung ist z. B.:

```
FC31(X:=5, Q1:=Quersumme);
```

1) *Wenn Sie im FC Formalparameter vereinbart haben, ist die Zuweisung von Aktualparametern zwingend erforderlich, bei FBs jedoch optional.*

7.11 SCL-Bausteine

Übersicht

In einer SCL-Quelldatei können Sie 1 bis n Bausteine als Quelltext programmieren.

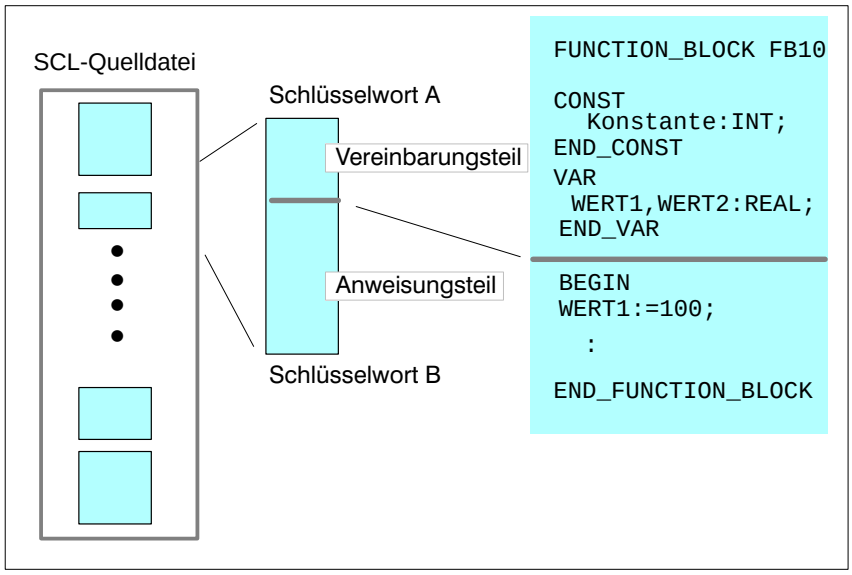
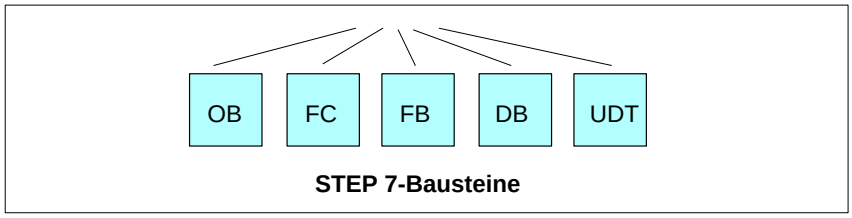


Bild 7-6 *Aufbau einer SCL-Quelldatei*

Bausteinarten

Die STEP 7-Bausteine sind durch ihre Funktion, ihre Struktur oder ihren Verwendungszweck abgegrenzte Teile eines Anwenderprogramms. Mit SCL können Sie folgende Bausteine programmieren:



Vorgefertigte Bausteine

Nicht jede Funktion müssen Sie selbst programmieren. Sie können auch auf vorgefertigte Bausteine zurückgreifen. Sie sind im Betriebssystem der Zentralbaugruppen oder in Bibliotheken *(S7lib)* des STEP7-Basispakets vorhanden und können z. B. für die Programmierung von Kommunikationsfunktionen genutzt werden.

Aufbau eines SCL-Bausteins

Jeder Baustein besteht aus folgenden Teilen:

- Bausteinkopfanfang/ende (Schlüsselwort entsprechend der Bausteinart)
- Vereinbarungsteil
- Anweisungsteil (wird bei Datenbausteinen als Zuweisungsteil bezeichnet)

Vereinbarungsteil

Im Vereinbarungsteil müssen sämtliche Festlegungen getroffen werden, die als Basis für den Anweisungsteil notwendig sind: z.B. Definition von Konstanten und Vereinbarung von Variablen und Parametern.

Anweisungsteil

Der Anweisungsteil kann optional durch das Schlüsselwort `BEGIN` eingeleitet werden. Er endet mit dem Standardbezeichner für das Bausteinende `END_xxx` (siehe Kapitel 8.2).

Jede Anweisung wird mit einem Semikolon (" ; ") abgeschlossen. Optional kann vor jeder Anweisung eine Sprungmarke (Label) stehen. Die Syntax des Anweisungsteils und der Einzelanweisungen finden Sie in Kapitel 13.

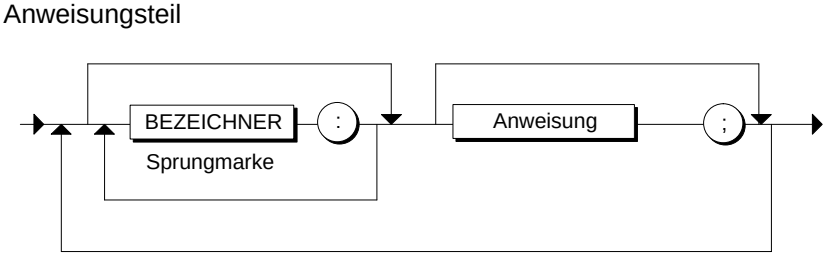


Bild 7-7 *Syntax: Anweisungsteil*

Hier ein Beispiel für den Anweisungsteil eines FB:

```
:                  //Ende Vereinbarungsteil
:
BEGIN              //BEGIN des Anweisungsteils
           X := X+1;
LABEL1 :   Y := Y+10;
           Z := X*Y;
           :
           GOTO LABEL1
LABELn :           //Ende des Anweisungsteils
END_FUNCTION_BLOCK
```

Im Anweisungsteil eines Datenbausteins können Sie über Wertzuweisungen Ihre DB-Daten mit Werten vorbesetzen. Deshalb wird der Anweisungsteil eines DB in den weiteren Kapiteln mit **Zuweisungsteil** bezeichnet.

S7-Programm

Nach dem Übersetzen werden die erzeugten Bausteine im Behälter "Bausteine" des jeweiligen S7-Programms abgelegt. Von dort aus müssen Sie sie in die CPU laden. Informationen hierzu finden Sie in **/231/** .

7.12 Kommentare

Übersicht

Kommentare dienen der Dokumentation und dem besseren Verständnis eines SCL-Bausteins. Sie sind nach dem Übersetzen für den Programmablauf ohne Bedeutung. Es gibt zwei Kommentararten:

- den Zeilenkommentar und
- den Blockkommentar

Zeilenkommentar

Der Zeilenkommentar wird mit '//' eingeleitet und erstreckt sich bis zum Ende der Zeile. Seine Länge ist begrenzt auf max. 253 Zeichen einschließlich dem Einleitungszeichen '//'. Er kann durch das folgende Syntaxdiagramm formal dargestellt werden:

Zeilenkommentar

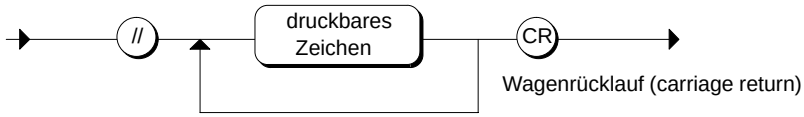


Bild 7-8 *Syntax: Zeilenkommentar*

Die druckbaren Zeichen entnehmen Sie bitte Tabelle A-2 im Anhang. Innerhalb des Zeilenkommentars sind die Zeichenpaare '(*' und '*)' bedeutungslos.

Blockkommentar

Der Blockkommentar kann über mehrere Zeilen gehen und wird als Block mit '(*' eingeleitet und mit '*)' abgeschlossen. Die Schachtelung von Blockkommentaren ist standardmäßig erlaubt. Sie können diese Einstellung jedoch ändern und die Schachtelung von Blockkommentaren unzulässig machen.

Blockkommentar

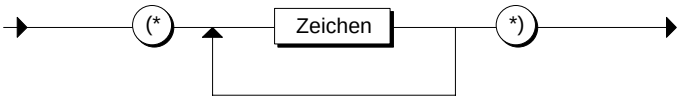


Bild 7-9 *Syntax: Blockkommentar*

Die zugelassenen Zeichen entnehmen Sie bitte Tabelle A-2 im Anhang.

Was zu beachten ist

Beachten Sie bei der Notation von Kommentaren:

- Blockkommentare in **Datenbausteinen** müssen wie Zeilenkommentare notiert werden, d.h. solche Kommentare werden auch mit '//' eingeleitet.
- Die Schachtelung von Kommentaren ist standardmäßig erlaubt. Diese Compiler–Einstellung kann aber über die Option "Geschachtelte Kommentare zulassen" geändert werden. Wählen Sie dazu den Menübefehl **Extras** ▸ **Einstellungen**, und wählen Sie im Register "Compiler" des folgenden Dialogfelds die Option ab.
- Ein Kommentar darf weder einen symbolischen Namen noch eine Konstante unterbrechen. Die Unterbrechung von Strings ist jedoch möglich.

Dieser Kommentar ist **falsch**:

```
FUNCTION_(* Anpassung*)BLOCK FB10
```

Beispiel für Einbau von Kommentaren

Im Beispiel finden Sie zwei Kommentarblöcke und einen Zeilenkommentar.

```
FUNCTION_BLOCK FB15
(* Hier steht ein Blockkommentar der
über mehrere Zeilen gehen kann*)
VAR
   SCHALTER: INT; // Zeilenkommentar
END_VAR;
BEGIN
  (* Der Variablen SCHALTER einen Wert zuweisen *)
  SCHALTER:= 3;
END_FUNCTION_BLOCK
```

Bild 7-10 Beispiel für Kommentare

Hinweis

Zeilenkommentare, die direkt hinter der Variablenvereinbarung eines Bausteins stehen, werden bei einer Rückübersetzung in ein AWL-Programm übernommen.

Diese Kommentare finden Sie innerhalb AWL im Schnittstellenbereich, d.h. im oberen Teil des Fensters (siehe auch **/231/**.)

Im Beispiel in Bild 7-10 wird also der erste Zeilenkommentar übernommen.

Aufbau einer SCL-Quelldatei

8

Übersicht

Eine SCL-Quelldatei besteht prinzipiell aus fortlaufendem Text. In einer solchen Quelldatei können Sie mehrere Bausteine programmieren. Diese können OB, FB, FC, DB oder UDT sein.

In diesem Kapitel wird der äußere Aufbau der Bausteine beschrieben. Die nachfolgenden Kapitel informieren Sie dann über die internen Strukturen.

Kapitelübersicht

8.1 Aufbau

Übersicht

Eine SCL-Quelldatei besteht aus dem Quelltext von 1 bis n Bausteinen (das können sein FB, FC, OB, DB und UDTs).

Damit sich Ihre SCL-Quelldatei in die einzelnen Bausteine übersetzen läßt, müssen Sie bestimmte Strukturen und Syntaxvorschriften dieser Bausteine beachten.

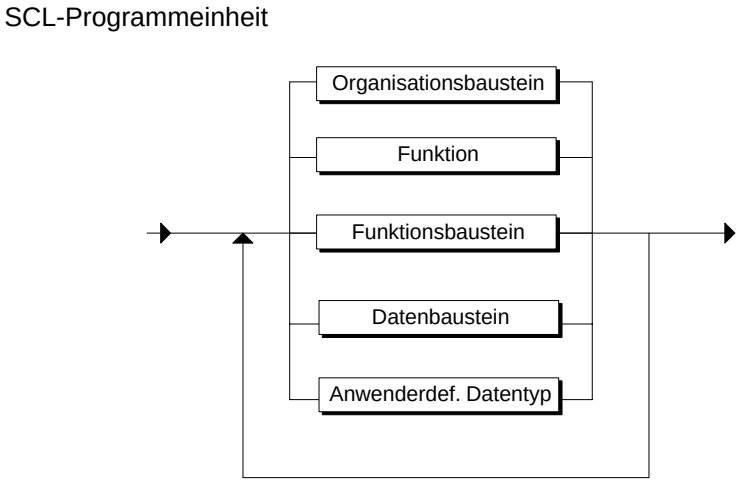


Bild 8-1 *Syntax: SCL-Programmeinheit*

Reihenfolge der Bausteine

Bezüglich der Reihenfolge der Bausteine müssen Sie bei der Erstellung der Quelldatei folgendes beachten:

Aufgerufene Bausteine stehen vor den aufrufenden Bausteinen.

Das heißt:

- Anwenderdefinierte Datentypen (UDTs) stehen vor den Bausteinen, in denen Sie verwendet werden.
- Datenbausteine mit einem zugeordneten anwenderdefiniertem Datentyp (UDT) stehen hinter dem UDT.
- Datenbausteine, auf die von allen Codebausteinen aus zugegriffen werden kann, stehen vor den Bausteinen, aus denen Sie aufgerufen werden.
- Datenbausteine mit zugeordnetem Funktionsbaustein stehen hinter dem Funktionsbaustein.
- Der Organisationsbaustein OB 1, der andere Bausteine aufruft, steht zuletzt. Bausteine, die wiederum von den aus OB 1 aufgerufenen Bausteinen aufgerufen werden, müssen vor diesen stehen.

Bausteine, die Sie in der Quelldatei aufrufen, aber nicht in derselben Quelldatei programmieren, müssen bereits zum Zeitpunkt der Dateiübersetzung im entsprechenden Anwenderprogramm vorhanden sein.

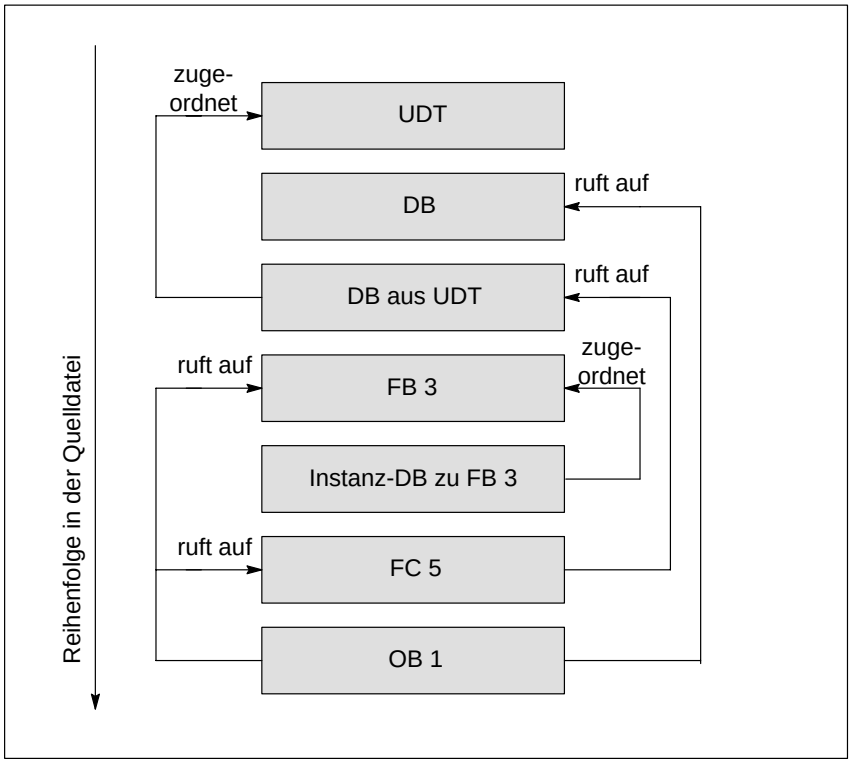


Bild 8-2 Bausteinstruktur einer Quelldatei (Beispiel)

Allgemeine Bausteinstruktur

Der Quellcode für einen Baustein besteht grundsätzlich aus folgenden Abschnitten:

- Bausteinanfang mit Angabe des Bausteins (absolut oder symbolisch)
- Bausteinattribute (optional)
- Vereinbarungsteil (unterschiedlich je nach Bausteinart)
- Anweisungsteil in Codebausteinen bzw.
  Zuweisung von Aktualwerten in Datenbausteinen (optional)
- Bausteinende

8.2 Bausteinanfang- und ende

Übersicht

Der Quelltext für einen einzelnen Baustein wird abhängig von der Bausteinart mit einem Standardbezeichner für den Anfang des Bausteins und der Bausteinbezeichnung eingeleitet; abgeschlossen wird er mit einem Standardbezeichner für das Ende des Bausteins (siehe Tabelle 8-1).

Tabelle 8-1 Standardbezeichner für Bausteinanfang- und ende

Syntax

Syntax	Bausteinart	Bezeichnung

`ORGANIZATION_BLOCK` *`ob_name`*
`:`
`END_ORGANIZATION_BLOCK`
	OB	Organisationsbaustein

`FUNCTION` *`fc_name:funktionstyp`*
`:`
`END_FUNCTION`
	FC	Funktion

`FUNCTION_BLOCK` *`fb_name`*
`:`
`END_FUNCTION_BLOCK`
	FB	Funktionsbaustein

`DATA_BLOCK` *`db_name`*
`:`
`END_DATA_BLOCK`
	DB	Datenbaustein

`TYPE` *`name udt_name`*
`:`
`END_TYPE`
	UDT	Anwenderdefinierter Datentyp

Bausteinbezeichnung

In der Tabelle 8-1 steht *xx_name* für die Bausteinbezeichnung nach der folgenden Syntax:

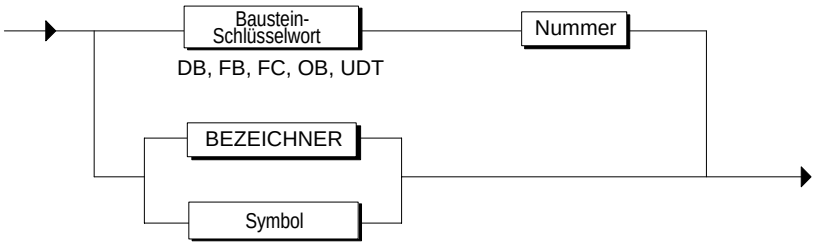


Bild 8-3 *Syntax: Bausteinbezeichnung*

Weitere Informationen finden Sie auch in Kapitel 7.5. Beachten Sie auch, daß Sie einen Bezeichner oder ein Symbol in der Symboltabelle von STEP 7 definieren müssen (siehe /**231**/).

Beispiel

```
FUNCTION_BLOCK FB10
FUNCTION_BLOCK Reglerbaustein
FUNCTION_BLOCK "Regler.B1&U2"
```

8.3 Bausteinattribute

Definition

Attribute für Bausteine können sein:

- Bausteinattribute
- Systemattribute für Bausteine.

Bausteinattribute

Überschrift, Version, Bausteinschutz, Autor, Name und Familie eines Bausteins können Sie mit Hilfe von Schlüsselwörtern angeben.

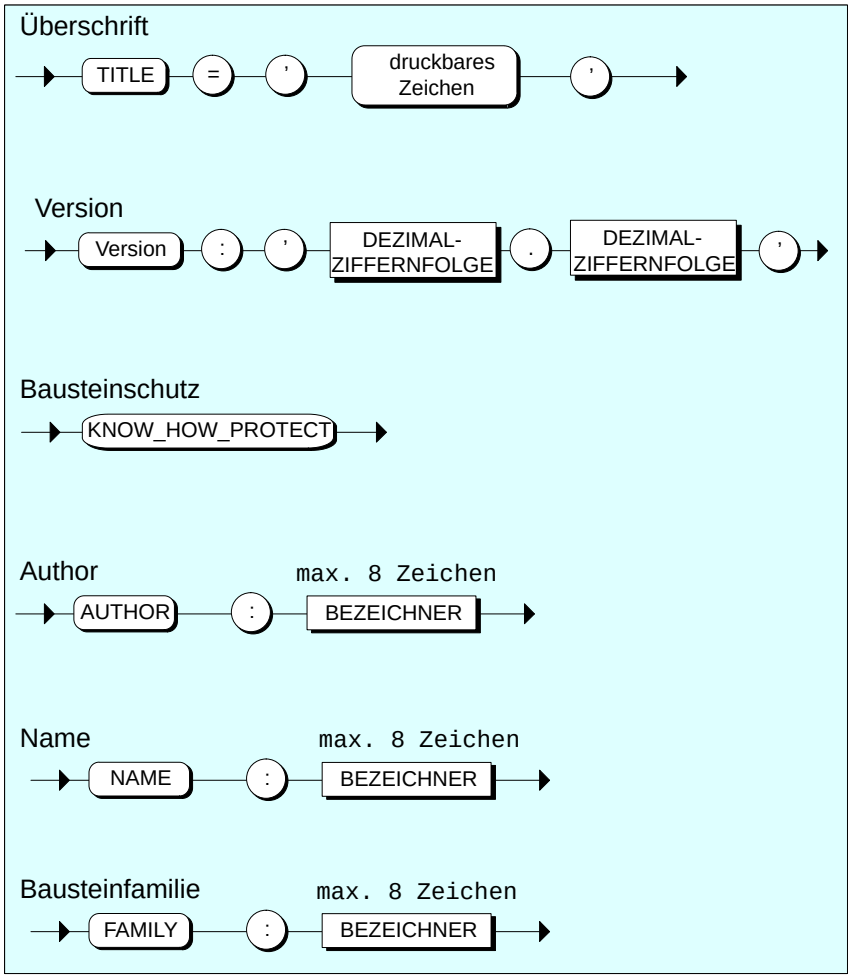


Bild 8-4 *Syntax: Bausteinattribute*

Systemattribute für Bausteine

Bausteinen können Sie außerdem noch Systemattribute, z. B. für die Leittechnikprojektierung, zuweisen.

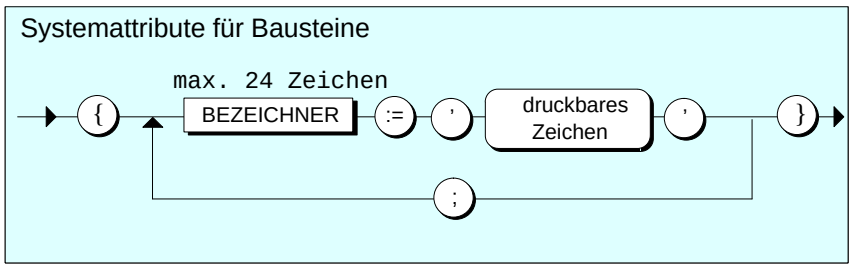


Bild 8-5 *Syntax: Systemattribute für Bausteine*

Tabelle 8-2 zeigt, welche Systemattribute für Bausteine Sie bei SCL vergeben können.

Tabelle 8-2 Systemattribute für Bausteine

Attribut	Wert	Dieses Attribut vergeben Sie, wenn	Zulässiger Bausteintyp
S7_m_c	true, false	der Baustein von einem Bedien- und Beobachtungsgerät aus bedient oder beobachtet werden soll.	FB
S7_tasklist	taskname1, taskname2, etc.	der Baustein außer in zyklischen Organisationsbausteinen auch in anderen OBs (z. B. Fehler- oder Anlauf-OBs) aufgerufen werden soll.	FB, FC
S7_block-view	big, small	der Baustein an einem Bedien- und Beobachtungsgerät in großem oder kleinem Format dargestellt werden soll.	FB, FC

Attributvergabe

Sie vergeben Bausteinattribute **nach** der Bausteinbezeichnung und **vor** dem Vereinbarungsteil.

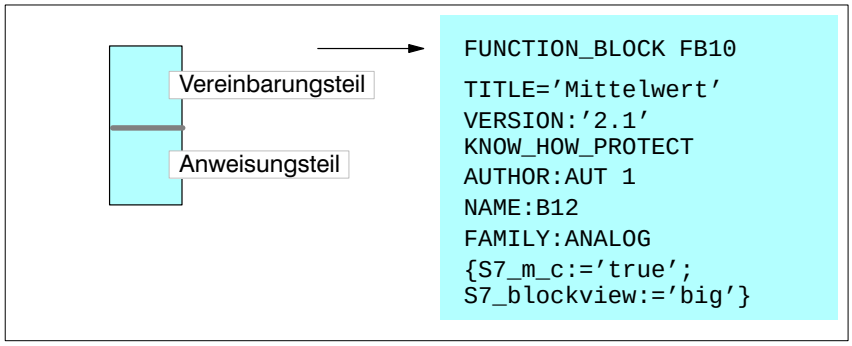


Bild 8-6 Attributvergabe

8.4 Vereinbarungsteil

Übersicht

Der Vereinbarungsteil dient zur Definition der lokalen und globalen Variablen, Parameter, Konstanten und Sprungmarken (Labels).

- Die lokalen Variablen, Parameter, Konstanten und Sprungmarken, die nur innerhalb eines Codebausteins Gültigkeit haben sollen, definieren Sie im Vereinbarungsteil des Codebausteins.
- Die globalen Daten, die von jedem Codebaustein aus ansprechbar sein sollen, definieren Sie im Vereinbarungsteil von DB.
- Im Vereinbarungsteil eines UDT legen Sie einen anwenderspezifischen Datentyp fest.

Aufbau

Ein Vereinbarungsteil gliedert sich in unterschiedliche Vereinbarungsblöcke, die jeweils durch ein eigenes Schlüsselwortpaar gekennzeichnet sind. Jeder Block enthält eine Vereinbarungsliste für gleichartige Daten wie Konstanten, Labels, statische Daten, temporäre Daten. Ein Blocktyp darf nur einmal vorkommen und ist nach der Tabelle nicht in allen Bausteinarten erlaubt. Die Reihenfolge der Blöcke dagegen ist beliebig.

Vereinbarungs-blöcke

Daten	Syntax	FB	FC	OB	DB	UDT

Konstanten	`CONST`
	Vereinbarungsliste
	`END_CONST`
		X	X	X		

Sprungmarken	`LABEL`
	Vereinbarungsliste
	`END_LABEL`
		X	X	X		

Temporäre Variable	`VAR_TEMP`
	Vereinbarungsliste
	`END_VAR`
		X	X	X		

Statische Variable	`VAR`
	Vereinbarungsliste
	`END_VAR`
		X	X[2)]		X[1)]	X[1)]

Eingangparameter	`VAR_INPUT`
	Vereinbarungsliste
	`END_VAR`
		X	X			

Ausgangsparameter	`VAR_OUTPUT`
	Vereinbarungsliste
	`END_VAR`
		X	X			

Durchgangsparameter	`VAR_IN_OUT`
	Vereinbarungsliste
	`END_VAR`
		X	X			

Vereinbarungsliste: die Liste der Bezeichner des Typs, der deklariert werden soll.						

1) *In DBs und UDTs wird das Schlüsselwort VAR und END_VAR durch STRUCT und END_STRUCT ersetzt.*

2) *Die Vereinbarung von Variablen innerhalb des Schlüselwortpaars VAR und END_VAR ist in Funktionen zwar erlaubt, die Vereinbarungen werden aber beim Übersetzen in den temporären Bereich verschoben.*

Systemattribute für Parameter

Eingangs-, Ausgangs- und Durchgangsparametern können Sie außerdem Systemattribute, z. B. für die Meldungs- oder Verbindungsprojektierung, zuweisen.

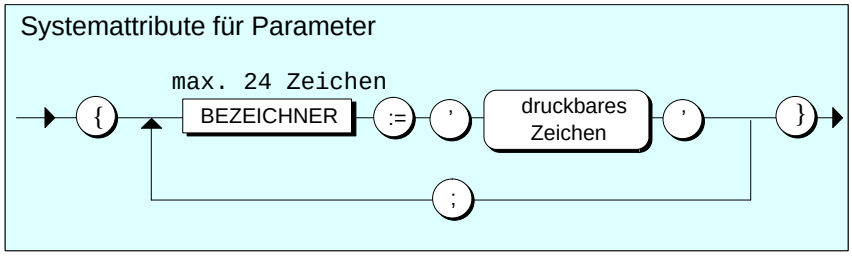


Bild 8-7 *Syntax: Systemattribute für Parameter*

Tabelle 8-3 zeigt, welche Systemattribute Sie den Parametern zuweisen können:

Tabelle 8-3 Systemattribute für Parameter

Attribut	Wert	Dieses Attribut vergeben Sie, wenn	Zulässiger Deklarationstyp
S7_server	connection, alarm_archiv	der Parameter für die Verbindungs- oder Meldungsprojektierung relevant ist. Dieser Parameter enthält die Verbindungs- bzw. die Meldungsnummer.	IN
S7_a_type	alarm, alarm_8, alarm_8p, alarm_s, notify, ar_send	Sie den Meldebausteintyp des Meldebausteins festlegen, der im Anweisungsteil aufgerufen wird (Voraussetzung: das Attribut S7_server:=alarm_archiv ist auch vergeben).	IN, nur bei FB
S7_co	pbkl, pbk, ptpl, obkl, fdl, iso, pbks, obkv	Sie den Verbindungstyp der Verbindung festlegen, die projektiert werden soll (Voraussetzung: das Attribut S7_server:=connection ist auch vergeben).	IN
S7_m_c	true, false	der Parameter von einem Bedien- und Beobachtungsgerät aus bedient oder beobachtet werden soll.	IN/OUT/ IN_OUT, nur bei FB
S7_shortcut	2 beliebige Zeichen, z. B. W, Y	dem Parameter Kurzzeichen für die Auswertung von Analogwerten zugeordnet werden sollen.	IN/OUT/ IN_OUT, nur bei FB
S7_unit	Einheit, z. B. Liter	dem Parameter Einheiten für die Auswertung von Analogwerten zugeordnet werden sollen.	IN/OUT/ IN_OUT, nur bei FB
S7_string_0	16 beliebige Zeichen, z. B. AUF	dem Parameter Text für die Auswertung von Binärwerten zugeordnet werden soll.	IN/OUT/ IN_OUT, nur bei FB, FC

Tabelle 8-3 Systemattribute für Parameter, Fortsetzung

Attribut	Wert	Dieses Attribut vergeben Sie, wenn	Zulässiger Deklarationstyp
S7_string_1	16 beliebige Zeichen, z. B. ZU	dem Parameter Text für die Auswertung von Binärwerten zugeordnet werden soll.	IN/OUT/ IN_OUT, nur bei FB, FC
S7_visible	true, false	der Parameter in CFC angezeigt werden soll oder nicht.	IN/OUT/IN_OUT, nur bei FB, FC
S7_link	true, false	der Parameter in CFC verschaltbar sein soll oder nicht.	IN/OUT/IN_OUT, nur bei FB, FC
S7_dynamic	true, false	der Parameter in CFC beim Testen dynamisierbar sein soll oder nicht.	IN/OUT/IN_OUT, nur bei FB, FC
S7_param	true, false	der Parameter in CFC parametrierbar sein soll oder nicht.	IN/IN_OUT, nur bei FB, FC

Attributvergabe

Sie vergeben die Systemattribute für Parameter in den Vereinbarungsblöcken Eingangsparameter, Ausgangsparameter bzw. Durchgangsparameter.

Beispiel:

```
VAR_INPUT
  in1  {S7_server:='alarm_archiv';
        S7_a_type:='ar_send'}:DWORD;
END_VAR
```

8.5 Anweisungsteil

Übersicht

Der Anweisungsteil beinhaltet Anweisungen[1)]

- die nach dem Aufruf eines Codebausteins zur Ausführung kommen. Diese Anweisungen dienen zur Verarbeitung von Daten und Adressen.
- zur Vorbesetzung von einzelnen Werten in Datenbausteinen.

Syntax

Bild 8-8 zeigt die Syntax des Anweisungsteils. Er besteht aus einer Wiederholung von Einzelanweisungen, wobei vor jeder Anweisung optional eine Sprungmarke (siehe Kapitel 11.6) stehen kann, die das Ziel einer Sprunganweisung ist.

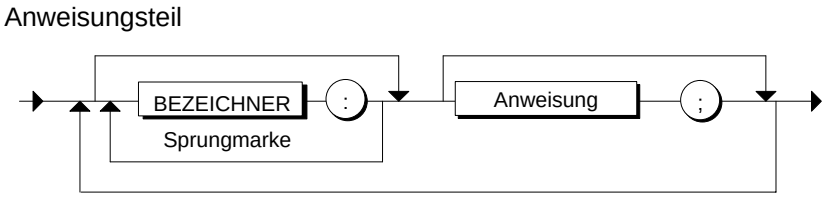


Bild 8-8 *Syntax: Anweisungsteil*

Gültige Anweisungen sind z.B.:

```
BEGIN
            ANFANGSWERT :=0;
            ENDWERT     :=200;
:
SPEICHERN:  ERGEBNIS    :=SOLLWERT;
:
```

Was ist wichtig?

Bei der Formulierung der Anweisungen müssen Sie darauf achten, daß

- der Anweisungsteil optional mit dem Schlüsselwort `BEGIN` beginnt.
- der Anweisungsteil mit dem Schlüsselwort für das Bausteinende abschließt.
- jede Anweisung mit einem Semikolon abgeschlossen wird und
- alle im Anweisungsteil verwendeten Bezeichner vereinbart sind.

1) *Wir verwenden in diesem Handbuch den Terminus "Anweisung" für alle Konstrukte, die eine ausführbare Funktion vereinbaren.*

8.6 Anweisung

Übersicht

Die einzelnen Anweisungen bestehen aus:

- **Wertzuweisungen**, die dazu dienen, einer Variablen einen Wert, das Ergebnis eines Ausdrucks, oder den Wert einer anderen Variablen zuzuweisen.
- **Kontrollanweisungen**, die dazu dienen, Anweisungen oder Gruppen von Anweisungen zu wiederholen oder innerhalb eines Programms zu verzweigen.
- **Unterprogrammbearbeitungen**, die zum Aufrufen von Funktionen und Funktionsbausteinen dienen.

Anweisung

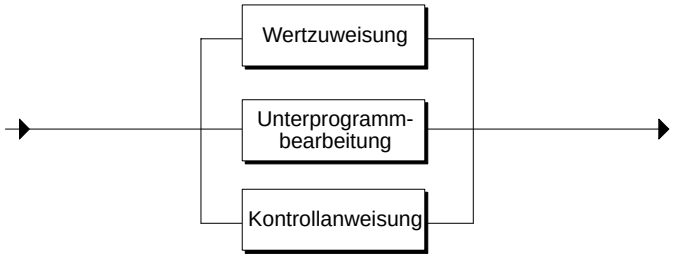


Bild 8-9 *Syntax: Anweisung*

Die Elemente, die zur Formulierung dieser Anweisungen benötigt werden, sind Ausdrücke, Operatoren und Operanden. Sie werden in den weiteren Kapiteln behandelt.

Beispiele

Die folgenden Beispiele sollen die verschiedenen Varianten der Anweisungen veranschaulichen:

```
// Beispiel für eine Wertzuweisung
      MESSWERT:= 0 ;

// Beispiel für eine Unterprogrammbearbeitung
      FB1.DB11(UEBERGABE:= 10) ;

// Beispiel für eine Kontrollanweisung
      WHILE ZAEHLER < 10 DO..
      :
      END_WHILE;
```

Beispiel 8-1 Anweisungen

8.7 Aufbau eines Funktionsbausteins (FB)

Übersicht

Ein Funktionsbaustein FB ist ein Codebaustein, der einen Teil eines Programms enthält und über einen zugeordneten Speicherbereich verfügt. Immer wenn ein FB aufgerufen wird, muß ihm ein Instanz-DB (siehe Kapitel 10) zugeordnet werden. Den Aufbau dieses Instanz-DB bestimmen Sie mit der Definition des FB-Vereinbarungsteils.

Funktionsbaustein

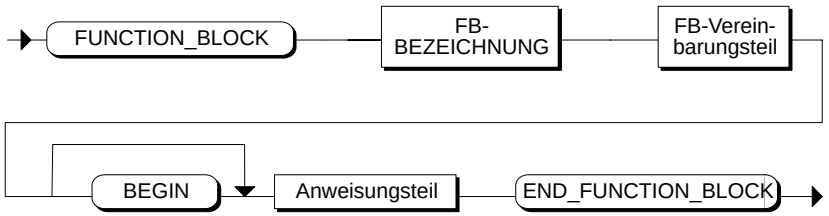


Bild 8-10 *Syntax: Funktionsbaustein (FB)*

FB-Bezeichnung

Geben Sie nach dem Schlüsselwort

```
FUNCTION_BLOCK
```

als FB-Bezeichnung das Schlüsselwort FB und dahinter die Bausteinnummer oder nur den symbolischen Namen des FB an.

Beispiele:

```
FUNCTION_BLOCK FB10
```

```
FUNCTION_BLOCK MOTOR_1
```

FB-Vereinbarungsteil

Der FB-Vereinbarungsteil dient zur Festlegung der bausteinspezifischen Daten. Die zulässigen Vereinbarungsblöcke entnehmen Sie bitte dem Kapitel 8.4. Beachten Sie, daß der Vereinbarungsteil auch den Aufbau des zugeordneten Instanz-DB bestimmt.

Beispiele:

```
CONST
   KONSTANTE:=5;
END_CONST
```

```
VAR
   WERT1,WERT2,WERT3:INT;
END_VAR
```

Beispiel

Beispiel 8-2 zeigt den Quellcode für einen Funktionsbaustein. Die Eingangs- und Ausgangsparameter (hier V1, V2) sind hier mit Anfangswerten vorbelegt.

```
FUNCTION_BLOCK FB11
      VAR_INPUT
            V1: INT:= 7;
      END_VAR
      VAR_OUTPUT
            V2: REAL:=3.14;
      END_VAR
      VAR
            DURCHLAUF_1:INT;
      END_VAR
      BEGIN
      IF V1 = 7 THEN
      DURCHLAUF_1:= V1;
      V2:= FC2 (TESTWERT:= DURCHLAUF_1);
      //Aufruf der Funktion FC2 mit
      //Parameterversorgung durch die statische
      //Variable DURCHLAUF_1
      END_IF;
END_FUNCTION_BLOCK
```

Beispiel 8-2 *Beispiel eines Funktionsbausteins*

8.8 Aufbau einer Funktion (FC)

Übersicht

Eine Funktion FC ist ein Codebaustein, dem kein eigener Speicherbereich zugeordnet ist. Eine FC benötigt daher keinen Instanz-DB. Im Gegensatz zu einem FB kann eine Funktion ein Funktionsergebnis (**Rückgabewert**) an den Aufrufpunkt zurückliefern. Die Funktion kann daher wie eine Variable in einem Ausdruck verwendet werden. Funktionen vom Typ VOID haben keinen Rückgabewert.

Funktion

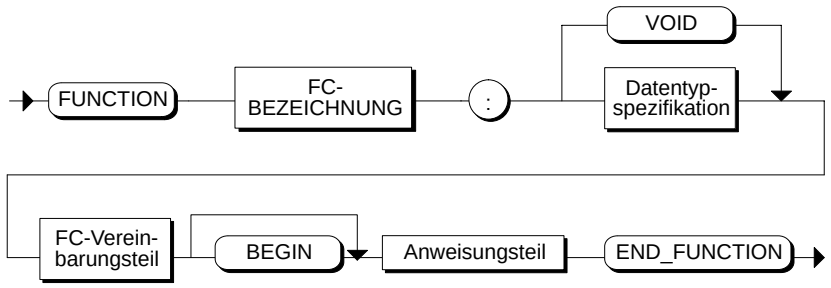


Bild 8-11 *Syntax: Funktion (FC)*

FC-Bezeichnung

Geben Sie nach dem Schlüsselwort

```
FUNCTION
```

als FC-Bezeichnung das Schlüsselwort FC und dahinter die Bausteinnummer oder nur den symbolischen Namen der FC an.

Beispiele:

```
FUNCTION FC100
```

```
FUNCTION DREHZAHL
```

Datentyp-spezifikation

Hier geben Sie den Datentyp des Rückgabewertes an. Zugelassen sind dabei alle Datentypen, die in Kapitel 9 beschrieben sind, mit Ausnahme der Datentypen STRUCT und ARRAY. Die Angabe eines Datentyps entfällt, wenn mit **VOID** auf den Rückgabewert verzichtet wird.

FC-Vereinbarungsteil

Die zulässigen Vereinbarungsblöcke entnehmen Sie bitte Kapitel 8.4.

Anweisungsteil

Innerhalb des Anweisungsteils muß dem Funktionsnamen das **Funktionsergebnis** zugewiesen werden. Eine gültige Anweisung innerhalb einer Funktion mit der Bezeichnung FC31 ist z.B.:

```
FC31:= WERT;
```

Beispiel

Das Beispiel zeigt Ihnen eine Funktion mit den formalen Eingangsparametern x1, x2, y1, y2, einem formalen Ausgangsparameter Q2 und einem Rückgabewert FC11.

Die Bedeutung der formalen Parametern finden Sie im Kapitel 10.

```
FUNCTION FC11: REAL
      VAR_INPUT
            x1: REAL;
            x2: REAL;
            y1: REAL;
            y2: REAL;
      END_VAR
      VAR_OUTPUT
            Q2: REAL;
      END_VAR
      BEGIN       // Anweisungsteil
      FC11:= SQRT // Rückgabe des Funktions-
                  // wertes
      ( (x2 - x1)**2 + (y2 - y1) **2 );
      Q2:= x1;
END_FUNCTION
```

Beispiel 8-3 *Beispiel einer Funktion*

8.9 Aufbau eines Organisationsbausteins (OB)

Übersicht

Der Organisationsbaustein OB ist wie ein FB oder FC Teil des Anwenderprogramms und wird vom Betriebssystem zyklisch oder bei bestimmten Ereignissen aufgerufen. Er bildet die Schnittstelle zwischen Anwenderprogramm und Betriebssystem.

Organisationsbaustein

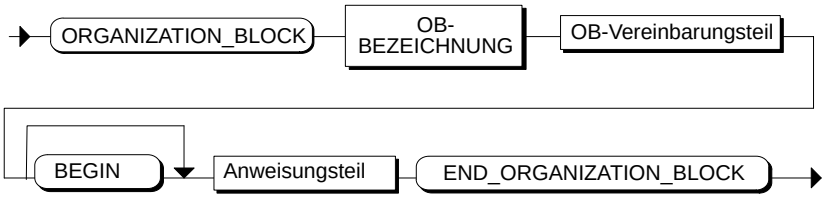


Bild 8-12 *Syntax: Organisationsbaustein (OB)*

OB-Bezeichnung

Geben Sie nach dem Schlüsselwort

```
ORGANIZATION_BLOCK
```

als OB-Bezeichnung das Schlüsselwort OB und dahinter die Bausteinnummer oder einfach den symbolischen Namen des OB an.

Beispiele:

```
ORGANIZATION_BLOCK OB14
```

```
ORGANIZATION_BLOCK UHRZEITALARM
```

OB-Vereinbarungsteil

Jeder OB benötigt zum Ablauf grundsätzlich **20 byte Lokaldaten** für seine Startinformation. Sie können je nach den Anforderungen des Programms in dem OB zusätzliche temporäre Variablen deklarieren. Die Beschreibung der 20 byte Lokaldaten entnehmen Sie bitte **/235/** .

Beispiel:

```
ORGANIZATION_BLOCK OB14
//UHRZEITALARM

VAR_TEMP
HEADER:ARRAY [1..20] OF BYTE;//20 byte für Startinfo
:
:
END_VAR
```

Die übrigen zulässigen Vereinbarungsblöcke für OBs entnehmen Sie bitte Kapitel 8.4.

8.10 Aufbau eines Datenbausteins (DB)

Übersicht

Der Datenbaustein DB enthält globale anwenderspezifische Daten, auf die **alle** Bausteine im Programm zugreifen sollen. Jeder FB, FC oder OB kann aus diesen DB lesen oder schreiben. Den Aufbau von Datenbausteinen, die nur bestimmten FBs zugeordnet sind, (die Instanz-DB) finden Sie in Kapitel 12.

Datenbaustein

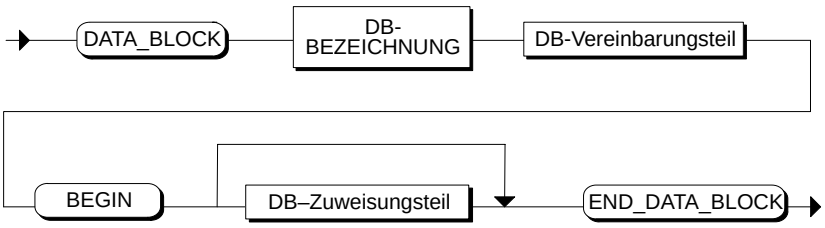


Bild 8-13 *Syntax: Datenbaustein (DB)*

DB-Bezeichnung

Geben Sie nach dem Schlüsselwort

```
DATA_BLOCK
```

als DB-Bezeichnung das Schlüsselwort DB und dahinter die Bausteinnummer oder einfach den symbolischen Namen des DB an.

Beispiel

```
DATA_BLOCK DB20
DATA_BLOCK MESSBEREICH
```

DB-Vereinbarungsteil

Im DB-Vereinbarungsteil definieren Sie die Datenstruktur des DB. Einer DB-Variablen kann entweder ein strukturierter Datentyp (STRUCT) oder ein anwenderdefinierter Datentyp (UDT) zugeordnet werden.

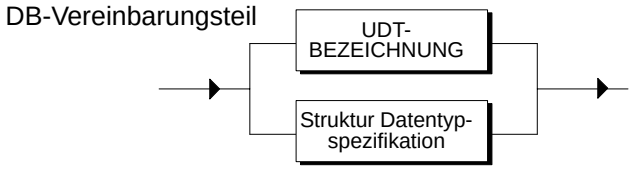


Bild 8-14 *Syntax: DB-Vereinbarungsteil*

Beispiel:

```
DATA_BLOCK DB20
           STRUCT      // Vereinbarungsteil
             WERT:ARRAY [1..100] OF INT;
           END_STRUCT

BEGIN                  // Anfang Zuweisungsteil
:

END_DATA_BLOCK         // Ende Datenbaustein
```

DB-Zuweisungsteil

Sie können die Daten, die Sie im Vereinbarungsteil vereinbart haben, für Ihren speziellen Anwendungsfall mit einzelnen DB-spezifischen Werten anpassen. Der Zuweisungsteil beginnt mit dem Schlüsselwort:

```
BEGIN
```

und besteht dann aus einer Folge von Wertzuweisungen mit folgender Syntax:

DB-Zuweisungsteil

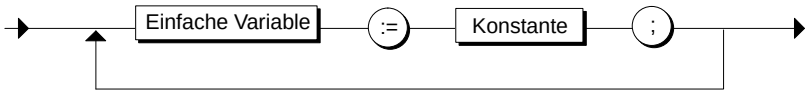


Bild 8-15 *Syntax: DB-Zuweisungsteil*

Hinweis

Bei der Vergabe von Anfangswerten (Initialisierung), der Angabe von Attributen und der Angabe von Kommentaren innerhalb eines DB gilt die Syntax von AWL. Informationen über die Schreibweisen der Konstanten, Attribute und Kommentare können Sie dem Benutzerhandbuch **/231/** oder dem Handbuch **/232/** entnehmen.

Beispiel

Das nachstehende Beispiel zeigt Ihnen, wie der Zuweisungsteil formuliert werden kann, wenn die Feldwerte [1] und [5] den Integer-Wert 5 und –1 statt der Vorbelegung 1 haben sollen.

```
DATA_BLOCK  DB20
STRUCT            //Datenvereinbarung mit
                  //Vorbelegung
      WERT  : ARRAY [ 1..100] OF INT := 100 (1);
      MERKER: BOOL    := TRUE;
      S_WORT: WORD    := W#16#FFAA;
      S_BYTE: BYTE    := B#16#FF;
      S_TIME: S5TIME  := S5T#1h30m30s;
END_STRUCT

BEGIN             //Zuweisungssteil
  //Wertzuweisung für bestimmte Feldelemente
  WERT [1]  := 5;
  WERT [5]  :=-1;
END_DATA_BLOCK
```

Beispiel 8-4 Zuweisungsteil eines DB

8.11 Aufbau eines anwenderdefinierten Datentyps (UDT)

Übersicht

Anwenderdefinierte Datentypen UDT sind von Ihnen erzeugte spezielle Datenstrukturen. Da anwenderdefinierte Datentypen einen Namen haben, sind sie mehrfach einsetzbar. Sie sind nach Ihrer Definition im gesamten Anwenderprogramm verwendbar und somit globale Datentypen. Sie können deshalb diese Datentypen:

- wie elementare oder zusammengesetzte Datentypen in Bausteinen verwenden oder
- als Vorlage für die Erstellung von Datenbausteinen mit gleicher Datenstruktur benutzen.

Anwenderdefinierter Datentyp

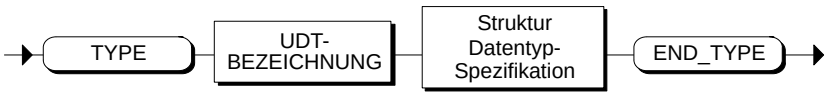


Bild 8-16 *Syntax: Anwenderdefinierter Datentyp (UDT)*

UDT-Bezeichnung

Geben Sie nach dem Schlüsselwort

```
TYPE
```

das Schlüsselwort UDT und dahinter eine Nummer oder einfach den symbolischen Namen des UDT an.

Beispiele:

```
TYPE UDT10
TYPE VERSORGUNGSBLOCK
```

Spezifikation der Datentypen

Die Spezifikation des Datentyps erfolgt immer mit Hilfe einer **STRUCT-Datentypspezifikation**. Der Datentyp UDT kann in den Vereinbarungsblöcken von Codebausteinen oder in Datenbausteinen verwendet bzw. DB zugeordnet werden. Die zulässigen Vereinbarungsblöcke und weitere Informationen entnehmen Sie bitte dem Kapitel 9.

Datentypen

9

Übersicht

Ein Datentyp ist die Zusammenfassung von Wertebereichen und Operationen zu einer Einheit. SCL besitzt, wie die meisten anderen Programmiersprachen vordefinierte (d. h. in die Sprache eingebaute) Datentypen. Darüberhinaus kann der Programmierer zusammengesetzte und anwenderdefinierte Datentypen bilden.

Kapitelübersicht

9.1 Übersicht

Übersicht

Tabelle 9-1 zeigt die unterschiedlichen Datentypen in SCL:

Tabelle 9-1 Die Datentypen in SCL

Elementare Datentypen			
BOOL	CHAR	INT	TIME
BYTE		DINT	DATE
WORD		REAL	TIME_OF_DAY
DWORD			S5TIME
Zusammengesetzte Datentypen			
DATE_AND_TIME	STRING	ARRAY	STRUCT
Anwenderdefinierte Datentypen			
UDT			
Parametertypen			
TIMER	BLOCK_FB	POINTER	ANY
COUNTER	BLOCK_FC		
	BLOCK_DB		
	BLOCK_SDB		

Diese Datentypen bestimmen:

- die Art und Bedeutung der Datenelemente
- die zulässigen Bereiche der Datenelemente
- die zulässige Menge der Operationen, die mit einem Operanden eines Datentyps ausgeführt werden können
- die Schreibweise der Daten dieses Datentyps.

9.2 Elementare Datentypen

Übersicht

Elementare Datentypen definieren die Struktur von Daten, die nicht in kleinere Einheiten zerlegt werden können. Sie entsprechen der Definition der Norm DIN EN 1131-3. Ein elementarer Datentyp beschreibt einen Speicherbereich mit fester Länge und steht für Bit-, Integer, Real, Zeitdauer, Uhrzeit und Zeichengrößen. Diese Datentypen sind alle in SCL vordefiniert.

Tabelle 9-2 Bitbreiten und Wertebereiche der elementaren Datentypen

Typ	Schlüsselwort	Bitbreite	Wertebereich
Bitdatentyp	Daten dieses Typs belegen entweder 1 bit (Datentyp BOOL), 8 bit, 16 bit oder 32 bit.		
Bit	BOOL	1	0, 1 oder FALSE, TRUE
Byte	BYTE	8	Ein numerischer Wertebereich ist nicht angebbar. Es handelt sich um Bitkombinationen, mit denen keine numerischen Ausdrücke gebildet werden können.
Wort	WORD	16	
Doppelwort	DWORD	32	
Zeichentyp	Daten dieses Typs belegen genau 1 Zeichen des ASCII-Zeichensatzes		
Einzelzeichen	CHAR	8	erweiterter ASCII-Zeichensatz
Numerische Typen	Sie stehen für die Verarbeitung numerischer Werte zur Verfügung.		
Integer (Ganzzahl)	INT	16	-32_768 bis 32_767
Doppelinteger	DINT	32	-2_147_483_648 bis 2_147_483_647
Gleitpunktzahl (IEE Gleitpunktz.)	REAL	32	-3.402822E+38 bis -1.175495E-38, 0.0, +1.175495E-38 bis 3.402822E+38
Zeittypen	Daten dieses Typs repräsentieren die unterschiedlichen Zeit-/und Datumswerte in STEP 7.		
S5-Zeit	S5TIME	16	T#0H_0M_0S_10MS bis T#2H_46M_30S
Zeitdaten: IEC-Zeit in Schritten von 1 ms	TIME (=DURATION)	32	-T#24D_20H_31M_23S_647MS bis T#24D_20H_31M_23S_647MS
Datum: IEC-Datum in Schritten von 1 Tag	DATE	16	D#1990-01-01 bis D#2168-12-31
Tageszeit: Uhrzeit in Schritten von 1 ms	TIME_OF_DAY (=TOD)	32	TOD#0:0:0 bis TOD#23:59:59.999

Hinweis zu S5-Zeit: Je nach Zeitbasis – 0.01S, 0.1S, 1S oder 10S – ist die Auflösung des Zeitwerts begrenzt. Der Compiler rundet die Werte entsprechend.

9.3 Zusammengesetzte Datentypen

Übersicht

SCL unterstützt folgende zusammengesetzte Datentypen:

Tabelle 9-3 Zusammengesetzte Datentypen

Datentyp	Beschreibung
DATE_AND_TIME DT	Definiert einen Bereich mit 64 bit (8 byte). Dieser Datentyp speichert (in binärcodiertem Dezimalformat) Datum und Uhrzeit und ist in SCL bereits vordefiniert.
STRING	Definiert einen Bereich für eine Zeichenfolge von maximal 254 Zeichen (Datentyp CHAR).
ARRAY	Definiert ein Feld aus Elementen eines Datentyps (entweder elementar oder zusammengesetzt).
STRUCT	Definiert eine Gruppierung von beliebig kombinierten Datentypen. Sie können ein Feld aus Strukturen oder auch eine Struktur aus Strukturen und Feldern definieren.

9.3.1 Datentyp DATE_AND_TIME

Übersicht

Der Datentyp DATE_AND_TIME setzt sich zusammen aus den Datentypen DATE und TIME. Er definiert einen Bereich mit 64 bit (8 bytes) für die Angabe von Datum und Uhrzeit. Der Datenbereich speichert die folgenden Informationen: Jahr–Monat–Tag–Stunden: Minuten: Sekunden.Millisekunden

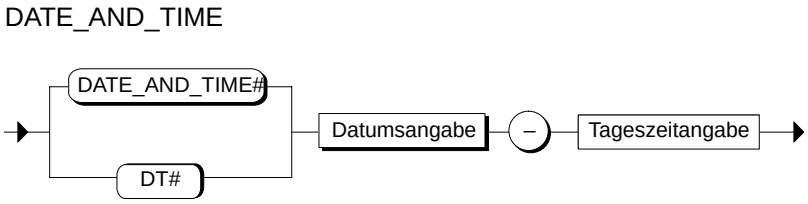


Bild 9-1 *Syntax: DATE_AND_TIME*

Wertebereich

Tabelle 9-4 Bitbreite und Wertebereich

Typ	Schlüsselwort	Bit-breite	Wertebereich
Datum und Uhr-zeit	DATE_AND_TIME (=DT)	64	`DT#1990-01-01-0:0:0.0` bis `DT#2089-12-31-23:59:59.999`

Die genaue Syntax für die Datums- und Tageszeitangabe finden Sie in Kapitel 11 dieser Beschreibung. Eine gültige Definition für 20.10.1995 12 Uhr 20 Minuten 30 Sekunden und 10 Millisekunden ist:

```
DATE_AND_TIME#1995–10–20–12:20:30.10
DT#1995–10–20–12:20:30.10
```

Hinweis

Um gezielt auf die Komponenten DATE oder TIME zuzugreifen, stehen Standard-FCs zur Verfügung.

9.3.2 Datentyp STRING

Übersicht

Der Datentyp STRING definiert eine Zeichenkette von maximal 254 Einzelzeichen.

Der Standardbereich, der für eine Zeichenkette reserviert ist, besteht aus 256 Bytes. Dies ist der Platz, der benötigt wird, um 254 Zeichen und einen Kopf von 2 Byte zu speichern.

Sie können den Speicherplatz für eine Zeichenkette verringern, indem Sie auch die maximale Anzahl der Zeichen definieren, die in der Zeichenkette gespeichert werden sollen. Ein *Nullstring*, d.h. ein String ohne Inhalt, stellt den kleinstmöglichen Wert dar.

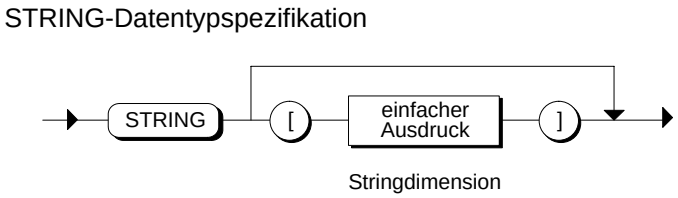


Bild 9-2 *Syntax: STRING-Datentypspezifikation*

Der einfache Ausdruck (Stringdimension) steht für die maximale Anzahl der Zeichen in der Zeichenkette.

Einige gültige STRING-Typen sind:

```
STRING[10]
STRING[3+4]
STRING[3+4*5]
STRING      max. Wertebereich (Default ≙ 254 Zeichen)
```

Wertebereich

In einer Zeichenkette sind alle Zeichen des ASCII-Codes zugelassen. In Kapitel 11 ist beschrieben, wie Steuerzeichen und nichtdruckbare Zeichen behandelt werden.

Hinweis

Bei Rückgabewerten einer FC sowie bei Ausgangs- und Durchgangsparametern kann die Standardlänge des Datentyps STRING von 254 Zeichen auf eine beliebige Anzahl Zeichen reduziert werden, um die Ressourcen Ihrer CPU besser zu nutzen. Wählen Sie den Menübefehl **Einstellungen** im Menü **Extras**, und im folgenden Dialogfeld das Register “Compiler”. Tragen Sie dann bei der Option “Maximale Stringlänge” die gewünschte Anzahl von Zeichen ein.

9.3.3 Datentyp ARRAY

Übersicht

Der Datentyp ARRAY umfaßt eine festgelegte Anzahl von Komponenten eines einzigen Datentyps. Im Syntaxdiagramm 9-3 für ARRAYs wird dieser Datentyp nach dem reservierten Wort `OF` genauer spezifiziert. SCL unterscheidet:

- den eindimensionale ARRAY-Typ
  (Liste von Datenelementen, die in aufsteigender Reihenfolge angeordnet sind.)
- den zweidimensionalen ARRAY-Typ
  (Tabelle von Daten, die aus Zeilen und Spalten besteht. Die erste Dimension bezieht sich auf die Zeilennummer, die zweite auf die Spaltennummer.)
- den höherdimensionalen ARRAY-Typ
  (Erweiterung des zweidimensionalen ARRAY-Typs um weitere Dimensionen. Die Anzahl der maximal zulässigen Dimensionen beträgt 6.)

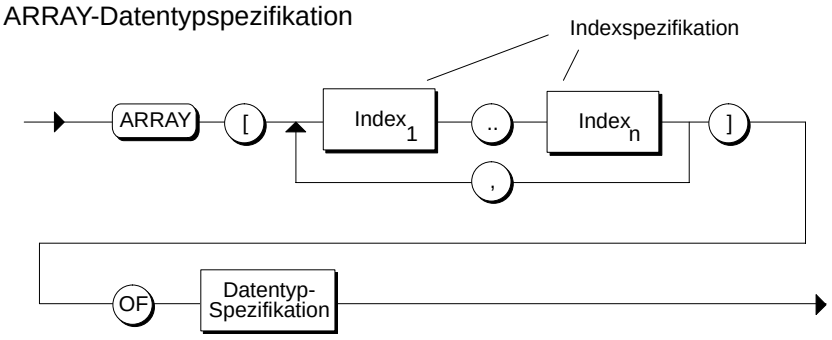


Bild 9-3 *Syntax: ARRAY-Datentypspezifikation*

Indexspezifikation

Sie beschreibt die Dimensionen des Feldes mit:

- dem kleinst- und größtmöglichen Index (Index-Bereich) für jede Dimension. Der Index kann ein beliebiger ganzzahliger Wert (-32768 bis 32767) sein.
- Die Grenzen müssen durch zwei Punkte getrennt angegeben werden.
- Die einzelnen Index-Bereiche werden durch Kommata getrennt und die gesamte Indexspezifikation in eckigen Klammern eingeschlossen.

Datentyp-spezifikation

Mit der Datentypspezifikation deklarieren Sie den Datentyp der Feldkomponenten. Als Datentypen sind alle in diesem Kapitel aufgeführten Möglichkeiten zugelassen. Der Datentyp eines ARRAY kann auch eine Struktur sein.

Die folgenden Spezifikationen sind mögliche ARRAY-Typen:

```
ARRAY[1..10] OF REAL
ARRAY[1..10] OF STRUCT..END_STRUCT
ARRAY[1..100, 1..10] OF REAL
```

9.3.4 Datentyp STRUCT

Übersicht

Ein Datentyp STRUCT beschreibt einen Bereich der aus einer festen Anzahl von Komponenten besteht, deren Datentypen verschieden sein dürfen. Im Syntaxdiagramm 9-4 werden diese Datenelemente unmittelbar nach dem Schlüsselwort STRUCT in der Komponentendeklaration angegeben.

Insbesondere kann ein Datenelement des Datentyps STRUCT selbst wieder strukturiert sein. Damit ist die Schachtelung von Strukturen zulässig.

In Kapitel 10 erfahren Sie, wie Sie auf die Daten einer Struktur zugreifen können.

STRUCT-Datentypspezifikation

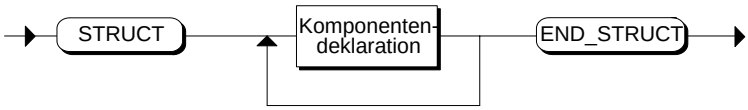


Bild 9-4 *Syntax: STRUCT-Datentypspezifikation*

Komponenten-deklaration

Das ist eine Auflistung der verschiedenen Komponenten zu einer Struktur. Nach dem Syntaxdiagramm 9-5 besteht diese Auflistung aus

- 1 bis n Bezeichnern
- dem zugeordneten Datentyp
- einer optionellen Vorbesetzung mit Anfangswerten.

Komponentendeklaration

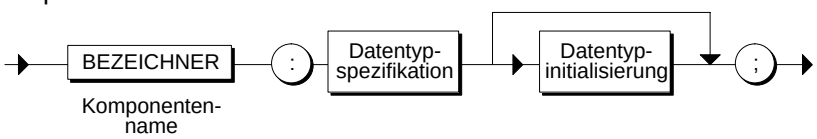


Bild 9-5 *Syntax: Komponentendeklaration*

Bezeichner

Das ist der Name eines Strukturelements für das die nachfolgende Datentypspezifikation gelten soll.

Datentyp-initialisierung

Sie können optionell nach der Datentypspezifikation ein einzelnes Strukturelement mit einem Anfangswert belegen. Diese Zuordnung erfolgt über eine Wertzuweisung und ist in Kapitel 10 beschrieben.

Beispiel

Das Beispiel zeigt ihnen eine mögliche Definition eines STRUCT-Datentyps.

```
STRUCT
 //ANFANG Komponentendeklaration
  A1       :INT;
  A2       :STRING[254];
  A3       :ARRAY [1..12] OF REAL;
   |        |
   Komponentennamen  Datentypspezifikationen
//ENDE Komponentendeklaration
END_STRUCT
```

Beispiel 9-1 Definition eines STRUCT-Datentyps

9.4 Anwenderdefinierter Datentyp (UDT)

Übersicht

Einen Datentyp UDT definieren Sie als Baustein (siehe Kapitel 8). Dieser Datentyp ist nach seiner Definition im gesamten Anwenderprogramm verwendbar und somit ein globaler Datentyp. Sie können diese Datentypen mit seiner UDT-Bezeichnung UDTx (x steht für eine Nummer) oder unter einem zugeordneten symbolischen Namen im Vereinbarungsteil eines Bausteins oder Datenbausteins verwenden.

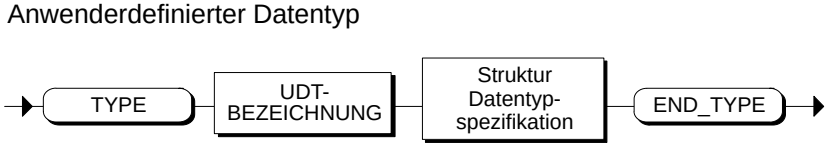


Bild 9-6 *Syntax: Anwenderdefinierter Datentyp (UDT)*

UDT-Bezeichnung

Eine Vereinbarung eines UDT wird durch das Schlüsselwort TYPE eingeleitet, gefolgt vom Namen des UDT (UDT-Bezeichner). Der Name des UDT kann entweder absolut, d.h. durch einen Standardnamen der Form UDTx (x ist eine Nummer) angegeben werden, oder es kann ein symbolischer Name verwendet werden (siehe auch Kapitel 8).

Datentyp-spezifikation

Nach der UDT-Bezeichnung folgt die Spezifikation des Datentyps. Hier ist nur die STRUCT-Datentypspezifikation (siehe Kapitel 9.3.4) zugelassen:

```
STRUCT
:
END_STRUCT
```

Anschließend wird die gesamte Vereinbarung eines UDT mit dem Schlüsselwort

```
END_TYPE
```

abgeschlossen.

Verwendung eines UDT

Der so definierte Datentyp kann zur Verwendung von Variablen oder Parametern, sowie zur Vereinbarung von DB verwendet werden. Es können auch Komponenten von Strukturen oder Feldern, auch innerhalb anderer UDT, mit Hilfe des UDT vereinbart werden

---

Hinweis

Bei der Vergabe von Anfangswerten (Initialisierung) innerhalb eines UDT gilt die Syntax von AWL. Informationen über die Schreibweisen der Konstanten können Sie dem Benutzerhandbuch **/231/** oder dem Handbuch **/232/** entnehmen.

Beispiel

Das Beispiel zeigt Ihnen eine Definition eines UDT und die Verwendung dieses Datentyps innerhalb einer Variablenerklärung. Es wird vorausgesetzt, daß in der Symboltabelle der Name "MESSWERTE" für UDT50 vereinbart wurde.

```
TYPE MESSWERTE    // UDT Definition
 STRUCT
  BIPOL_1 : INT;
  BIPOL_2 : WORD   := W#16#AFA1;
  BIPOL_3 : BYTE   := B#16#FF;
  BIPOL_4 : WORD   := B#(25,25);
  BIPOL_5 : INT    := 25;
  S_TIME  : S5TIME := S5T#1h20m10s;
  MESSUNG: STRUCT
            BIPOLAR_10V    : REAL;
            UNIPOLAR_4_20MA : REAL;
           END_STRUCT;
 END_STRUCT
END_TYPE
```

```
FUNCTION_BLOCK FB11
VAR
      MESS_BEREICH: MESSWERTE;
END_VAR
BEGIN
//...
      MESS_BEREICH.BIPOL:= -4;
      MESS_BEREICH.MESSUNG.UNIPOLAR_4_20MA:= 2.7;
//...
END_FUNCTION_BLOCK
```

Beispiel 9-2 Vereinbarung von anwenderdefinierten Datentypen

9.5 Parametertypen

Übersicht

Neben elementaren, zusammengesetzten und anwenderdefinierten Datentypen können Sie für das Festlegen der formalen Bausteinparameter von FBs und FCs sogenannte **Parametertypen** verwenden. Diese Datentypen dienen dazu

- Zähler-/Zeitfunktionen als Parameter zu vereinbaren (`TIMER/COUNTER`)
- FC, FB, DB und SDB als Parameter zu vereinbaren (`BLOCK_xx`)
- einen Operanden beliebigen Datentyps als Parameter zuzulassen (`ANY`)
- einen Speicherbereich als Parameter zuzulassen (`POINTER`).

Tabelle 9-5 Parametertypen

Parameter	Größe	Beschreibung
TIMER	2 byte	Kennzeichnet ein bestimmtes Zeitglied, das von dem Programm in dem aufgerufenen Codebaustein verwendet werden soll. Aktualparameter: `z. B. T1`
COUNTER	2 byte	Kennzeichnet einen bestimmten Zähler, der von dem Programm in dem aufgerufenen Codebaustein verwendet werden soll. Aktualparameter: `z. B. Z10`
BLOCK_FB BLOCK_FC BLOCK_DB BLOCK_SDB	2 byte	Kennzeichnet einen bestimmten Baustein, der von dem Programm in dem aufgerufenen Codebaustein verwendet werden soll. Aktualparameter: `z. B. FC101` `DB42`
ANY	10 byte	Wird verwendet, wenn als Datentyp des Aktualparameters ein beliebiger Datentyp erlaubt sein soll.
POINTER	6 byte	Kennzeichnet einen bestimmten Speicherbereich, der von dem Programm verwendet werden soll. Aktualparameter: `z. B. M50.0`

TIMER und COUNTER

Sie legen ein bestimmtes Zeitglied oder einen bestimmten Zähler fest, der bei der Bearbeitung eines Bausteins verwendet werden soll. Die Datentypen `TIMER` und `COUNTER` sind nur für Eingangsparameter (`VAR_INPUT`) erlaubt.

BLOCK-Typen

Sie legen einen bestimmten Baustein fest, der als Eingangsparameter verwendet werden soll. Die Vereinbarung des Eingangsparameters bestimmt die Bausteinart (FB, FC, DB). Bei der Parameterversorgung geben Sie die Baustein-Bezeichnung, entweder absolut (z. B. FB20), oder durch einen symbolischen Namen an.

SCL stellt keine Operationen auf diese Datentypen zur Verfügung. Es können lediglich Parameter dieses Types bei Bausteinaufrufen versorgt werden. Bei FCs ist das Durchreichen eines Eingangsparameters nicht möglich.

In SCL können Sie Operanden der folgenden Datentypen als Aktualparameter zuordnen:

- Funktionsbausteine ohne Formalparameter
- Funktionen ohne Formalparameter und Rückgabewert (VOID)
- Datenbausteine und Systemdatenbausteine.

ANY

In SCL besteht die Möglichkeit Bausteinparameter vom Datentyp `ANY` zu vereinbaren, wobei beim Aufruf eines solchen Bausteins diese Parameter mit Operanden beliebigen Datentyps versorgt werden dürfen. SCL bietet jedoch nur eine Verarbeitungsmöglichkeit des Datentyps `ANY` an: das Weiterreichen an unterlagerte Bausteine.

In SCL können Sie Operanden der folgenden Datentypen als Aktualparameter zuordnen:

- Elementare Datentypen:
  Sie geben die absolute Adresse oder den symbolischen Namen des Aktualparameters an.
- Zusammengesetzte Datentypen:
  Sie geben den symbolischen Namen der Daten mit zusammengesetztem Datentyp an (z. B. Felder und Strukturen).
- Datentyp `ANY`:
  Dies ist nur möglich, wenn der Operand ein Formalparameter verträglicher Parameterart ist.
- Datentyp `NIL`:
  Sie geben einen Nullpointer an.
- Zeiten, Zähler und Bausteine:
  Sie geben den Bezeichner an (z. B. T1, Z20 oder FB6).

Der Datentyp `ANY` ist für formale Eingangsparameter, Durchgangsparameter von FBs und FCs und für Ausgangsparameter von FCs erlaubt.

Hinweis

Wenn Sie beim Aufruf eines FB oder einer FC einen Formalparameter vom Typ ANY mit einer temporären Variablen versorgen, dürfen Sie diesen Parameter in dem aufgerufenen Baustein nicht an einen weiteren Baustein durchreichen. Die Adressen der temporären Variaben verlieren beim Durchreichen Ihre Gültigkeit.

POINTER

In SCL besteht die Möglichkeit Bausteinparameter vom Datentyp `POINTER` zu vereinbaren, wobei beim Aufruf eines solchen Bausteins diese Parameter mit Operanden beliebigen Datentyps versorgt werden dürfen. SCL bietet jedoch nur eine Verarbeitungsmöglichkeit des Datentyps `POINTER` an: das Weiterreichen an unterlagerte Bausteine.

In SCL können Sie Operanden der folgenden Datentypen als Aktualparameter zuordnen:

- Elementare Datentypen:
  Sie geben die absolute Adresse oder den symbolischen Namen des Aktualparameters an.
- Zusammengesetzte Datentypen:
  Sie geben den symbolischen Namen der Daten mit zusammengesetztem Datentyp an (z. B. Felder und Strukturen).
- Datentyp `POINTER`:
  Dies ist nur möglich, wenn der Operand ein Formalparameter verträglicher Parameterart ist.
- Datentyp `NIL`:
  Sie geben einen Nullpointer an.

Der Datentyp `POINTER` ist für formale Eingangsparameter, Durchgangsparameter von FBs und FCs und für Ausgangsparameter von FCs erlaubt.

Hinweis

Wenn Sie beim Aufruf eines FB oder einer FC einen Formalparameter vom Typ POINTER mit einer temporären Variablen versorgen, dürfen Sie diesen Parameter in dem aufgerufenen Baustein nicht an einen weiteren Baustein durchreichen. Die Adressen der temporären Variaben verlieren beim Durchreichen Ihre Gültigkeit.

Beispiele

```
FUNCTION ABSTAND: REAL
      VAR_INPUT
            MeinDB:BLOCK_DB;
            ZEIT  : TIMER;
      END_VAR
      VAR
            INDEX: INTEGER;
      END_VAR
      BEGIN
      MeinDB.DB5:=5;
      ABSTAND:=....     // RETURNWERT
END_FUNCTION
```

Beispiel 9-3 Datentypen BLOCK_DB und TIMER

```
FUNCTION FC100: VOID
      VAR_IN_OUT
            in, out:ANY;
      END_VAR
      VAR_TEMP
            ret: INT;
      END_VAR
      BEGIN
      //...
      ret:=SFC20(DSTBLK:=out,SCRBLK:=in);
      //...
END_FUNCTION

FUNCTION_BLOCK FB100
      VAR
            ii:INT;
            aa, bb:ARRAY[1..1000] OF REAL;
      END_VAR
      BEGIN
      //...
      FC100(in:=aa, out:=bb);
      //...
END_FUNCTION_BLOCK
```

Beispiel 9-4 Datentyp ANY

Vereinbarung lokaler Variablen und Bausteinparameter

10

Übersicht

Lokale Variablen und Bausteinparameter sind Daten, die innerhalb eines Codebausteins (FC, FB, OB) vereinbart werden und nur für diesen Codebaustein Gültigkeit haben. Dieses Kapitel informiert Sie über die Vereinbarung und die Initialisierung dieser Daten.

Kapitelübersicht

10.1 Übersicht

Einteilung der Variablen

Tabelle 10-1 zeigt, in welche Kategorien sich die lokalen Variablen einteilen lassen:

Tabelle 10-1 Lokale Variablen

Variable	Bedeutung
Statische Variablen	Statische Variablen sind lokale Variablen, deren Wert über alle Bausteindurchläufe hinweg erhalten bleibt (Bausteingedächtnis). Sie dienen der Speicherung von Werten eines Funktionsbausteins und werden im Instanz-Datenbaustein abgelegt.
Temporäre Variablen	Temporäre Variablen gehören lokal zu einem Codebaustein und belegen **keinen** statischen Speicherbereich, da sie im Stack der CPU abgelegt werden. Ihr Wert bleibt nur während eines Bausteinablaufs erhalten. Auf temporäre Variablen kann außerhalb des Bausteins, in dem die Variablen deklariert wurden, **nicht** zugegriffen werden.

Einteilung der Bausteinparameter

Bausteinparameter sind Platzhalter, die erst bei der konkreten Verwendung (Aufruf) des Bausteins festgelegt werden. Die im Baustein stehenden Platzhalter bezeichnet man als Formalparameter, die beim Aufruf des Bausteins zugewiesenen Werte als Aktualparameter. Die Formalparameter eines Bausteins können wie lokale Variablen betrachtet werden.

Die Bausteinparameter lassen sich nach Tabelle 10-2 in folgende Kategorien einteilen:

Tabelle 10-2 Bausteinparameter

Bausteinparameter	Bedeutung
Eingangsparameter	Eingangsparameter nehmen beim Aufruf des Bausteins die aktuellen Eingangswerte auf. Sie können nur gelesen werden.
Ausgangsparameter	Ausgangsparameter übergeben die aktuellen Ausgangswerte an den aufrufenden Baustein. Sie können beschrieben aber auch gelesen werden.
Durchgangsparameter	Durchgangsparameter übernehmen beim Aufruf den aktuellen Wert einer Variablen, verarbeiten diesen und legen anschließend die Ergebnisse wieder in der gleichen Variablen ab.

Flags (OK-Flag)

Der SCL-Compiler bietet ein Flag an, das zur Erkennung von Fehlern bei der Ausführung von Programmen in der CPU dient. Es ist eine lokale Variable vom Typ BOOL mit dem vordefinierten Namen ”OK”.

Variablen- und Parameter-vereinbarung

Tabelle 10-3 zeigt, daß jeder Kategorie der lokalen Variablen oder Parameter ein eigener Vereinbarungsblock und ein eigenes Schlüsselwortpaar zugeordnet ist.

Jeder Block enthält die Deklarationen, die für diesen Vereinbarungsblock erlaubt sind. Er darf im Vereinbarungsteil des Bausteins nur einmal vorkommen, wobei die Reihenfolge dieser Blöcke beliebig ist.

Die in einem Baustein zugelassenen Vereinbarungsblöcke sind in der Tabelle 10-3 mit einem ”x” gekennzeichnet.

Tabelle 10-3 Vereinbarungsblöcke für lokale Variablen und Parameter

Daten	Syntax	FB	FC	OB
Statische Variable	VAR : END_VAR	X	X[1)]	
Temporäre Variable	VAR_TEMP : END_VAR	X	X	X
Bausteinparameter als:				
Eingangsparameter	VAR_INPUT : END_VAR	X	X	
Ausgangsparameter	VAR_OUTPUT : END_VAR	X	X	
Durchgangsparameter	VAR_IN_OUT : END_VAR	X	X	

1) Die Vereinbarung von Variablen innerhalb des Schlüselwortpaars VAR und END_VAR ist in Funktionen zwar erlaubt, die Vereinbarungen werden aber beim Übersetzen in den temporären Bereich verschoben.

Initialisierung

Den Variablen und Parametern müssen Sie bei der Vereinbarung einen Datentyp zuordnen, der die Struktur und damit auch den Speicherbedarf bestimmt. Außerdem können Sie statischen Variablen und den Parametern eines Funktionsbausteins Anfangswerte zuweisen. Tabelle 10-4 gibt eine Übersicht, in welchen Fällen eine Initialisierung möglich ist.

Tabelle 10-4 Initialisierung von lokalen Daten

Datenkategorie	Initialisierung
Statische Variablen	möglich
Temporäre Variablen	nicht möglich
Bausteinparameter	nur möglich bei Eingangs- und Ausgangsparametern eines Funktionsbausteins

10.2 Variablen- und Parameterdeklaration

Übersicht

Eine Variablen- oder Parameterdeklaration besteht aus einem frei wählbaren Bezeichner für den Variablennamen und einer Datentypangabe. Die allgemeine Form zeigt das Syntaxdiagramm. Die Vergabe von Systemattributen für Parameter ist ausführlich beschrieben in Kapitel 8.4.

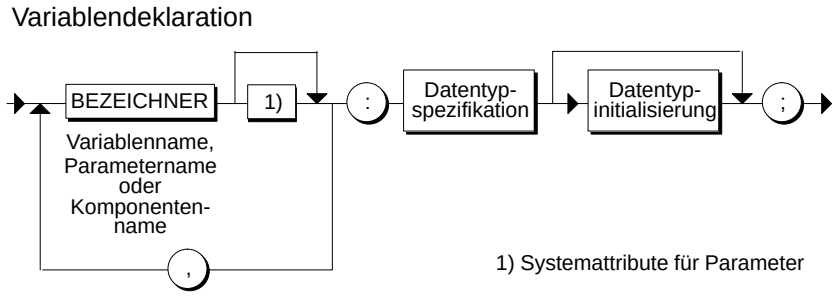


Bild 10-1 *Syntax: Variablendeklaration*

Beispiele für gültige Deklarationen sind :

```
WERT1   : REAL;
```

oder falls es mehrere Variablen desselben Typs gibt:

```
WERT2, WERT3,WERT4,....: INT;

FELD    : ARRAY[1..100, 1..10] OF REAL;

SATZ    : STRUCT
             MESSFELD:ARRAY[1..20] OF REAL;
             SCHALTER:BOOL;
          END_STRUCT
```

Datentyp-spezifikation

Es sind alle in Kapitel 9 behandelten Datentypen zugelassen.

Hinweis

Reservierte Wörter, die nur in SCL Gültigkeit haben, können Sie als Bezeichner vereinbaren, indem Sie das Zeichen ”#” voranstellen (z.B. #FOR). Das kann nützlich sein, wenn Sie Aktualparameter an Bausteine übergeben wollen, die in einer anderen Sprache (z.B. AWL) erstellt wurden.

10.3 Initialisierung

Prinzip

Statische Variablen, Eingangsparameter und Ausgangsparameter eines FBs, können bei der Vereinbarung mit einem Wert vorbesetzt werden. Diese Vorbesetzung erfolgt mit einer Wertzuweisung (:=) nach der Datentypangabe. Wie das Syntaxdiagramm 10-2 zeigt, können Sie entweder:

- einer einfachen Variablen eine Konstante oder
- einem Feld eine Initialisierungliste zuweisen.

Initialisierung

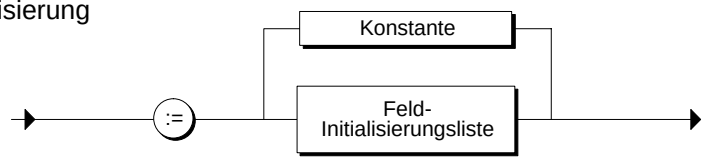


Bild 10-2 *Syntax: Initialisierung*

Beispiel:

```
WERT    :REAL   := 20.25;
```

Beachten Sie, daß eine Initialisierung einer Variablenliste (A1, A2, A3,...: INT:=...) nicht möglich ist. Sie müssen die Variablen in diesem Fall einzeln initialisieren. Felder werden nach Bild 10-3 mit Anfangswerten besetzt.

Feld-Initialisierungsliste

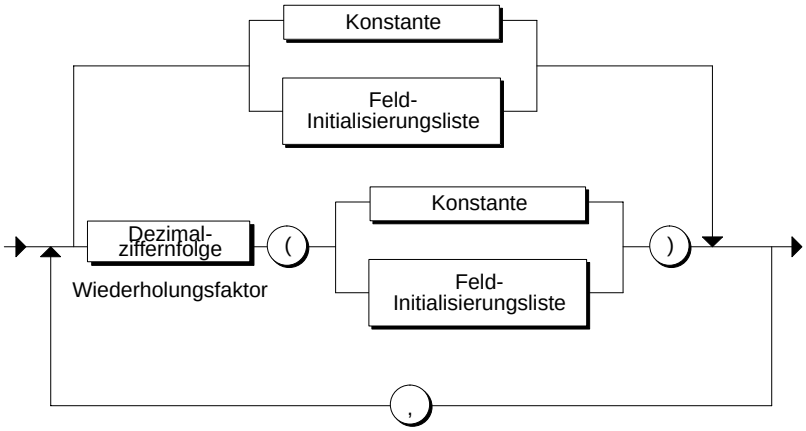


Bild 10-3 *Syntax: Feld-Initialisierungsliste*

```
FELD  :     ARRAY[1..10, 1..100] OF INT:=10(100(0));
```

Wiederholungsfaktor (Anzahl Spalten)

Wert

Wiederholungsfaktor (Anzahl Zeilen)

Beispiele

Beispiel 10-1 zeigt die Initialisierung einer statischen Variablen:

```
VAR
      INDEX1: INT:= 3;
END_VAR
```

Beispiel 10-1 Initialisierung statischer Variablen

Die Initialisierung eines zweidimensionalen Feldes zeigt Ihnen das Beispiel 10-2 . Wenn Sie die folgende Datenstruktur in SCL unter dem Namen `REGLER` vereinbaren wollen, dann schreiben Sie:

-54	736	-83	77
-1289	10362	385	2
60	-37	-7	103
60	60	60	60

```
VAR
REGLER:
ARRAY [1..4, 1..4] OF INT:=  -54,  736, -83, 77,
                           -1289, 10362, 385,  2,
                              60,  -37,  -7, 103,
                           4(60);

END_VAR
```

Beispiel 10-2 Feldinitialisierung

Eine Strukturinitialisierung zeigt Ihnen das Beispiel 10-3:

```
VAR
GENERATOR:STRUCT
  DATEN: REAL     := 100.5;
  A1:    INT      := 10;
  A2:    STRING[6]:= 'FAKTOR';
  A3:    ARRAY[1..12] OF REAL:= 12(100.0);
END_STRUCT;
END_VAR
```

Beispiel 10-3 Strukturinitialisierung

10.4 Instanzdeklaration

Übersicht

Im Vereinbarungsteil von Funktionsbausteinen können Sie außer den bereits bekannten Variablen mit elementarem, zusammengesetztem oder anwenderdefiniertem Datentyp auch Variablen vom Typ FB oder SFB deklarieren. Diese Variablen werden **lokale Instanzen** des FB oder SFB genannt.

Die lokalen Instanzdaten werden im Instanz-Datenbaustein des aufrufenden Funktionsbausteins gespeichert.

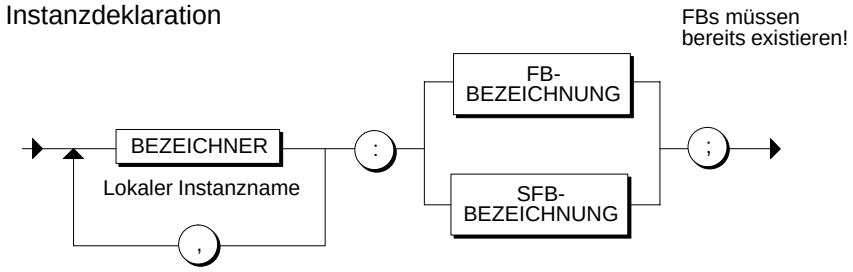


Bild 10-4 *Syntax: Instanzdeklaration*

Beispiele: Gültige Beispiele nach der Syntax in Bild 10-4 sind:

```
Versorgung1 : FB10;
Versorgung2,Versorgung3,Versorgung4 : FB100;
Motor1      : Motor ;
```

Dabei ist // Motor ist ein in der Symboltabelle eingetragenes Symbol.

Symbol	Adresse	Datentyp
MOTOR	FB20	FB20

Bild 10-5 Zugehörige Symboltabelle in STEP 7

Initialisierung

Eine lokale instanzspezifische Initialisierung ist nicht möglich.

10.5 Statische Variablen

Übersicht

Eine statische Variable ist eine lokale Variable, deren Wert über alle Bausteindurchläufe hinweg erhalten bleibt (Bausteingedächtnis). Sie dient der Speicherung von Werten eines Funktionsbausteins. Die Variablen werden im Instanz-Datenbaustein Ihres Funktionsbausteins gespeichert.

Statischer Variablenblock

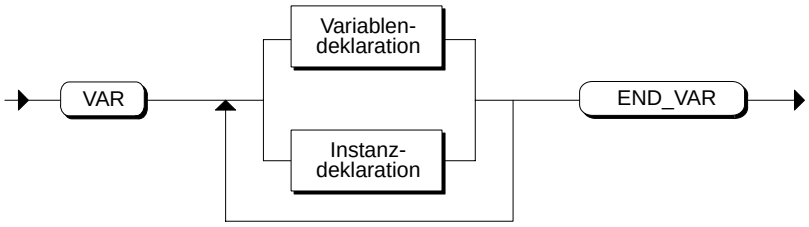


Bild 10-6 *Syntax: Statischer Variablenblock*

Vereinbarungsblock
VAR
END_VAR

Dieser Vereinbarungsblock ist Bestandteil des FB-Vereinbarungsteils. Sie können in diesem Block:

- Variablennamen und Datentypen mit der Variablendeklaration vereinbaren, optional mit Initialisierung (siehe Kapitel 10.2)
- mit der Instanzdeklaration andere bereits existierende Variablendeklarationen einfügen (siehe Kapitel 10.4).

Nach dem Übersetzen bestimmt dieser Block zusammen mit den Blöcken für die Bausteinparameter den Aufbau des zugeordneten Instanz-Datenbausteins.

Beispiel

Beispiel 10-4 zeigt die Vereinbarung statischer Variablen:

```
VAR
   DURCHLAUF       :INT;
   MESSFELD        :ARRAY[1..10] OF REAL;
   SCHALTER        :BOOL;
   MOTOR_1,Motor_2 :FB100;   // Instanzdeklaration

END_VAR
```

Beispiel 10-4 Vereinbarung statischer Variablen

Zugriff

Der Zugriff auf die Variablen erfolgt im Anweisungsteil:

- **Zugriff von innen**: d.h. im Anweisungsteil des Funktionsbausteins, in dessen Vereinbarungsteil die Variable deklariert wurde. Dies ist im Kapitel 14, Wertzuweisung, erklärt.
- **Zugriff von außen über den Instanz-DB**: Über die indizierte Variable *DBx.variable.* DBx ist die Datenbausteinbezeichnung.

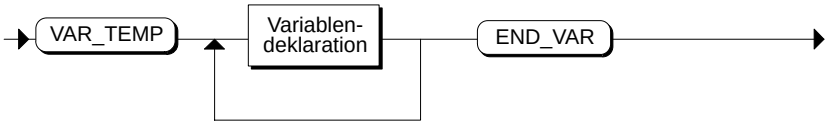
10.6 Temporäre Variablen

Übersicht

Temporäre Variablen gehören lokal zu einem Codebaustein und belegen **keinen** statischen Speicherbereich. Ihr Wert bleibt nur während eines Bausteinablaufs erhalten. Auf temporäre Variablen kann außerhalb des Bausteins, in dem die Variablen deklariert wurden, **nicht** zugegriffen werden.

Sie sollten Daten dann als temporäre Daten vereinbaren, wenn Sie diese nur zur Speicherung von Zwischenergebnissen bei der Bearbeitung Ihres OB, FB oder FC benötigen.

Temporärer Variablenblock



Keine Initialisierung möglich

Bild 10-7 *Syntax: Temporärer Variablenblock*

Vereinbarungs-block
VAR_TEMP
END_VAR

Dieser Vereinbarungsblock ist Bestandteil eines FB, FC, oder OB. Dabei werden innerhalb der Variablendeklaration Variablennamen und Datentypen angegeben (siehe Kapitel 10.2).

Bei Beginn der Ausführung eines OB, FB oder FC ist der Wert der temporären Daten nicht definiert. Eine Initialisierung ist nicht möglich.

Beispiel

Beispiel 10-5 zeigt die Vereinbarung von bausteintemporären Variablen:

```
VAR_TEMP
      PUFFER_1    :ARRAY [1..10] OF INT;
      HILF1,HILF2 :REAL;
END_VAR
```

Beispiel 10-5 Vereinbarung bausteintemporärer Variablen

Zugriff

Der Zugriff auf die Variablen erfolgt immer im Anweisungsteil des Codebausteins, in dessen Vereinbarungsteil die **Variable** deklariert wurde (**Zugriff von innen**), siehe Kapitel 14, Wertzuweisung.

10.7 Bausteinparameter

Übersicht

Bausteinparameter sind Formalparameter eines Funktionsbausteins oder einer Funktion. Wenn der Funktionsbaustein oder die Funktion aufgerufen wird, ersetzen die Aktualparameter die Formalparameter und bilden somit einen Mechanismus zum Informationsaustausch zwischen den aufgerufenen und aufrufenden Bausteinen.

- Formale Eingangsparameter nehmen die aktuellen Eingangswerte auf (Datenfluß von außen nach innen).
- Formale Ausgangsparameter dienen der Übergabe von Ausgangswerten (Datenfluß von innen nach außen).
- Formale Durchgangsparameter haben sowohl die Funktion eines Eingangs- als auch eines Ausgangsparameters.

Weitere Informationen über die Benutzung von Parametern und den damit verbundenen Informationsaustausch erhalten Sie in Kapitel 16.

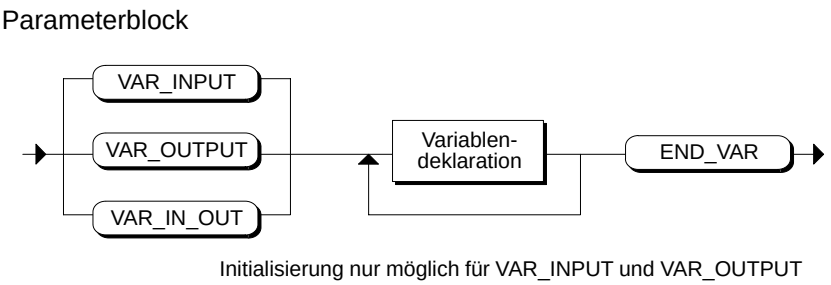


Bild 10-8 *Syntax: Parameterblock*

Vereinbarungsblock
`VAR_INPUT`
`VAR_OUTPUT`
`VAR_IN_OUT`

Dieser Vereinbarungsblock ist Bestandteil eines FB oder FC. Dabei werden innerhalb der Variablendeklaration der Variablenname und der zugeordnete Datentyp angegeben (siehe Kapitel 10.2).

Nach dem Übersetzen eines FB bestimmen diese Blöcke zusammen mit dem Block `VAR` und `END_VAR` den Aufbau des zugeordneten Instanz-Datenbausteins.

Beispiel

Beispiel 10-6 zeigt die Vereinbarung eines Parameters:

```
VAR_INPUT         //Eingangsparameter
  REGLER    :DWORD;
  UHRZEIT   :TIME_OF_DAY;
END_VAR
```

```
VAR_OUTPUT        //Ausgangsparameter
  SOLLWERTE: ARRAY [1..10] OF INT;
END_VAR
```

```
VAR_IN_OUT        //Durchgangsparameter
  EINSTELLUNG: INT;
END_VAR
```

Beispiel 10-6 Vereinbarung für Parameter

Zugriff

Der Zugriff auf die Bausteinparameter erfolgt im Anweisungsteil eines Codebausteins:

- **Zugriff von innen**: d.h. im Anweisungsteil des Bausteins, in dessen Vereinbarungsteil der **Parameter** deklariert wurde. Dies ist im Kapitel 14 (Wertzuweisung) und im Kapitel 13 (Ausdrücke, Operatoren und Operanden) erklärt.
- **Zugriff von außen über Instanz-DB**: Auf Bausteinparameter von Funktionsbausteinen können Sie über den zugeordneten Insztanz-DB zugreifen (siehe Abschnitt 14.8)

10.8 Flags (OK-Flag)

Beschreibung

Das OK-Flag dient dazu, die korrekte oder inkorrekte Ausführung eines Bausteins zu vermerken. Es ist eine globale Variable vom Typ BOOL mit dem Schlüsselwort ”OK”.

Tritt während der Ausführung einer Bausteinanweisung ein Fehler auf (z.B. ein Überlauf bei einer Multiplikation), wird das OK-Flag auf FALSE gesetzt. Beim Verlassen des Bausteins wird der Wert des OK-Flags in den implizit definierten Ausgangsparameter ENO (Kapitel 16.4) gespeichert und kann so vom aufrufenden Baustein ausgewertet werden.

Zu Beginn eines Bausteindurchlaufes hat das OK-Flag den Wert TRUE. Es kann an beliebiger Stelle im Baustein durch SCL-Anweisungen entweder abgefragt oder auf TRUE/FALSE gesetzt werden.

Vereinbarung

Das OK-Flag ist eine systemvereinbarte Variable. Es ist keine Vereinbarung erforderlich. Sie müssen aber die Compileroption ”OK-Flag” vor dem Übersetzen wählen, wenn Sie das OK-Flag in Ihrem Awenderprogramm verwenden möchten.

Beispiel

Das Beispiel 10-7 verdeutlicht die Anwendung des OK-Flags:

```
      // OK-Flag auf TRUE setzen,
      // damit überprüft werden kann,
      // ob die folgende Aktion korrekt
      // abläuft.
OK: = TRUE;
SUM: = SUM + IN;
IF OK THEN
      // Die Addition verlief korrekt.
      //:
      //:
ELSE  // Die Addition verlief fehlerhaft.
      //:
END_IF;
```

Beispiel 10-7 Verwendung der OK-Flags

Vereinbarung von Konstanten und Sprungmarken

11

Übersicht

Konstanten sind Daten, die einen festen Wert besitzen, der sich zur Programmlaufzeit nicht ändern kann. Wenn der Wert der Konstanten durch die Schreibweise ausgedrückt wird, spricht man von **Literalkonstanten.**

Eine Vereinbarung von Konstanten ist nicht notwendig. Sie haben aber die Möglichkeit, im Vereinbarungsteil symbolische Namen für die Konstanten zu vergeben.

Sprungmarken stehen für die Namen von Sprungzielen innerhalb des Anweisungsteil des Codebausteins.

Symbolische Namen von Konstanten sowie die Sprungmarken werden jeweils getrennt in eigenen Vereinbarungsblöcken vereinbart.

Kapitelübersicht

11.1 Konstanten

Anwendung von Konstanten

In Wertzuweisungen und Ausdrücken werden neben Variablen und Bausteinparametern auch Konstanten verwendet. Konstanten können dabei als Literalkonstanten oder mit einem symbolischen Namen verwendet werden.

Vereinbarung symbolischer Namen

Die Vereinbarung symbolischer Namen für Konstanten erfolgt innerhalb des CONST-Vereinbarungsblock im Vereinbarungsteil Ihres Codebausteins (siehe Kapitel 8.4).

Konstantenblock

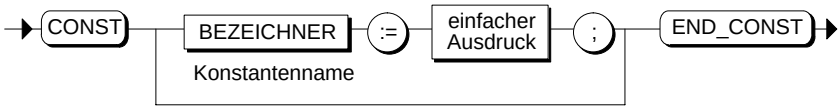


Bild 11-1 *Syntax: Konstantenblock*

Der einfache Ausdruck steht hierbei für arithmetische Ausdrücke, bei denen Sie die Grundoperationen +, –, *, /, DIV und MOD verwenden können.

Beispiel

Beispiel 11-1 veranschaulicht die Vereinbarung der symbolischen Namen:

```
CONST
 Zahl       := 10 ;
 UHRZEIT1   := TIME#1D_1H_10M_22S.2MS ;
 NAME       := 'SIEMENS' ;
 ZAHL2      := 2 * 5 + 10 * 4 ;
 ZAHL3      := 3 + ZAHL2 ;
 END_CONST
```

Beispiel 11-1 Vereinbarung von symbolischen Konstanten

Schreibweisen

SCL bietet verschiedene Schreibweisen (Formate) zum Eingeben oder Anzeigen von Konstanten. Diese Schreibweisen nennt man Literale. Die folgende Information behandelt die einzelnen Literale.

11.2 Literale

Definition

Das Literal ist eine formale Schreibweise, um den Wert und den Typ einer Konstanten zu bestimmen. Es gibt folgende Gruppen von Literalen:

- numerisches Literal
- Zeichenliteral
- Zeitangabe

Für den Wert einer Konstanten gibt es je nach Datentyp und Datenformat eine bestimmte Schreibweise:

15	WERT 15	als Integerzahl in Dezimaldarstellung
2#1111	WERT 15	als Integerzahl in Binärdarstellung
16#F	WERT 15	als Integerzahl in Hexadarstellung

Literal mit mehreren Schreibweisen für den Wert 15

Zuweisung von Datentypen zu den Konstanten

Einer Konstanten wird der Datentyp zugewiesen, dessen Wertebereich gerade noch ausreicht, um die Konstante ohne Wertverlust aufzunehmen. Bei Verwendung einer Konstanten in einem Ausdruck, z. B. einer Wertzuweisung, muß der Datentyp der Zielvariable den Wert der Konstante aufnehmen können. Wenn zum Beispiel ein Integerliteral angegeben wird, dessen Wert über den Integerbereich hinaus geht, wird angenommen, daß es sich um ein Doppelinteger handelt. Der Compiler bringt eine Fehlermeldung, wenn Sie diesen Wert einer Variablen vom Typ Integer zuweisen.

11.3 Schreibweisen für Integer- und Realzahlliterale

Übersicht

SCL bietet für die Schreibweisen numerischer Werte:

- Integerliterale für ganzahlige Werte
- Realzahlliterale für Gleitpunktzahlen.

In beiden Literalen benutzen Sie eine Ziffernfolge, deren Aufbau Bild 11-2 zeigt. Diese Ziffernfolge wird in den folgenden Syntaxdiagrammen vereinfacht mit Dezimalziffernfolge bezeichnet.

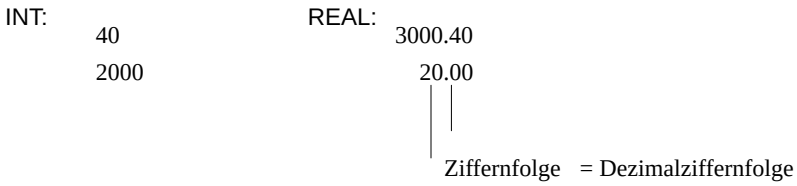


Bild 11-2 Ziffernfolge in einem Literal

Die Dezimalziffernfolge innerhalb von Literalen kann optional durch Unterstriche getrennt werden. Die Unterstriche unterstützen die bessere Lesbarkeit bei größeren Zahlen.

Dezimalziffernfolge

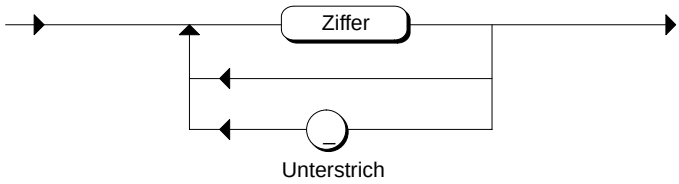


Bild 11-3 *Syntax: Dezimalziffernfolge*

Die folgenden Beispiele sind zulässige Schreibweisen für eine Dezimalziffernfolge innerhalb von Literalen:

```
1000
1_120_200
666_999_400_311
```

Integerliterale

Integerliterale sind ganzzahlige Werte. Diese können im SCL-Programm, je nach Länge, Variablen mit den folgenden Datentypen zugewiesen werden:

```
BOOL, BYTE, INT, DINT, WORD, DWORD
```

Bild 11-4 zeigt die Syntax eines Integerliterals:

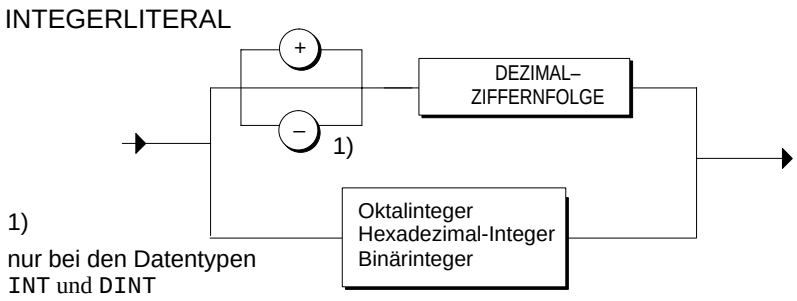


Bild 11-4 *Syntax: Integerliteral*

Die folgenden Beispiele sind zulässige Schreibweisen für Dezimalziffernfolge innerhalb von Integerliteralen:

```
1000
+1_120_200
–666_999_400_311
```

Binäre-/Oktale-/ Hexadezimale Werte

Die Angabe eines Integerliterals in einem anderen Zahlensystem als dem dezimalen, erfolgt durch das Voranstellen der Präfixe **2#, 8# oder 16#** gefolgt von der Zahl in der Darstellung des jeweiligen Systems. Der Unterstrich kann optional zwischen den Ziffern eingesetzt werden, um eine bessere Lesbarkeit großer Zahlen zu unterstützen.

Die allgemeine Schreibweise eines Integerliterals ist im Bild 11-5 am Beispiel einer Ziffernfolge für eine Oktalzahl erklärt:

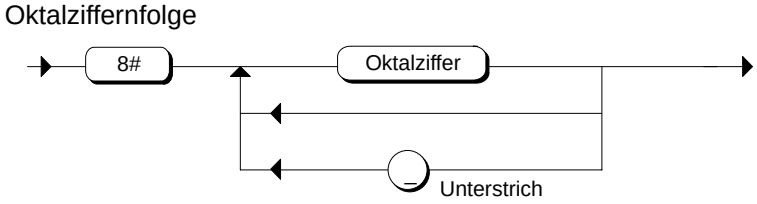


Bild 11-5 *Syntax: Oktalziffernfolge*

Die folgenden Beispiele sind mögliche Schreibweisen für Integerliterale:

```
Wert_2:=2#0101;  // Binärzahl, Dezimalwert 5
Wert_3:=8#17;    // Oktalzahl, Dezimalwert 14
Wert_4:=16#F;    // Hexadezimalzahl, Dezimalwert 15
```

Realzahlliterale

Realzahlliterale sind Werte mit Nachkommastellen. Sie können den Variablen mit dem Datentyp REAL zugewiesen werden. Die Angabe des Vorzeichens ist optional. Wird kein Vorzeichen angegeben wird die Zahl als positiv interpretiert. Bild 11-6 zeigt die Syntax für eine Realzahl:

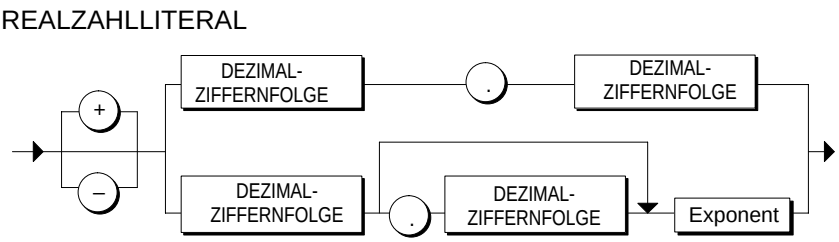


Bild 11-6 *Syntax: Realzahlliteral*

Bei der exponentiellen Schreibweise können Sie zur Angabe von Gleitpunktzahlen einen Exponenten verwenden. Die Angabe des Exponenten erfolgt durch das Voranstellen des Buchstabens ”E” oder ”e” gefolgt von einer Dezimalziffernfolge. Bild 11-7 zeigt die Syntax für die Angabe eines Exponenten.

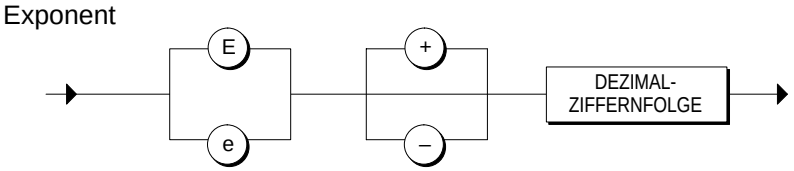


Bild 11-7 *Syntax: Exponent*

Beispiel:

Die Größe 3 x 10 ^{10} kann in SCL durch folgende Realzahlen dargestellt werden:

```
3.0E+10   3.0E10        3e+10       3E10
0.3E+11   0.3e11        30.0E+9     30e9
```

Beispiele

Das abschließende Beispiel 11-2 faßt die Möglichkeiten nochmal zusammen:

```
// Integerliterale
ZAHL1:= 10 ;
ZAHL2:= 2#1010 ;
ZAHL3:= 16#1A2B ;
// Realzahlliterale
ZAHL4:= -3.4 ;
ZAHL5:= 4e2 ;
ZAHL6:= 40_123E10;
```

Beispiel 11-2 Numerische Literale

11.4 Schreibweisen für Character- und Stringliterale

Übersicht

SCL bietet auch die Möglichkeit, Textinformationen einzugeben und zu verarbeiten, beispielsweise eine Zeichenkette, die als Meldung angezeigt werden soll.

Berechnungen sind mit den Zeichenliteralen nicht möglich, das bedeutet, daß Zeichenliterale **nicht** in Ausdrücken verwendet werden können. Es wird unterschieden zwischen:

- Characterliteral, das ist ein Einzelzeichen und
- Stringliteral, das ist eine Zeichenfolge aus max. 254 Einzelzeichen.

Characterliteral (Einzelzeichen)

Das Characterliteral enthält genau ein Zeichen (siehe Bild 11-8). Das Zeichen wird durch einfache Hochkommata (') eingeschlossen.

CHARACTERLITERAL

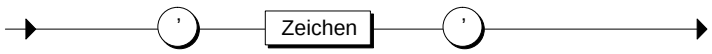


Bild 11-8 *Syntax: Characterliteral*

Beispiel:

```
Zeichen_1 := 'B';        // Zeichen B
```

Stringliteral

Ein Stringliteral ist eine Zeichenkette von max. 254 Zeichen (Buchstaben, Ziffern und Sonderzeichen), die in Anführungszeichen (') stehen. Sowohl Klein- wie Großbuchstaben können benutzt werden.

STRINGLITERAL

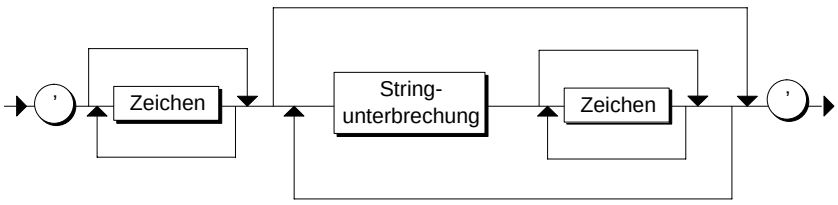


Bild 11-9 *Syntax: Stringliteral*

Einige gültige Stringliterale :

```
'ROT'     '76181 Karlsruhe'          '270–32–3456'
'DM19.95' 'Die richtige Antwort ist:'
```

Beachten Sie bitte, daß bei einer Zuweisung eines Stringliterals an eine Stringvariable die Anzahl der Zeichen auf < 254 begrenzt sein kann.

Die Wertzuweisung :

```
TEXT:STRING[20]:='SIEMENS KARLSRUHE Rheinbrückenstr.'
```

verursacht eine Fehlermeldung und hinterlegt in der Variablen 'TEXT'

```
'SIEMENS KARLSRUHE Rh'
```

Spezielle Formatierungszeichen, das Anführungzeichen (') oder ein $-Zeichen können Sie mit Fluchtsymbol $ eingeben. Ein Stringliteral dürfen Sie mehrmals unterbrechen und fortsetzen.

String-unterbrechung

Ein String steht entweder in einer Zeile eines SCL-Bausteins oder wird durch spezielle Kennungen auf mehrere Zeilen aufgeteilt. Zur Unterbrechung eines Strings dient die Kennung '$>', zur Fortsetzung in einer Folgezeile dient die Kennung '$<'.

```
TEXT:STRING[20]:='Der FB$>//Vorabversion
$<konvertiert';
```

Der Raum zwischen der Unterbrechungs- und der Fortsetzungskennung darf sich auch auf mehrere Zeilen erstrecken und neben Leerzeichen lediglich Kommentar enthalten. Ein Stringliteral dürfen Sie auf diese Weise mehrmals unterbrechen und fortsetzen (siehe auch Bild 11-10).

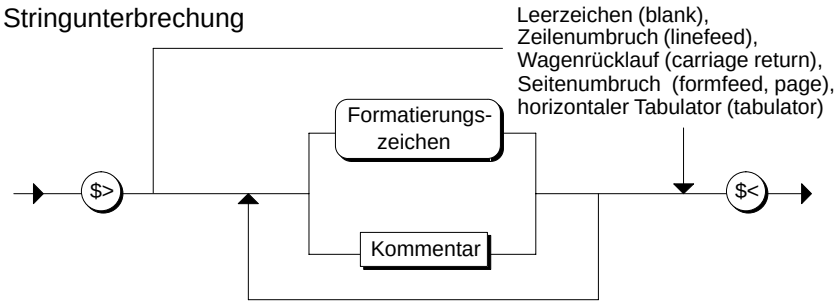


Bild 11-10 *Syntax: Stringunterbrechung*

Druckbare Zeichen

Die Zeichen eines Character- oder Stringliterals dürfen dem vollständigen erweiterten ASCII-Zeichensatz entstammen. Spezielle Formatierungszeichen und nicht direkt darstellbare Zeichen (' und $) müssen Sie mit Hilfe des Fluchtsymbols $ eingeben.

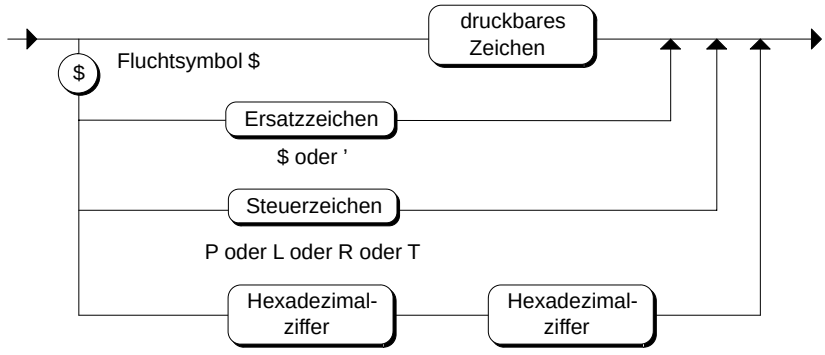


Bild 11-11 *Syntax: Zeichen* Ersatzdarstellung im Hexacode

Nichtdruckbare Zeichen

In einem Character- bzw. Stringliteral können Sie auch nichtdruckbare Zeichen aus dem vollständigen erweiterten ASCII-Zeichensatz angeben. Sie müssen dazu die Ersatzdarstellung im Hexacode angeben.

Ein ASCII-Zeichen geben Sie durch **$hh** an, wobei **hh** stellvertretend für den hexadezimal ausgedrückten Wert des ASCII-Zeichens steht.

Beispiel:

```
ZEICHEN_A   :='$41'; //entspricht dem Zeichen 'A'
Blank       :='$20'; //entspricht dem Zeichen ␣
```

Weitere Informationen über Ersatz- und Steuerzeichen finden Sie in Anhang A.

Beispiele

Die folgenden Beispiele veranschaulichen die Formulierung der Zeichenliterale:

```
// Characterliteral
Zeichen:= 'S' ;

// Stringliteral:
NAME:= 'SIEMENS' ;

// Unterbrechung eines Stringliterals:
MELDUNG1:= 'MOTOR- $>
$< Steuerung' ;

 // String in hexadezimaler Darstellung:
 MELDUNG1:= '$41$4E' (*Zeichenfolge AN*);
```

Beispiel 11-3 Zeichenliterale

11.5 Schreibweisen für Zeitangaben

Schreibweise für Zeittypen

SCL bietet verschiedene Formate zum Eingeben von Zeit- und Datumswerten. Folgende Zeitangaben sind möglich:

```
Datum
Zeitdauer
Tageszeit
Datum und Zeit
```

Datum

Ein Datum wird durch den Präfix DATE# oder D# eingeleitet (siehe Bild 11-12).

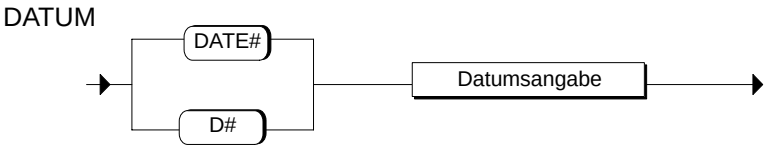


Bild 11-12 *Syntax: Datum*

Die **Datumsangabe** erfolgt durch Integerzahlen für die Jahreszahl (4-stellig), der Monatsangabe und die Tagesangabe, die durch Bindestriche zu trennen sind.

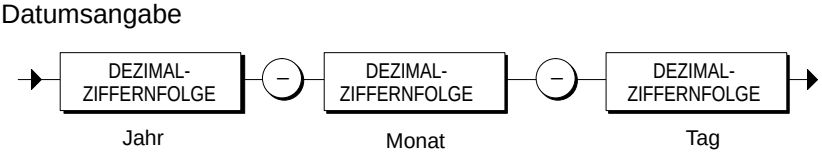


Bild 11-13 *Syntax: Datumsangabe*

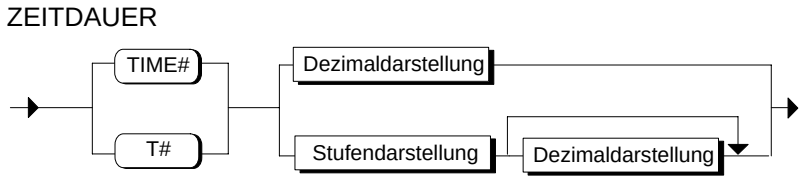
Gültige Datumsangaben sind:

```
// Datumsangabe
ZEITVARIABLE1:= DATE#1995-11-11;
ZEITVARIABLE2:= D#1995-05-05;
```

Zeitdauer

Eine Zeitdauer wird durch den Präfix TIME# oder T# eingeleitet (siehe Bild 11-14). Die Angabe der Zeitdauer kann auf zwei Arten erfolgen:

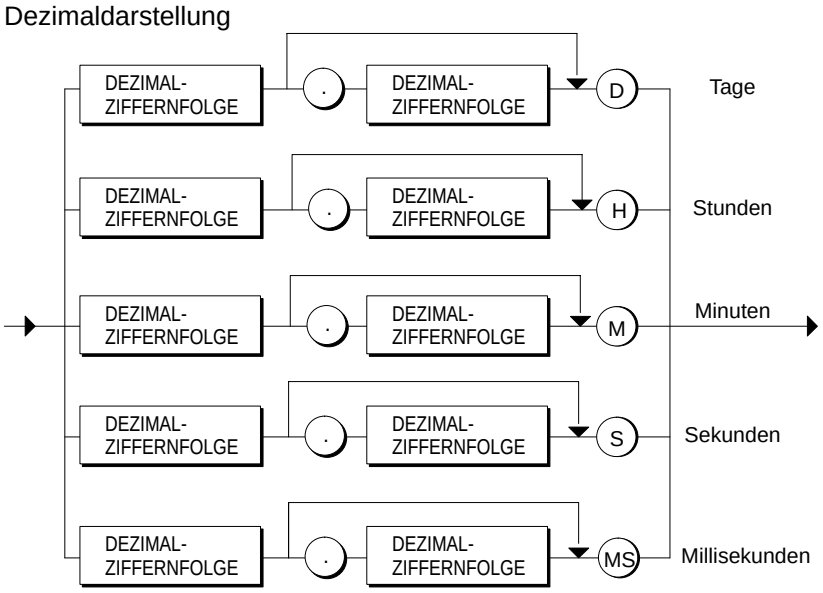
- Dezimaldarstellung
- Stufendarstellung



- Jede Zeiteinheit (z. B. Stunden, Minuten) darf nur 1 x angegeben werden.
- Die Reihenfolge - Tage, Stunden, Minuten, Sekunden, Millisekunden - ist einzuhalten.

Bild 11-14 *Syntax: Zeitdauer*

Die **Dezimaldarstellung** verwenden Sie, wenn Sie Ihre Zeitdauer alternativ in Tagen, Stunden, Minuten, Sekunden oder Millisekunden ausdrücken müssen:



Der Einstieg in die Dezimaldarstellung ist nur bei noch nicht definierten Zeiteinheiten möglich.

Bild 11-15 *Syntax: Dezimaldarstellung*

Beispiele

Gültige Angaben sind:

```
TIME#20.5D              für 20,5  Tage
TIME#45.12M             für 45,12 Minuten
T#300MS                 für 300 Millisekunden
```

Die **Stufendarstellung** verwenden Sie, wenn Sie Ihre Zeitdauer als eine Sequenz von Tagen, Stunden, Minuten, Sekunden oder Millisekunden angeben müssen (siehe Bild 11-16). Das Weglassen einzelner Komponenten ist erlaubt. Eine Komponente muß aber mindestens angegeben werden.

Stufendarstellung

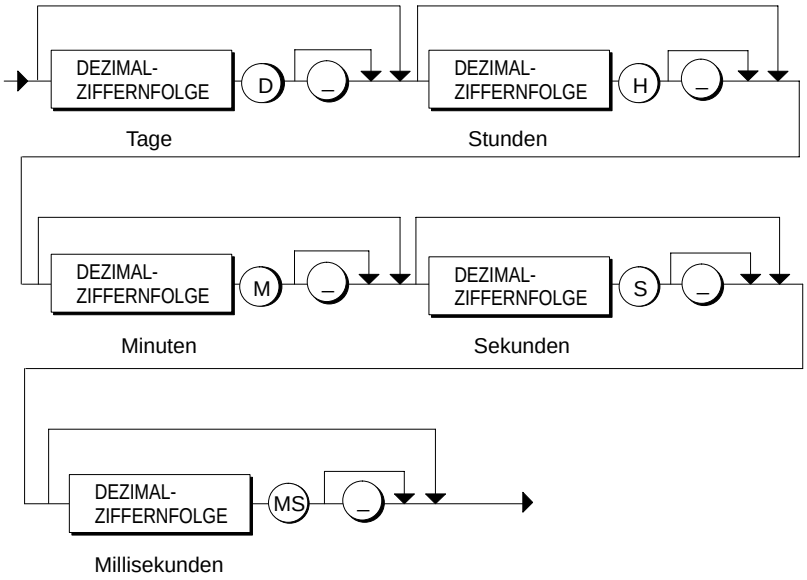


Bild 11-16 *Syntax: Stufendarstellung*

Gültige Angaben sind:

```
TIME#20D_12H
TIME#20D_10H_25M_10s
TIME#200S_20MS
```

Tageszeit

Eine Tageszeit wird durch den Präfix TIME_OF_DAY# oder TOD# eingeleitet (siehe Bild 11-17).

TAGESZEIT

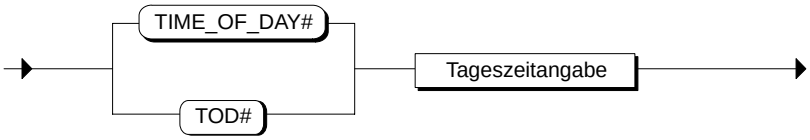


Bild 11-17 *Syntax: Tageszeit*

Die **Tageszeitangabe** erfolgt durch die Angabe der Stunden, Minuten und der Sekunden, die durch Doppelpunkt zu trennen sind. Die Angabe der Millisekunden ist optional. Sie wird durch einen Punkt von den anderen Angaben getrennt. Bild 11-18 zeigt die Syntax der Tageszeitangabe:

Tageszeitangabe

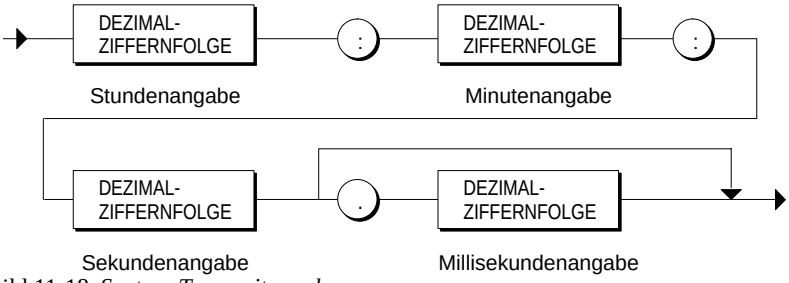


Bild 11-18 *Syntax: Tageszeitangabe*

Gültige Angaben sind:

```
//Tageszeitangabe
UHRZEIT1:= TIME_OF_DAY#12:12:12.2;
UHRZEIT2:= TOD#11:11:11.200;
```

Datum und Zeit

Eine Datums- und Zeitangabe wird durch den Präfix `DATE_AND_TIME#` oder `DT#` eingeleitet (siehe Bild 11-19). Sie ist ein aus der Datumsangabe und der Tageszeitangabe zusammengesetztes Literal.

DATUM UND ZEIT

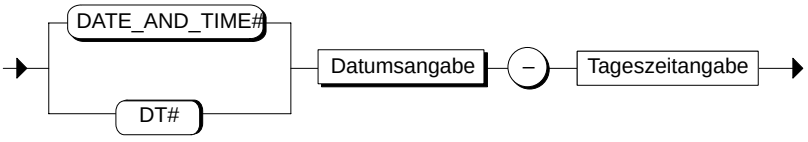


Bild 11-19 *Syntax: Datum und Zeit*

Das Beispiel verdeutlicht die Angabe von Datum und Zeit:

```
// Tageszeitangabe
UHRZEIT1:= DATE_AND_TIME#1995-01-01–12:12:12.2;
UHRZEIT2:= DT#1995-02-02–11:11:11;
```

11.6 Sprungmarken

Beschreibung

Sprungmarken (Labels) dienen dazu, das Ziel einer GOTO-Anweisung (siehe Kapitel 11-4) zu definieren.

Vereinbarung von Sprungmarken

Sprungmarken werden im Vereinbarungsteil eines Codebausteins mit ihrem symbolischen Namen (siehe Kapitel 8.4) vereinbart:

Sprungmarkenblock

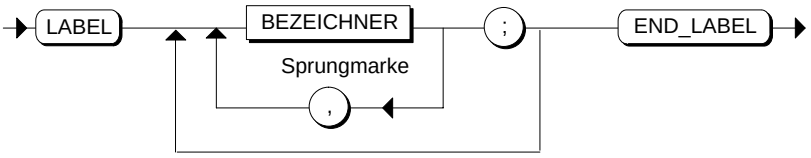


Bild 11-20 *Syntax: Sprungmarkenblock*

Beispiel

Beispiel 11-4 verdeutlicht die Vereinbarung der Sprungmarken:

```
LABEL
      MARKE1, MARKE2, MARKE3;
END_LABEL;
```

Beispiel 11-4 Sprungmarken

Vereinbarung globaler Daten

12

Zusammenfassung

Globale Daten sind Daten, die von jedem Codebaustein (FC, FB, OB) aus benutzbar sind. Auf diese Daten kann entweder absolut oder symbolisch zugegriffen werden. In diesem Kapitel werden die einzelnen Datenbereiche vorgestellt. Es wird beschrieben, wie Sie diese Daten ansprechen können.

Kapitelübersicht

12.1 Übersicht

Globale Daten

Sie haben in SCL die Möglichkeit, globale Daten anzusprechen. Es gibt zwei Arten von globalen Daten:

- **CPU-Speicherbereiche**

  Diese Speicherbereiche sind systemvereinbarte Daten: z. B. Eingänge, Ausgänge und Merker (siehe Kapitel 7.5). Die Anzahl der zur Verfügung stehenden CPU-Speicherbereiche ist durch die jeweilige CPU festgelegt.

- **Globale Anwenderdaten als ladbare Datenbausteine**

  Diese Datenbereiche liegen innerhalb von Datenbausteinen. Um sie benutzen zu können, müssen Sie vorher die Datenbausteine erstellen und darin die Daten vereinbaren. Im Fall von Instanz-Datenbausteinen werden sie von Funktionsbausteinen abgeleitet und automatisch erzeugt.

Zugriffsarten

Der Zugriff auf die globalen Daten kann auf folgende Arten erfolgen:

- **absolut:** Über Operandenkennzeichen und absolute Adresse.
- **symbolisch:** Über ein Symbol, das Sie vorher in der Symboltabelle definiert haben (siehe **/231/**).
- **indiziert:** Über Operandenkennzeichen und Feldindex.
- **strukturiert:** Über eine Variable.

Tabelle 12-1 Verwendung der Zugriffsarten auf globale Daten

Zugriffsart	CPU-Speicherbereiche	Globale Anwenderdaten
absolut	ja	ja
symbolisch	ja	ja
indiziert	ja	ja
strukturiert	nein	ja

12.2 CPU-Speicherbereiche

Definition

Die CPU-Speicherbereiche sind systemvereinbarte Bereiche. Deshalb brauchen Sie diese Bereiche in Ihrem Codebaustein nicht zu vereinbaren.

Verschiedene Speicherbereiche

Jede CPU stellt folgende Speicherbereiche mit einem eigenen Adreßraum zur Verfügung:

- Ein- / Ausgänge im Prozeßabbild
- Peripherieein- / -ausgänge
- Merker
- Zeitglieder, Zähler (siehe Kapitel 17)

Syntax für den Zugriff

Der Zugriff auf einen CPU-Speicherbereich erfolgt in einer Wertzuweisung im Anweisungsteil eines Codebausteins (siehe Kapitel 14.3):

- mit einem einfachen Zugriff, den Sie absolut oder symbolisch angeben können oder
- mit einem indizierten Zugriff.

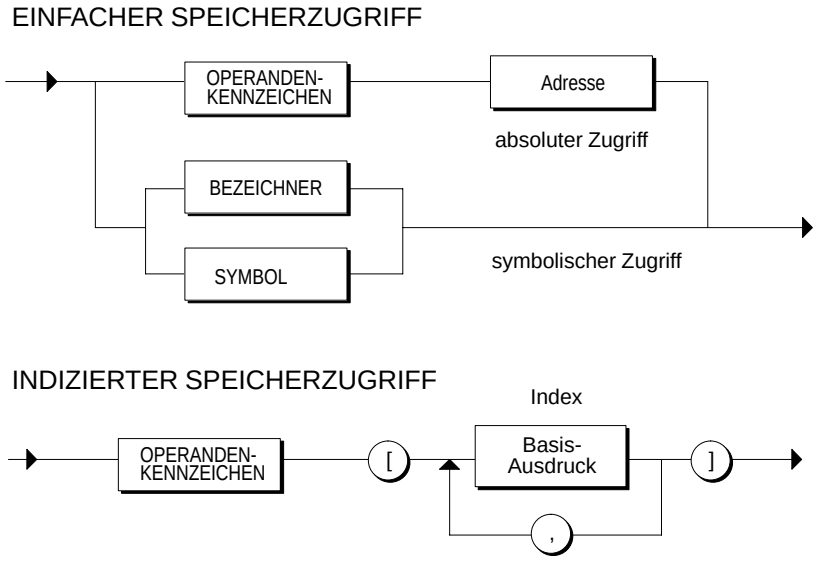


Bild 12-1 *Syntax:Einfacher und Indizierter Speicherzugriff*

12.3 Absoluter Zugriff auf CPU-Speicherbereiche

Prinzip

Der absolute Zugriff auf einen Speicherbereich erfolgt z. B. über eine Wertzuweisung an eine typgleiche Variable:

```
STATUS_2:= EB10;
```

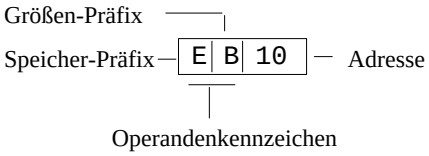
Absoluter Zugriff

Typgleiche Variable

Der Absolutbezeichner verweist auf einen Speicherbereich innerhalb der CPU. Sie spezifizieren diesen Bereich, indem Sie das Operandenkennzeichen (hier `EB`) gefolgt von der Adresse (hier `10`) angeben.

Absolutbezeichner

Der Absolutbezeichner setzt sich aus dem Operandenkennzeichen mit Speicher- und Größen-Präfix sowie einer Adresse zusammen.



Operanden-kennzeichen

Die Kombination aus Speicher- und Größen-Präfix bildet das Operandenkennzeichen.

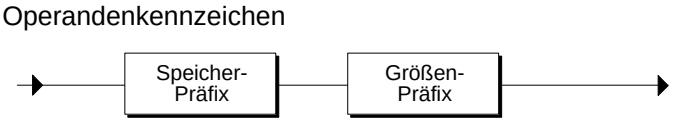


Bild 12-2 *Syntax: Operandenkennzeichen*

Speicher-Präfix

Mit dem Speicher-Präfix legen Sie die Art des Speicherbereichs fest. Nach Bild 12-3 stehen Ihnen folgende Arten zur Verfügung[1)]:

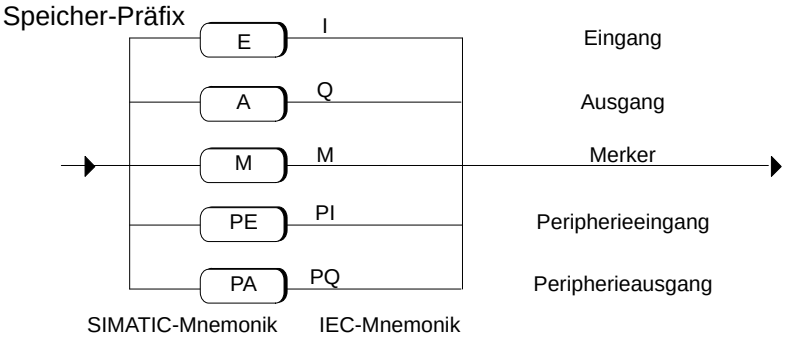


Bild 12-3 *Syntax: Speicher-Präfix*

1) *Abhängig von der Einstellung im SIMATIC Manager haben entweder die SIMATIC oder die IEC Operandenkennzeichen eine reservierte Bedeutung. Man kann die Sprache und unabhängig davon die Mnemonic im SIMATIC Manager einstellen.*

Größen-Präfix

Durch Angabe des Größen-Präfix spezifizieren Sie die Größe bzw. den Typ des Speicherbereichs, der aus der Peripherie gelesen werden soll, z.B. ein Byte oder ein Wort. Die Angabe des Größen-Präfix ist optional, wenn Sie ein Bit spezifizieren. Bild 12-4 zeigt die Syntax:

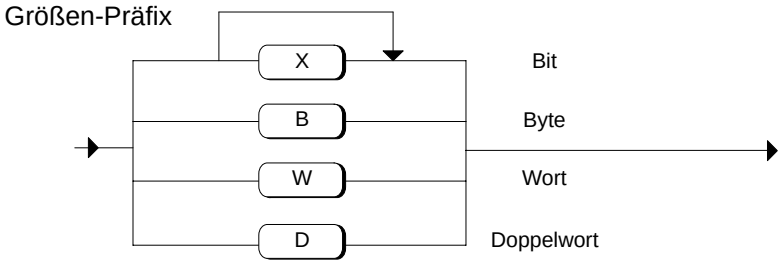


Bild 12-4 *Syntax: Größen-Präfix*

Adresse

Bei der Angabe einer Adresse geben Sie, je nachdem welchen Größen-Präfix Sie benutzt haben, eine absolute Adresse an, die auf ein Bit, Byte, Wort oder Doppelwort zeigt. Nur wenn Sie ”Bit” spezifiziert haben, können Sie eine zusätzliche Bitadresse angeben (siehe Bild 12-5). Die erste Nummer entspricht der Byteadresse und die zweite der Bitadresse.

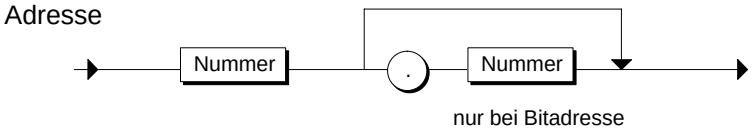


Bild 12-5 *Syntax: Adresse*

Beispiele

Einige Beispiele für den absoluten Zugriff:

```
STATUSBYTE  := EB10;
STATUS_3    := E1.1;
Messwert    := EW20;
```

Beispiel 12-1 Absoluter Zugriff

12.4 Symbolischer Zugriff auf CPU-Speicherbereiche

Prinzip

Bei der symbolischen Programmierung verwenden Sie anstelle des absoluten Zugriffs mit Operandenkennzeichen und Adresse einen symbolischen Namen für die Adressierung eines CPU-Speicherbereiches:

Symbol	Operand	Datentyp	Kommentar
Motorkontakt	E 1.7	BOOL	Kontaktschalter 1 für Motor A 1
Eingang	EW 10	INT	Eingangsstatuswort
Eingangsbyte1	EB 1	BYTE	Eingangsbyte
”Eingang 1.1”	E 1.1	BOOL	Lichtschranke
Messkanäle	MW 2	WORD	Messwertpuffer

Die Zuordnung der symbolischen Namen zu den jeweiligen Operanden Ihres Anwenderprogrammes führen Sie durch, indem Sie eine Symboltabelle erstellen.

Als Datentypangabe sind alle elementaren Datentypen zugelassen, sofern sie die Zugriffsbreite aufnehmen können.

Zugriff

Der Zugriff erfolgt z. B. über eine Wertzuweisung an eine typgleiche Variable mit der vereinbarten Symbolik.

```
MESSWERT_1  := Motorkontakt;
```

Erstellen der Symboltabelle

Das Erstellen der Symboltabelle und die Werteingabe in die Symboltabelle erfolgt mit STEP 7.

Sie können die Symboltabelle mit dem SIMATIC Manager oder direkt mit SCL über den Menübefehl **Extras ▸ Symboltabelle** öffnen.

Darüberhinaus ist es möglich, als Textdatei vorliegende Symboltabellen, die mit beliebigen Texteditoren erstellt sein können, zu importieren und weiterzubearbeiten (Informationen hierzu finden Sie in /**231**/).

Beispiele

Einige Beispiele für den symbolischen Zugriff:

```
STATUSBYTE  := Eingangsbyte;
STATUS_3    := ”Eingang 1.1”;
Messwert    := Messkanäle;
```

Beispiel 12-2 Symbolischer Zugriff

12.5 Indizierter Zugriff auf CPU-Speicherbereiche

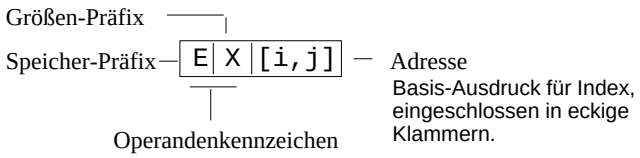
Prinzip

Sie haben auch die Möglichkeit, auf die Speicherbereiche einer CPU indiziert zuzugreifen. Dies hat gegenüber dem absoluten Zugriff den Vorteil, daß Sie die Zugriffe durch die Verwendung von variablen Indizes dynamisch adressieren können. So können Sie z.B. die Laufvariable einer FOR-Schleife zur Indizierung verwenden.

Der indizierte Zugriff auf einen Speicherbereich geschieht ähnlich wie der absolute Zugriff. Er unterscheidet sich nur in der Angabe der Adresse. Anstelle der Adresse wird ein Index spezifiziert, der eine Konstante, eine Variable oder ein arithmetischer Ausdruck sein kann.

Absolutbezeichner

Der Absolutbezeichner setzt sich beim indizierten Zugriff aus dem Operandenkennzeichen sowie einem Basisausdruck für das Indizieren zusammen (nach Kapitel 12.3).



Regeln für den indizierten Zugriff

Die Indizierung muß den folgenden Regeln entsprechen:

- Bei einem Zugriff, der vom Datentyp BYTE, WORD oder DWORD ist, müssen Sie genau einen Index verwenden. Der Index wird als Byteadresse interpretiert. Die Zugriffsbreite wird durch das Größen-Präfix festgelegt.
- Bei einem Zugriff, der vom Datentyp BOOL ist, müssen Sie zwei Indizes benutzen. Der erste Index spezifiziert die Byteadresse, der zweite Index die Bitposition innerhalb des Bytes.
- Jeder Index muß ein arithmetischer Ausdruck vom Datentyp `INT` sein.

```
MESSWORT_1  := EW[ZAEHLER];

AUSMARKE    := E[BYTENR, BITNR];
```

Beispiel 12-3 Indizierter Zugriff

12.6 Datenbausteine

Übersicht

In Datenbausteinen können Sie für Ihren Anwendungsfall alle die Daten speichern und verarbeiten, deren Gültigkeitsbereich sich auf das ganze Programm bzw. auf das ganze Projekt bezieht. Jeder Codebaustein kann darauf lesend oder schreibend zugreifen.

Vereinbarung

Die Syntax für den Aufbau von Datenbausteinen können Sie in Kapitel 8 nachschlagen. Sie müssen zwischen zwei Arten von Datenbausteinen unterscheiden:

- Datenbausteine
- Instanz-Datenbausteine

Zugriff auf Datenbausteine

Der Zugriff auf die Daten eines beliebigen Datenbausteins kann immer auf folgende Arten erfolgen:

- einfach bzw. absolut
- indiziert
- strukturiert

Bild 12-6 gibt einen Überblick über die Zugriffsarten:

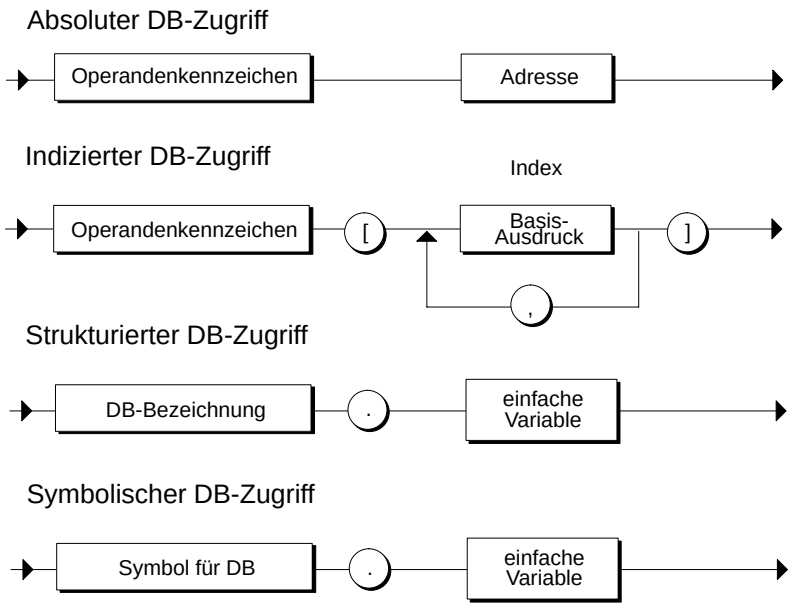


Bild 12-6 *Syntax: Absoluter, indizierter und strukturierter DB-Zugriff*

12.7 Absoluter Zugriff auf Datenbausteine

Prinzip

Der absolute Zugriff erfolgt wie bei den CPU-Speicherbereichen über eine Wertzuweisung an eine typgleiche Variable. Nach der Angabe der DB-Bezeichnung folgt das Schlüsselwort `'D'` mit Angabe des Größen-Präfix (z.B. `X` für `BIT`) und der Byteadresse (z.B. `13.1`).

```
STATUS_5:= DB11.DX13.1;
```

Adresse
Größen-Präfix
DB-Bezeichnung
Typgleiche Variable

Zugriff

Sie spezifizieren den Zugriff, indem Sie die DB-Bezeichnung zusammen mit dem Größen-Präfix und der Adresse angeben (siehe Bild 12-7).

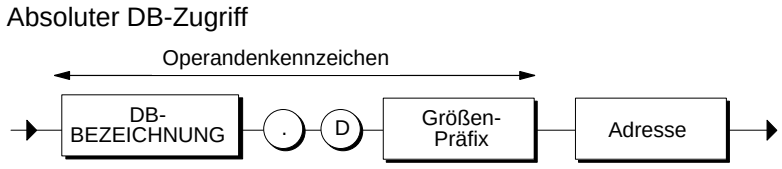


Bild 12-7 *Syntax: Absoluter DB-Zugriff*

Größen-Präfix

Gibt die Länge des Speicherbereiches im Datenbaustein an, der angesprochen werden soll, z.B. ein Byte oder ein Wort. Die Angabe des Größen-Präfix ist optional, wenn Sie ein Bit spezifizieren möchten. Bild 12-8 zeigt die Syntax für den Größen-Präfix.

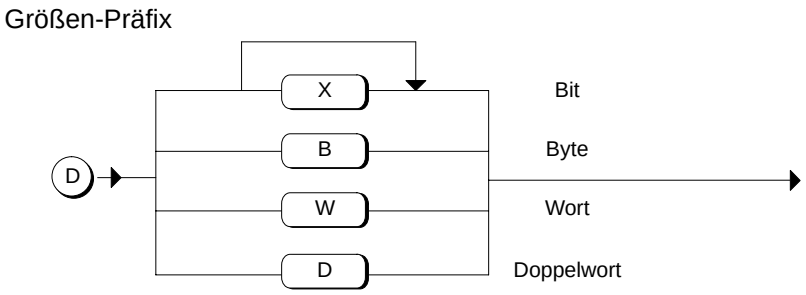


Bild 12-8 *Syntax: Größen-Präfix*

Adresse

Bei der Angabe der Adresse nach Bild 12-9 geben Sie, je nachdem welchen Größen-Präfix Sie benutzt haben, eine absolute Adresse an, die auf ein Bit, Byte, Wort oder Doppelwort zeigt. Nur wenn Sie ”Bit” spezifiziert haben, können Sie eine zusätzliche Bitadresse angeben. Die erste Nummer entspricht der Byteadresse und die zweite der Bitadresse.

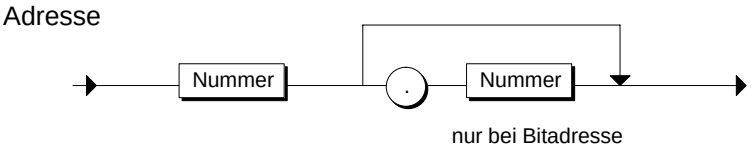


Bild 12-9 *Syntax: Adresse*

Beispiele

Einige Beispiele für den absoluten Zugriff auf einen Datenbaustein. Der Datenbaustein selbst wird im ersten Teil absolut und im zweiten symbolisch angegeben:

```
STATUSBYTE  := DB101.DB10;
STATUS_3    := DB30.D1.1;
Messwert    := DB25.DW20;

STATUSBYTE  := Statusdaten.DB10;
STATUS_3    := ”Neue Daten”.D1.1;
Messwert    := Messdaten.DW20;

STATUS_1    := WORD_TO_BLOCK_DB(INDEX).DW10;
```

Beispiel 12-4 Absoluter Zugriff

12.8 Indizierter Zugriff auf Datenbausteine

Indizierter Zugriff

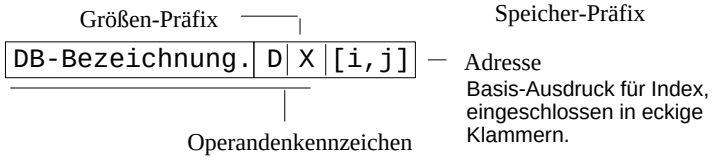
Sie haben auch die Möglichkeit, auf globale Datenbausteine indiziert zuzugreifen. Dies hat gegenüber der absoluten Adressierung den Vorteil, daß Sie durch die Verwendung von variablen Indizes dynamisch adressieren. Sie können z.B. die Laufvariable einer FOR-Schleife zur Indizierung verwenden.

Der indizierte Zugriff auf einen Datenbaustein geschieht ähnlich wie der absolute Zugriff. Er unterscheidet sich nur in der Angabe der Adresse.

Anstelle der Adresse wird ein Index spezifiziert, der eine Konstante, eine Variable oder ein arithmetischer Ausdruck sein kann.

Absolutbezeichner

Der Absolutbezeichner setzt sich beim indizierten Zugriff aus dem Operandenkennzeichen (nach Kapitel 12.7) sowie einem Basisausdruck für das Indizieren zusammen.



Regeln beim indizierten Zugriff

Die Indizierung muß den folgenden Regeln entsprechen:

- Jeder Index muß ein arithmetischer Ausdruck vom Datentyp `INT` sein.
- Bei einem Zugriff, der vom Datentyp `BYTE`, `WORD` oder `DWORD` ist, müssen Sie genau einen Index verwenden. Der Index wird als Byteadresse interpretiert. Die Zugriffsbreite wird durch das Größen-Präfix festgelegt.
- Bei einem Zugriff, der vom Datentyp `BOOL` ist, müssen Sie zwei Indizes benutzen. Der erste Index spezifiziert die Byteadresse, der zweite Index die Bitposition innerhalb des Bytes.

```
STATUS_1:= DB11.DW[ZAEHLER];
STATUS_2:= DB12.DX[WNR, BITNR];

STATUS_1:= Datenbasis1.DW[ZAEHLER];
STATUS_2:= Datenbasis2.DX[WNR, BITNR];

STATUS_1:= WORD_TO_BLOCK_DB(INDEX).DW[ZAEHLER];
```

Beispiel 12-5 Indizierter Zugriff

12.9 Strukturierter Zugriff auf Datenbausteine

Prinzip

Der strukturierte Zugriff erfolgt über eine Wertzuweisung an eine typgleiche Variable.

```
ZEIT_1:= DB11.TAGESZEIT ;
    |        |        |
    |        |        Einfache Variable
    |        DB-Bezeichnung
    Typgleiche Variable
```

Die Variable in dem Datenbaustein referenzieren Sie, indem Sie den DB-Namen und, getrennt durch einen Punkt, den Namen der einfachen Variablen angeben. Die Syntax dazu ist im Bild 12-6 beschrieben.

Die Einfache Variable steht für eine Variable, der Sie bei der Vereinbarung einen elementaren oder einen zusammengesetzten Datentyp zugeordnet haben.

Beispiele

```
Im Vereinbarungsteil des FB10:
VAR
Ergebnis:   STRUCT ERG1 : INT;
                   ERG2 : WORD;
            END_STRUCT
END_VAR
```

```
Anwenderdefinierter Datentyp UDT1:
TYPE UDT1   STRUCT ERG1 : INT;
                   ERG2 : WORD;
            END_STRUCT
```

```
DB20 mit anwenderdefiniertem Datentyp:
DB20
UDT1
BEGIN ...
```

```
DB30 ohne anwenderdefinierten Datentyp:
DB30        STRUCT ERG1 : INT;
                   ERG2 : WORD;
            END_STRUCT
BEGIN ...
```

Beispiel 12-6 Vereinbarung der Daten für die Datenbausteine

```
Funktionsbaustein mit den Zugriffen:
..
FB10.DB10();
ERGWORT_A   :=    DB10.Ergebnis.ERG2;
ERGWORT_B   :=    DB20.ERG2;
ERGWORT_C   :=    DB30.ERG2;
```

Beispiel 12-7 Zugriff auf die Daten der Datenbausteine

Ausdrücke, Operatoren und Operanden 13

Übersicht

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. SCL unterscheidet:

- arithmetische Ausdrücke
- Potenzausdrücke
- Vergleichsausdrücke
- logische Ausdrücke

Kapitelübersicht

13.1 Operatoren

Übersicht

Ausdrücke bestehen aus Operatoren und Operanden. Die meisten Operatoren von SCL verknüpfen zwei Operanden und werden deshalb als *binär* bezeichnet. Andere Operatoren arbeiten mit nur einem Operanden, daher bezeichnet man sie als *unär.*

Binäre Operatoren werden zwischen die Operanden geschrieben (z. B. A + B). Ein unärer Operator steht immer unmittelbar vor seinem Operand (z. B. –B).

Die in Tabelle 13-1 gezeigte Priorität der Operatoren regelt die Reihenfolge der Berechnung. Die Ziffer ”1” entspricht dabei der höchsten Priorität.

Operatoren-klassen

Tabelle 13-1 Übersicht der Operatoren

Klasse	Operator	Darstellung	Priorität
Zuweisungsoperator *Dieser Operator hinterlegt einen Wert in einer Variablen*	Zuweisung	: =	11
Arithmetische Operatoren *Diese Operatoren benötigt man für mathematische Rechnungen*	Potenz	**	2
	Unäre Operatoren		
	unäres Plus	+	3
	unäres Minus	-	3
	Arithmetische Basisoperatoren		
	Multiplikation	*	4
	Division	/	4
	Modulo-Funktion	MOD	4
	Ganzzahlige Division	DIV	4
	Addition	+	5
	Subtraktion	-	5
Vergleichsoperatoren *Diese Operatoren benötigt man, um Bedingungen formulieren zu können*	Kleiner	<	6
	Größer	>	6
	Kleiner gleich	<=	6
	Größer gleich	>=	6
	Gleichheit	=	7
	Ungleichheit	<>	7
Logische Operatoren *Diese Operatoren benötigt man für logische Ausdrücke*	Negation (unär)	NOT	3
	Logische Basisoperatoren		
	Und	AND oder &	8
	Exklusiv-Oder	XOR	9
	Oder	OR	10
Klammerung	(**Ausdruck**)	( )	1

13.2 Syntax von Ausdrücken

Übersicht

Ausdrücke lassen sich durch das Syntaxdiagramm in Bild 13-1 darstellen. Arithmetische und logische Ausdrücke sowie Vergleichsausdrücke und der Potenzausdruck weisen einige Besonderheiten auf, weshalb diese in den separaten Kapiteln 13.3 bis 13.6 behandelt werden.

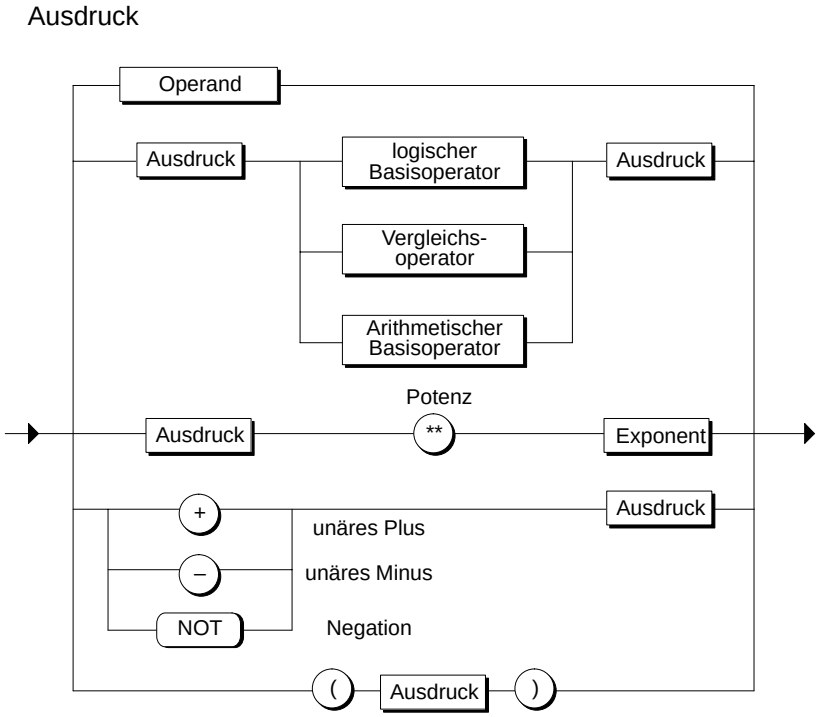


Bild 13-1 *Syntax: Ausdruck*

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.

Auswertungs-reihenfolge

Die Auswertungsreihenfolge eines Ausdrucks ist abhängig von:

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

Regeln

Die Verarbeitung der Ausdrücke erfolgt nach bestimmten Regeln:

- Die Operatoren werden entsprechend ihrer Priorität bearbeitet.
- 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 kann den Operatorenvorrang außer Kraft setzen, d.h. die Klammerung hat die höchste Priorität.
- Ausdrücke in Klammern werden als einzelne Operanden betrachtet und immer als erstes ausgewertet.
- Die Anzahl von linken Klammern muß mit der Anzahl von rechten Klammern übereinstimmen.
- Arithmetische Operationen können nicht auf Zeichen oder logischen Daten angewendet werden. Deshalb sind Ausdrücke wie 'A'+'B' und (n<=0) + (n<0) falsch.

Beispiele

Die verschiedenen Ausdrücke könnten so aufgebaut sein:

```
EB10                        // Operand
A1 AND (A2)                 // logischer Ausdruck
(A3) < (A4)                 // Vergleichsausdruck
3+3*4/2                     // arithmetischer Ausdruck

MESSWERT**2                 // Potenzausdruck
(DIFFERENZ)**DB10.EXPONENT  // Potenzausdruck
(SUMME)**FC100(..)          // Potenzausdruck
```

Beispiel 13-1 Verschiedene Ausdrücke

13.2.1 Operanden

Übersicht

Operanden sind Objekte, mit denen Ausdrücke gebildet werden können. Operanden lassen sich durch das Syntaxdiagramm in Bild 13-2 darstellen.

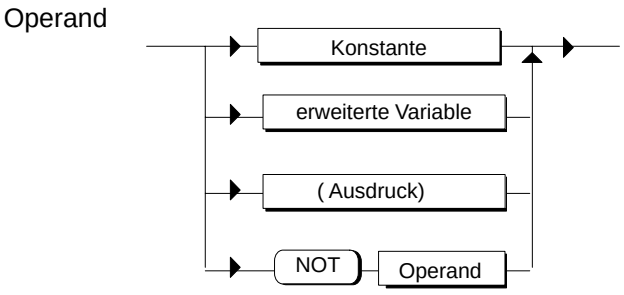


Bild 13-2 *Syntax: Operand*

Konstante

Es können Konstanten mit ihrem numerischen Wert oder mit einem symbolischen Namen oder Zeichenfolgen verwendet werden.

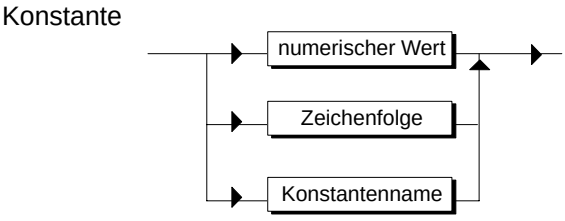


Bild 13-3 *Syntax: Konstante*

Beispiele für gültige Konstanten sind:

```
4_711
4711
30.0
'ZEICHEN'
FAKTOR
```

Erweiterte Variable

Die erweiterte Variable ist ein Oberbegriff für eine Reihe von Variablen, die in Kapitel 14 genauer behandelt werden.

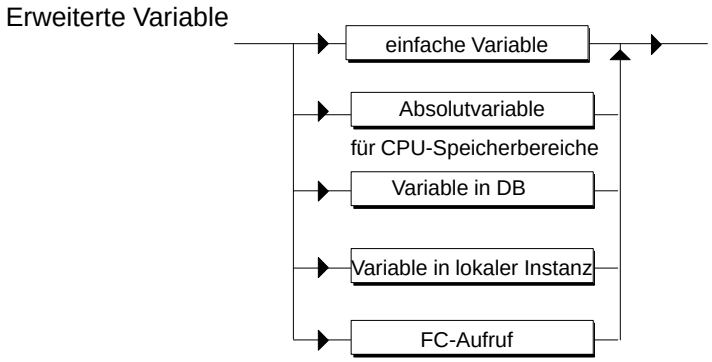


Bild 13-4 *Syntax: Erweiterte Variable*

Beispiele für die erweiterte Variable

Beispiele für gültige Variablen sind:

```
SOLLWERT          einfache Variable
EW10              absolute Variable
E100.5            absolute Variable
DB100.DW[INDEX]   Variable in DB
MOTOR.DREHZAL     Variable in lokaler Instanz
SQR(20)           Standard-Funktion
FC192(SOLLWERT)   Funktionsaufruf
```

Beispiel 13-2 Erweiterte Variablen in Ausdrücken

Hinweis

Bei einem Funktionsaufruf wird das errechnete Ergebnis, der Rückgabewert, anstelle des Funktionsnamens in den Ausdruck eingefügt. Aus diesem Grund sind VOID-Funktionen, die keinen Rückgabewert haben, als Operanden für einen Ausdruck **nicht** zulässig.

13.3 Arithmetische Ausdrücke

Definition

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

Arithmetischer Basisoperator

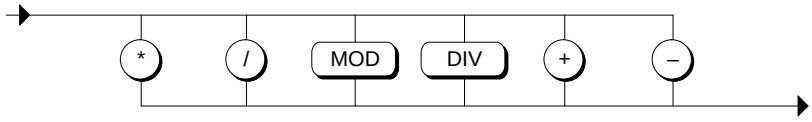


Bild 13-5 *Syntax: Arithmetischer Basisoperator*

Arithmetische Operationen

Tabelle 13-2 faßt die möglichen Operationen zusammen und zeigt, welchem Typ das Ergebnis in Abhängigkeit der Operanden zuzuordnen ist. Dabei steht zur Abkürzung:

ANY_INT	für die Datentypen	INT, DINT
ANY_NUM	für die Datentypen	ANY_INT und REAL

Tabelle 13-2 Arithmetische Operatoren

Operation	Operator	1.Operand	2. Operand	Ergebnis [1]	Priorität
Potenz	**	ANY_NUM	INT	REAL	2
unäres Plus	+	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	
unäres Minus	-	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	
Multiplikation	*	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	
Division	/	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY INT	TIME	
Integer-Division	DIV	ANY_INT	ANY_INT	ANY_INT	4
		TIME	ANY_INT	TIME	
Modulo-Division	MOD	ANY_INT	ANY_INT	ANY_INT	4
Addition	+	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	
		TOD	TIME	TOD	
		DT	TIME	DT	
Subtraktion	–	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	
		TOD	TIME	TOD	
		DATE	DATE	TIME	
		TOD	TOD	TIME	
		DT	TIME	DT	
		DT	DT	TIME	

[1] *Beachten Sie, daß der Ergebnistyp vom mächtigsten Operandentyp bestimmt wird.*

Regeln

Operatoren in arithmetischen Ausdrücken werden in der Reihenfolge ihrer Priorität behandelt (siehe Tabelle 13-2).

- Es empfiehlt sich, negative Zahlen wegen der Übersichtlichkeit in Klammern zu setzen, auch dort wo es nicht notwendig ist.
- Bei Divisionen mit zwei ganzzahligen Operanden vom Datentyp INT liefern die Operatoren “DIV” und “/” dasselbe Ergebnis (siehe Beispiel 13-3).
- Die Divisionsoperatoren ”/”, ”MOD” und ”DIV” erfordern, daß der zweite Operand ungleich Null ist.
- Wenn ein Operand vom Typ INT (Ganzzahl) und der andere vom Typ REAL (Gleitpunktzahl) ist, ist das Ergebnis immer vom Typ REAL.

Beispiele

Die folgenden Beispiele veranschaulichen die Bildung arithmetischer Ausdrücke.

Nehmen wir an, daß i und j Integervariablen sind, deren Werte 11 bzw. –3 sind. Einige Integerausdrücke und ihre entsprechenden Werte werden im Beispiel 13-3 gezeigt.

Ausdruck	Wert
i + j	8
i – j	14
i * j	–33
i DIV j	–3
i MOD j	2
i/j	–3

Beispiel 13-3 Arithmetische Ausdrücke

Nehmen wir an, daß i und j Intergervariablen sind, deren Werte 3 bzw. –5 sind. Dann wird das Ergebnis des arithmetischen Ausdrucks in Beispiel 13-4 (Intergerwert 7) der Variablen WERT zugewiesen.

```
WERT:= i + i * 4 / 2 - (7+i) / (-j) ;
```

Beispiel 13-4 Arithmetischer Ausdruck

13.4 Exponenten

Übersicht

Bild 13-6 soll die Bildung des Exponenten in einem Potenzausdruck (siehe Kapitel 13.2) veranschaulichen. Beachten Sie dabei, daß der Exponent insbesondere auch mit erweiterten Variablen gebildet werden kann.

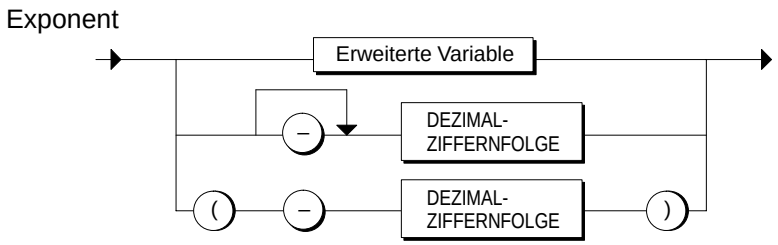


Bild 13-6 *Syntax: Exponent*

```
MESSWERT**2              // Potenzausdruck
(DIFFERENZ)**DB10.EXPONENT //Potenzausdruck
(SUMME)**FC100(..)       // Potenzausdruck
```

Beispiel 13-5 Potenzausdrücke mit verschiedenen Exponenten

13.5 Vergleichsausdrücke

Definition

Ein Vergleichsausdruck ist ein mit Vergleichsoperatoren gebildeter Ausdruck vom Typ BOOL. Vergleichsausdrücke werden durch Kombinationen von Operanden desselben Typs oder derselben Typklasse mit den in Bild 13-7 aufgeführten Operatoren gebildet.

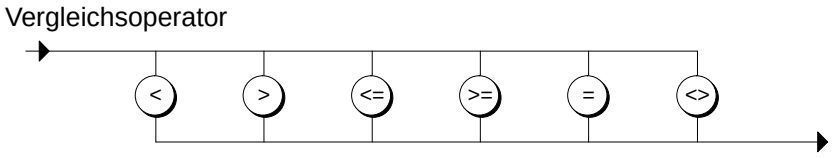


Bild 13-7 *Syntax: Vergleichsoperator*

Vergleichs-operationen

Vergleichsoperatoren vergleichen ihre beiden Operanden bezüglich ihres numerischen Wertes.

```
1.Operand Operator 2.Operand ⇒ boolescher Wert
```

Als Ergebnis erhält man einen Wert, der entweder wahr oder falsch ist. Der Wert ist ”wahr” (TRUE), falls der Vergleich erfüllt ist, und ”falsch” (FALSE), wenn er nicht erfüllt ist.

Regeln

Die folgenden Regeln müssen bei der Bildung von Vergleichsausdrücken beachtet werden:

- Logische Operanden sollten in Klammern stehen, um deutlich zu machen, in welcher Reihenfolge die logischen Operationen durchgeführt werden sollen.
- Logische Ausdrücke können nach dem Gesetz der booleschen Logik verknüpft werden, um Abfragen wie ”wenn a < b **und** b < c, dann...” zu realisieren. Als Ausdruck können Variablen oder Konstanten vom Typ BOOL sowie Vergleichsausdrücke verwendet werden.
- Innerhalb folgender Typklassen sind alle Variablen vergleichbar:
  - `INT, DINT, REAL`
  - `BOOL, BYTE, WORD, DWORD`
  - `CHAR, STRING`
- Bei folgenden Zeittypen sind nur typgleiche Variablen vergleichbar:
  - `DATE, TIME, TOD, DT`
- Beim Vergleich von Zeichen (Typ CHAR) erfolgt die Auswertung entsprechend der ASCII Zeichenfolge.
- S5TIME-Variablen sind nicht vergleichbar.
- Wenn beide Operanden vom Typ DT oder STRING sind, müssen Sie mit den entsprechenden IEC-Funktionen verglichen werden.

Beispiele

Die folgenden Beispiele veranschaulichen die Bildung der Vergleichsausdrücke:

```
// Das Ergebnis des Vergleichausdrucks wird
// negiert.

      IF NOT (ZAEHLER > 5) THEN
            //...
            //...
      END_IF;


// Das Ergebnis des ersten Vergleichsausdrucks
// wird negiert und mit dem Ergebnis des
// zweiten konjugiert

      A:= NOT (ZAEHLER1 = 4) AND (ZAEHLER2 = 10);


// Disjunktion zweier Vergleichsausdrücke
      WHILE (A >= 9) OR (ABFRAGE <> 'n') DO
            //...
            //...
      END_WHILE;
```

Beispiel 13-6 Logische Ausdrücke

13.6 Logische Ausdrücke

Definition

Ein logischer Ausdruck ist ein mit logischen Operatoren gebildeter Ausdruck. Mit den Operatoren (AND, &, XOR, OR) können logische Operanden (Typ BOOL) oder Variablen vom Datentyp BYTE, WORD oder DWORD verknüpft werden, um logische Ausdrücke zu bilden. Um den Wert eines logischen Operanden zu verneinen (d.h. umzukehren), wird der NOT-Operator verwendet.

Logischer Basisoperator

NOT ist kein Basisoperator!
Der Operator wirkt wie Vorzeichen.

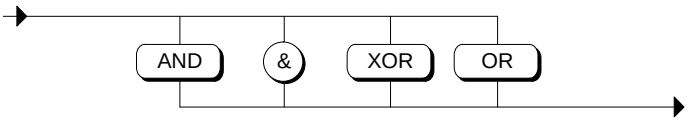


Bild 13-8 *Syntax: Logischer Basisoperator*

Logische Operationen

Tabelle 13-3 gibt Auskunft über die verfügbaren logischen Ausdrücke und die Datentypen für Ergebnis und Operanden. Dabei steht als Abkürzung:

`ANY_BIT` für die Datentypen `BOOL, BYTE, WORD, DWORD`

Tabelle 13-3 Logische Operatoren

Operation	Operator	1.Operand	2. Operand	Ergebnis	Priorität
Negation	NOT	ANY_BIT	-	ANY_BIT	3
Konjunktion	AND	ANY_BIT	ANY_BIT	ANY_BIT	8
Exklusiv Disjunktion	XOR	ANY_BIT	ANY_BIT	ANY_BIT	9
Disjunktion	OR	ANY_BIT	ANY_BIT	ANY_BIT	10

Ergebnisse

Das Ergebnis eines logischen Ausdrucks ist entweder

- 1 (*true*) oder 0 (*false*) bei der Verknüpfung von booleschen Operanden oder
- ein Bitmuster nach der Bitverknüpfung zwischen den beiden Operanden.

Beispiele

Nehmen wir an, daß n eine Intergervariable mit dem Wert 10 ist und s eine Zeichenvariable, die das Zeichen A darstellt. Einige logische Ausdrücke mit diesen Variablen sind dann

```
Ausdruck                          Wert

( n>0 )   AND  ( n<20)      wahr
( n>0 )   AND  ( n<5 )      falsch
( n>0 )   OR   ( n<5 )      wahr
( n>0 )   XOR  ( n<20)      falsch
( n=10 )  AND  ( s='A')     wahr
( n<>5 )  OR   ( s>='A')    wahr
```

Beispiel 13-7 Logische Ausdrücke

Wertzuweisungen

14

Übersicht

Eine Wertzuweisung dient dazu, einer Variablen den Wert eines Ausdrucks zuzuweisen. Der bisherige Wert wird überschrieben.

Kapitelübersicht

Weitere Informationen

In SCL gibt es einfache und strukturierte Anweisungen. Zu den einfachen Anweisungen gehört neben der Wertzuweisung, die Aufrufanweisung und die GOTO-Anweisung. Informationen hierzu finden Sie in den Kapiteln 15 und 16.

Die Kontrollanweisungen für eine Programmverzweigung und für eine Schleifenbearbeitung zählt zu den strukturierten Anweisungen. Erläuterungen dazu finden Sie in Kapitel 15.

14.1 Übersicht

Prinzip

Wertzuweisungen ersetzen den momentanen Wert einer Variablen mit einem neuen Wert, der über einen Ausdruck angegeben wird. Dieser Ausdruck kann auch Bezeichner von Funktionen (FC) enthalten, die dadurch aktiviert werden und entsprechende Werte zurückliefern (Rückgabewert).

Wie das Syntaxdiagramm zeigt, wird der Ausdruck auf der rechten Seite der Wertzuweisung ausgewertet und das Ergebnis in der Variablen abgespeichert, deren Namen auf der linken Seite des Zuweisungszeichens steht. Die Gesamtheit der dabei zugelassenen Variablen zeigt Bild 14-1.

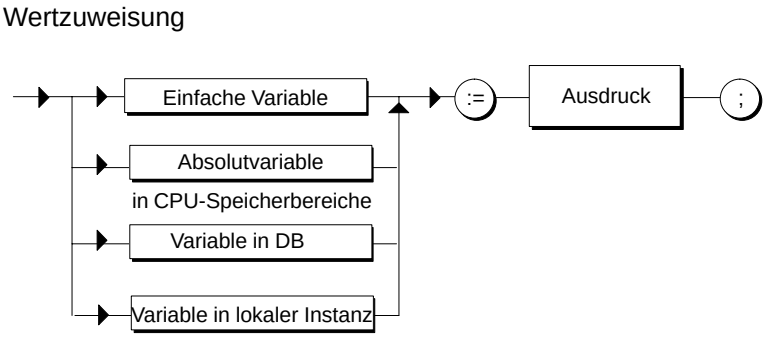


Bild 14-1 *Syntax: Wertzuweisung*

Ergebnis

Der Typ eines Zuweisungsausdrucks ist der Typ des linken Operanden.

14.2 Wertzuweisungen mit Variablen eines elementaren Typs

Zuweisung

Jeder Ausdruck und jede Variable mit einem elementaren Datentyp kann einer anderen typengleichen Variablen zugewiesen werden.

```
Bezeichner := Ausdruck;
Bezeichner := Variable_elementaren_Typ;
```

Beispiele

Folgende Beispiele sind gültige Wertzuweisungen

```
FUNCTION_BLOCK FB10
VAR
      SCHALTER_1  : INT;
      SCHALTER_2  : INT;
      SOLLWERT_1  : REAL;
      SOLLWERT_2  : REAL;
      ABFRAGE_1   : BOOL;
      ZEIT_1      : S5TIME;
      ZEIT_2      : TIME;
      DATUM_1     : DATE;
      TAGESZEIT_1 : TIME_OF_DAY;
END_VAR

BEGIN
// Zuweisung einer Konstanten zu einer Variablen
      SCHALTER_1  := -17;
      SOLLWERT_1  := 100.1;
      ABFRAGE_1   := TRUE;
      ZEIT_1      := TIME#1H_20M_10S_30MS;
      ZEIT_2      := TIME#2D_1H_20M_10S_30MS;
      DATUM_1     := DATE#1996-01-10;

// Zuweisung einer Variablen
      SOLLWERT_1  := SOLLWERT_2;
      SCHALTER_2_ := SCHALTER_1;

// Zuweisung eines Ausdrucks zu einer Variablen
      SCHALTER_2  := SCHALTER_1 * 3;
END_FUNCTION_BLOCK
```

Beispiel 14-1 Wertzuweisungen mit elementaren Datentypen

14.3 Wertzuweisungen mit Variablen vom Typ STRUCT und UDT

Variablen vom Typ STRUCT und UDT

Variablen vom Typ STRUCT und UDT sind strukturierte Variablen, die entweder für eine komplette Struktur oder eine Komponente dieser Struktur stehen.

Gültige Angaben für eine Strukturvariable sind:

```
Abbild           //Bezeichner für eine Struktur
Abbild.element   //Bezeichner für Strukturkomponente
Abbild.feld      //Bezeichner eines einfachen Feldes
                 //innerhalb einer Struktur
Abbild.feld[2,5] //Bezeichner einer Feldkomponente
                 //innerhalb einer Struktur
```

Zuweisung einer kompletten Struktur

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:

```
Structname_1:=Structname_2;
```

Zuweisung von Struktur-komponenten

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

```
Structname_1.element1   := Wert;
Structname_1.element1   := 20.0;
Structname_1.element1   := Structname_2.element1;
Structname_1.feldname1  := Structname_2.feldname1;
Structname_1.feldname[10]:= 100;
```

Beispiele

Die Wertzuweisungen für Daten von Strukturen sollen anhand der folgenden Beispiele verdeutlicht werden:

```
FUNCTION_BLOCK FB10
VAR
      HILFSVAR    :REAL;
      MESSWERT    :STRUCT  //Zielstruktur
            SPANNUNG    :REAL;
            WIDERSTAND  :REAL;
            Einfachfeld :ARRAY[1..2,1..2] OF INT;
      END_STRUCT;

      ISTWERT     :STRUCT  //Quellstruktur
            SPANNUNG    :REAL;
            WIDERSTAND  :REAL;
            Einfachfeld :ARRAY[1..2,1..2] OF INT;
      END_STRUCT;
END_VAR
BEGIN
//Zuweisung einer kompletten Struktur an
//eine komplette Struktur
      MESSWERT:= ISTWERT;

//Zuweisung einer Strukturkomponente zu einer
//Strukturkomponente
      MESSWERT.SPANNUNG:= ISTWERT.SPANNUNG;

// Zuweisung einer Strukturkomponente zu einer
// typengleichen Variablen
      HILFSVAR:= ISTWERT.WIDERSTAND;

// Zuweisung einer Konstanten zu einer
// Strukturkomponente
      MESSWERT.WIDERSTAND:= 4.5;

// Zuweisung einer Konstanten zu einem Element
// eines einfachen Feldes
      MESSWERT.EINFACHFELD[1,2]:= 4;
END_FUNCTION_BLOCK
```

Beispiel 14-2 Wertzuweisungen mit Variablen vom Typ STRUCT

14.4 Wertzuweisungen mit Variablen vom Typ ARRAY

Feldvariable

Ein Feld besteht aus 1 bis max. 6 Dimensionen und Feldelementen, die alle vom selben Typ sind. Für die Zuweisung von Feldern an eine Variable gibt es zwei Varianten:

Sie können **komplette** Felder oder **Teilfelder** referenzieren. Ein komplettes Feld referenzieren Sie, indem Sie den Variablennamen des Feldes angeben.

```
feldname_1
```

Ein einzelnes Feldelement wird mit dem Feldnamen gefolgt von Indexwerten in eckigen Klammern angesprochen. Für jede Dimension steht ein Index zur Verfügung. Sie werden durch Komma getrennt und ebenfalls in eckige Klammern eingeschlossen. Ein Index muß ein arithmetischer Ausdruck vom Datentyp `INT` sein.

```
feldname_1[2]
feldname_1[4,5]
```

Zuweisung eines kompletten Feldes

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:

```
feldname_1:=feldname_2;
```

Zuweisung einer Feldkomponente

Eine Wertzuweisung für ein zulässiges Teilfeld erhalten Sie, indem Sie in den eckigen Klammern hinter dem Namen des Feldes von rechts beginnend Indizes weglassen. Damit sprechen Sie einen Teilbereich an, dessen Dimensionsanzahl gleich der Anzahl der weggelassenen Indizes ist.

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:

```
feldname_1[i]:=feldname_2[j];
feldname_1[i]:=ausdruck;
bezeichner_1 :=feldname_1[i];
```

Beispiele

Die Wertzuweisungen für Felder sollen anhand der nachstehenden Beispiele verdeutlicht werden:

```
FUNCTION_BLOCK FB3
VAR
      SOLLWERTE   :ARRAY [0..127] OF INT;
      ISTWERTE    :ARRAY [0..127] OF INT;

// Vereinbarung einer Matrix
// (=zweidimensionales Feld)
// mit 3 Zeilen und 4 Spalten
      REGLER: ARRAY [1..3, 1..4] OF INT;

// Vereinbarung eines Vektors
// (=eindimensionales Feld)
// mit 4 Komponenten
      REGLER_1: ARRAY [1..4] OF INT;
END_VAR

BEGIN
// Zuweisung eines kompletten Feldes zu einem
// Feld
      SOLLWERTE:= ISTWERTE;

// Zuweisung eines Vektors zu der zweiten Zeile
// des Feldes REGLER
      REGLER [2]:= REGLER_1;

//Zuweisung einer Feldkomponente zu einer
//Feldkomponente des Feldes REGLER
      REGLER [1,4]:= REGLER_1 [4];
END_FUNCTION_BLOCK
```

Beispiel 14-3 Wertzuweisungen für Felder

14.5 Wertzuweisungen mit Variablen vom Typ STRING

STRING-Variable

Eine Variable vom Datentyp STRING enthält eine Zeichenkette von maximal 254 Zeichen.

Zuweisung

Jeder Variablen vom Datentyp STRING kann eine andere typengleiche Variable zugewiesen werden. Gültige Zuweisungen sind:

```
stringvariable_1 := Stringliteral ;
stringvariable_1 := stringvariable_2 ;
```

Beispiel

Die Wertzuweisungen mit STRING-Variablen sollen anhand der nachstehenden Beispiele verdeutlicht werden:

```
FUNCTION_BLOCK FB3
VAR
      ANZEIGE_1   : STRING[50] ;
      STRUKTUR1   : STRUCT
            ANZEIGE_2 : STRING[100] ;
            ANZEIGE_3 : STRING[50] ;
      END_STRUCT;
END_VAR
BEGIN
// Zuweisung einer Konstanten zu einer STRING-
// Variable

      ANZEIGE_1 := 'Fehler in Modul 1';
// Zuweisung einer Strukturkomponente zu einer
// STRING-Variable.

      ANZEIGE_1 := STRUKTUR1.ANZEIGE_3;

// Zuweisung einer STRING-Variable zu
// einer STRING-Strukturkomponente

      If ANZEIGE_1 <> STRUKTUR.ANZEIGE_3 THEN
         STRUKTUR.ANZEIGE_3 := ANZEIGE_1;
      END_IF;
END_FUNCTION_BLOCK
```

Beispiel 14-4 Wertzuweisungen für STRING-Variablen

14.6 Wertzuweisungen mit Variablen vom Typ DATE_AND_TIME

DATE_AND_TIME-Variable

Der Datentyp DATE_AND_TIME definiert einen Bereich mit 64 bit (8 byte) für die Angabe von Datum und Uhrzeit.

Zuweisung

Jeder Variablen vom Datentyp DATE_AND_TIME kann eine andere typengleiche Variable oder Konstante zugewiesen werden. Gültige Zuweisungen sind:

```
dtvariable_1 := Datum_und_Zeitliteral;
dtvariable_1 := dtvariable_2;
```

Beispiel

Die Wertzuweisungen mit DATE_AND_TIME-Variablen sollen anhand der nachstehenden Beispiele verdeutlicht werden:

```
FUNCTION_BLOCK FB3
VAR
      ZEIT_1      : DATE_AND_TIME;
      STRUKTUR1   : STRUCT
            ZEIT_2 : DATE_AND_TIME;
            ZEIT_3 : DATE_AND_TIME;
      END_STRUCT;
END_VAR

BEGIN
// Zuweisung einer Konstanten zu einer
// DATE_AND_TIME-Variable
ZEIT_1 := DATE_AND_TIME#1995-01-01-12:12:12.2;
STRUKTUR1.ZEIT_3 := DT#1995-02-02-11:11:11;

// Zuweisung einer Strukturkomponente zu einer
// DATE_AND_TIME-Variablen.

ZEIT_1 := STRUKTUR1.ZEIT_2;


// Zuweisung einer DATE_AND_TIME-Variable
// zu einer DATE_AND_TIME-Strukturkomponente
If ZEIT_1 < STRUKTUR1.ZEIT_3 THEN
   STRUKTUR1.ZEIT_3 := ZEIT_1;
END_IF;
END_FUNCTION_BLOCK
```

Beispiel 14-5 Wertzuweisungen für DATE_AND_TIME-Variablen

14.7 Wertzuweisungen mit Absolutvariablen für Speicherbereiche

Absolutvariable

Die Absolutvariable referenziert die gültigen Speicherbereiche einer CPU. Bei der Wertzuweisung haben Sie die drei in Kapitel 12 beschriebenen Möglichkeiten, diese Bereiche anzusprechen.

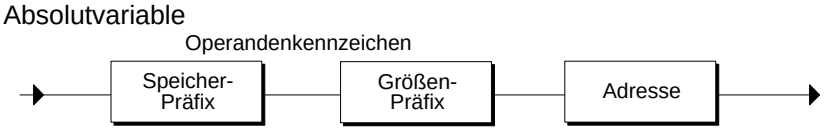


Bild 14-2 *Syntax: Absolutvariable*

Zuweisung

Sie können mit Ausnahme der Peripherieeingänge und Eingänge über das Prozeßabbild jeder Absolutvariablen eine typengleiche Variable oder einen typengleichen Ausdruck zuweisen.

Beispiel

Die Wertzuweisungen für Absolutvariablen sollen anhand der nachstehenden Beispiele verdeutlicht werden:

```
FUNCTION_BLOCK_FB10
VAR
      STATUSWORT1 : WORD;
      STATUSWORT2 : BOOL;
      STATUSWORT3 : BYTE;
      STATUSWORT4 : BOOL;
      ADRESSE     : INT:= 10;
END_VAR
BEGIN
    // Zuweisung eines Eingangsworts an eine
    // Variable (einfacher Zugriff)
      STATUSWORT1 := EW4 ;

    // Zuweisung eines Ausgangsbits an eine
    // Variable (einfacher Zugriff)
      STATUSWORT2 := a1.1 ;

    // Zuweisung eines Eingangsbytes an eine
    // Variable (indizierter Zugriff)
      STATUSWORT3 := EB[ADRESSE];

    // Zuweisung eines Eingangsbits an eine
    // Variable (indizierter Zugriff)
      FOR ADRESSE := 0 TO 7 BY 1 DO
            STATUSWORT4 := e[1,ADRESSE] ;
      END_FOR;
END_FUNCTION_BLOCK
```

Beispiel 14-6 Wertzuweisungen für Absolutvariablen

14.8 Wertzuweisungen mit globalen Variablen

Variablen in DB

Auch der Zugriff auf globale Variablen in Datenbausteinen erfolgt mittels einer Wertzuweisung an typgleiche Variablen oder umgekehrt. Sie haben die Möglichkeit strukturiert, absolut oder indiziert zuzugreifen (Kapitel 12).

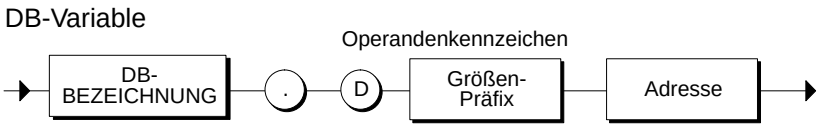


Bild 14-3 *Syntax: DB-Variable*

Zuweisung

Sie können jeder globalen Variablen eine typengleiche Variable oder einen typgleichen Ausdruck zuweisen. Gültige Zuweisungen sind:

```
DB11.DW10 := 20;
DB11.DW10 := Status;
```

Beispiele

Das folgende Beispiel setzt voraus, daß im Datenbaustein DB11 eine Variable ″ZAHL″ vom Datentyp INTEGER und eine Struktur ″ZAHL1″ mit der Komponente ″ZAHL2″ vom Typ INTEGER vereinbart wurde.

```
// Benötigter Datenbaustein DB11
DATA_BLOCK DB11
STRUCT
      ZAHL  :     INT  := 1;
      ZAHL1 :     STRUCT
            ZAHL2 :     INT := 256;
      END_STRUCT;
      WORT3 :     WORD := W#16#aa;
      WORT4 :     WORD := W#16#aa;
      WORT5 :     WORD := W#16#aa;
      WORT6 :     WORD := W#16#aa;
      WORT7 :     WORD := W#16#aa;
      WORT8 :     WORD := W#16#aa;
      WORT9 :     WORD := W#16#aa;
      WORT10:     WORD;
END_STRUCT

BEGIN
WORT10 := W#16#bb;
END_DATA_BLOCK
```

Beispiel 14-7 Wertzuweisungen für globale Variablen

Der Datenbaustein DB11 kann dann z. B. folgendermaßen verwendet werden:

```
VAR
      REGLER_1    : ARRAY [1..4] OF INT;
      STATUSWORT1 : WORD ;
      STATUSWORT2 : ARRAY [1..10] OF INT;
      STATUSWORT3 : INT
      ADRESSE     : INT ;
END_VAR
BEGIN
      // Zuweisen des Worts 10 aus DB11 an eine
      // Variable (einfacher Zugriff)
      STATUSWORT1 := DB11.DW10
      // Der 1.Feldkomponente wird die Variable
      // ”ZAHL” aus DB11 zugewiesen
      // (strukturierter Zugriff):
      REGLER_1[1] := DB11.ZAHL;
       // Zuweisen der Strukturkomponente ”ZAHL2”
       // der Struktur ”ZAHL1” an die Variable
       // Statuswort3
      STATUSWORT3 := DB11.ZAHL1.ZAHL2
      // Zuweisen eines Worts mit Index ADRESSE aus
      // DB11 an eine Variable
      // (indizierter Zugriff)
      FOR ADRESSE := 1 TO 10 BY 1 DO
       STATUSWORT2[ADRESSE]:= DB11.DW[ADRESSE] ;
      END_FOR;
```

Beispiel 14-8 Wertzuweisungen für globale Variablen eines DB

Kontrollanweisungen

15

Übersicht

Nur wenige Bausteine können Sie so programmieren, daß alle Anweisungen bis zum Bausteinende 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 SCL-Bausteins.

Kapitelübersicht

15.1 Übersicht

Auswahl-anweisungen

In Programmen tritt häufig das Problem auf, daß in Abhängigkeit von bestimmten Bedingungen verschiedene Anweisungen durchgeführt werden sollen. Mit einer Auswahlanweisung haben Sie die Möglichkeit, den Programmfluß in 2 bis n alternative Anweisungsfolgen zu verzweigen.

Tabelle 15-1 Arten von Verzweigungen

Verzweigungs-art	Funktion
IF-Anweisung	Mit der IF-Anweisung können Sie den Programmfluß in Abhängigkeit von einer Bedingung, die entweder TRUE oder FALSE ist, in eine von zwei Alternativen verzweigen.
CASE-Anweisung	Mit einer CASE-Anweisung können Sie den Programmfluß im Sinne einer 1:n-Verzweigung steuern, indem Sie eine Variable einen Wert aus n möglichen annehmen lassen.

Wiederholungs-anweisungen

Die Schleifenbearbeitung können Sie mit Hilfe von Wiederholungsanweisungen steuern. Eine Wiederholungsanweisung gibt an, welche Teile Ihres Programms, in Abhängigkeit von bestimmten Bedingungen, wiederholt werden sollen.

Tabelle 15-2 Arten von Anweisungen zur Schleifenbearbeitung

Verzweigungs-art	Funktion
`FOR`-Anweisung	dient zur Wiederholung einer Folge von Anweisungen, solange die Laufvariable innerhalb des angegebenen Wertebereichs liegt.
`WHILE`-Anweisung	dient zur Wiederholung einer Folge von Anweisungen, solange eine Durchführungsbedingung erfüllt ist.
`REPEAT`-Anweisung	dient zur Wiederholung einer Folge von Anweisungen, bis eine Abbruchbedingung erfüllt ist.

Sprung-anweisungen

Ein Programmsprung bewirkt den sofortigen Sprung zu einer angegebenen Sprungmarke und damit zu einer anderen Anweisung innerhalb desselben Bausteines.

Tabelle 15-3 Arten von Sprunganweisungen

Verzweigungs-art	Funktion
`CONTINUE`-Anweisung	dient zum Abbruch der Ausführung des momentanen Schleifen-durchlaufes.
`EXIT`-Anweisung	dient zum Verlassen einer Schleife an beliebiger Stelle und unabhängig vom Erfülltsein der Abbruchbedingung.
`GOTO`-Anweisung	bewirkt den sofortigen Sprung zu einer angegeben Sprungmarke.
`RETURN`-Anweisung	bewirkt das Verlassen eines aktuellen bearbeiteten Bausteins.

Bedingungen

Die Bedingung ist entweder ein Vergleichsausdruck oder ein logischer Ausdruck. Die Bedingung ist vom Typ BOOL und kann die beiden Werte TRUE oder FALSE annehmen.

Beispiele für gültige **Vergleichsausdrücke** sind:

```
ZAEHLER<=100
SQR(A)>0.005
Anwort = 0
SALDO>=UEBERTRAG
ch1< 'T'
```

Beispiele für die Benutzung von Vergleichsausdrücken mit **logischen** Operatoren sind:

```
(ZAEHLER<=100) AND(CH1<'*')
(SALDO<100.0) OR (STATUS ='R')
(Antwort<0)OR((Anwort>5.0) AND (ANTWORT<10.0))
```

Hinweis

Beachten Sie, daß die logischen Operanden (hier Vergleichsausdrücke) in Klammern stehen, um jede Mehrdeutigkeit über die Reihenfolge bei der Auswertung zu vermeiden.

15.2 IF-Anweisung

Prinzip

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

IF-Anweisung

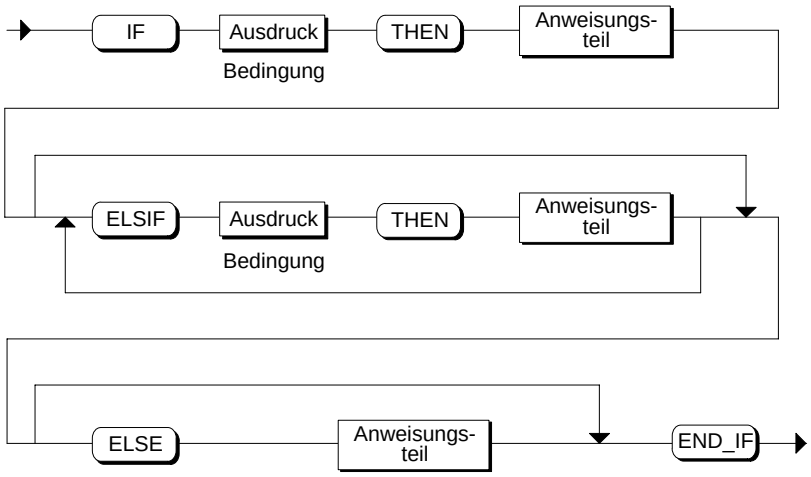


Bild 15-1 *Syntax: IF-Anweisung*

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

Ausführung

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, andernfalls werden die Ausdrücke in den ELSIF-Zweigen ausgewertet.
2. 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).

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

Beachten Sie, daß die Anweisung END_IF mit einem Semikolon abzuschließen ist.

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.

Beispiel

Beispiel 15-1 veranschaulicht den Gebrauch der IF-Anweisung:

```
IF E1.1 THEN
      N:= 0;
      SUM:= 0;
      OK:= FALSE; // OK-Flag auf FALSE setzen
ELSIF START = TRUE THEN
      N:= N + 1;
      SUM:= SUM + N;
ELSE
      OK:= FALSE;
END_IF;
```

Beispiel 15-1 IF-Anweisungen

15.3 CASE-Anweisung

Prinzip

Die CASE-Anweisung dient der 1 aus n Auswahl eines Programmteils. Diese Auswahl beruht auf dem laufenden Wert eines Auswahl-Ausdrucks.

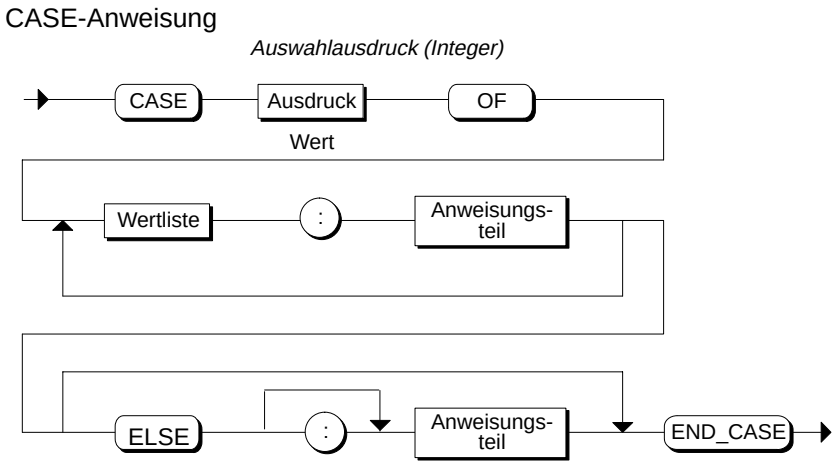


Bild 15-2 *Syntax: CASE-Anweisung*

Ausführung

Die CASE-Anweisung wird nach folgenden Regeln bearbeitet.:

1. Bei der Abarbeitung der CASE-Anweisung wird überprüft, ob der Wert des Auswahl-Ausdrucks in einer angegebenen Wertliste enthalten ist. Jeder Wert in dieser Liste stellt einen der erlaubten Werte für den Auswahlausdruck dar. Der Auswahl-Ausdruck muß einen Wert vom Typ INTEGER liefern.
2. Bei Übereinstimmung wird der der Liste zugeordnete Anweisungsteil ausgeführt.
3. Der ELSE-Zweig ist optional. Er wird ausgeführt, wenn der Vergleichsvorgang keine Übereinstimmung ergibt.

Hinweis

Beachten Sie, daß die Anweisung END_CASE mit einem Semikolon abzuschließen ist.

Wertliste

Sie enthält die erlaubten Werte für den Auswahl-Ausdruck.

Wertliste

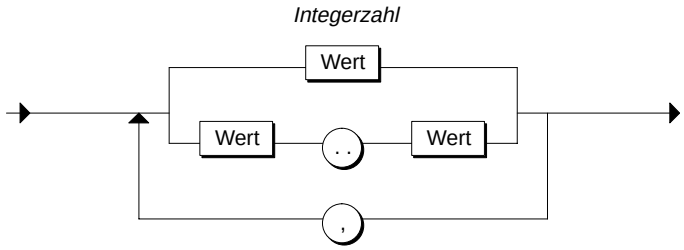


Bild 15-3 *Syntax: Wertliste*

Regeln

Bei der Formulierung der Wertliste müssen Sie beachten, daß

- jede Wertliste mit einer Konstanten, einer Konstantenliste oder einem Konstantenbereich beginnen kann.
- die Werte innerhalb der Werteliste INTEGER-Werte sein müssen
- jeder Wert nur einmal vorkommen darf.

Beispiele

Beispiel 15-2 veranschaulicht den Gebrauch der CASE-Anweisung. Die Variable TW ist regelgemäß vom Typ INTEGER.

```
CASE TW OF
      1:    DISPLAY   := OVEN_TEMP;
      2:    DISPLAY   := MOTOR_SPEED;
      3:    DISPLAY   := GROSS_TARE;
                  AW4 := 16#0003;
      4..10:DISPLAY   := INT_TO_DINT (TW);
                  AW4 := 16#0004;
      11,13,19:DISPLAY:= 99;
                  AW4 := 16#0005;
ELSE:       DISPLAY   := 0;
            TW_ERROR  := 1;
END_CASE;
```

Beispiel 15-2 CASE-Anweisung

Hinweis

Achten Sie darauf, daß die Laufzeit der Schleifen nicht zu lang wird, da sonst die CPU mit Quittungsverzug in STOP geht.

15.4 FOR-Anweisung

Prinzip

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 INT oder DINT sein.

FOR-Anweisung

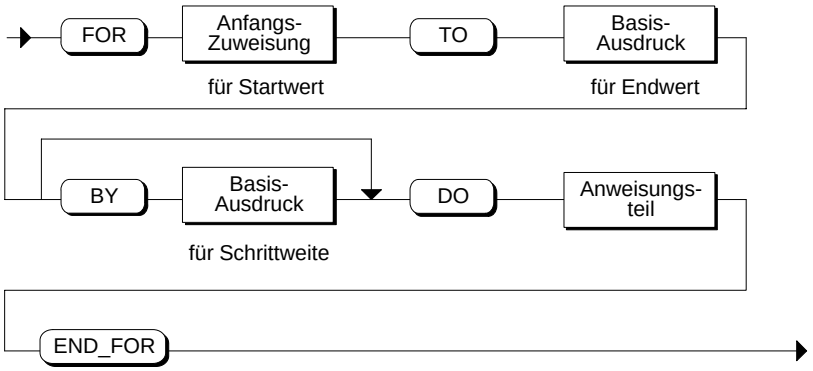


Bild 15-4 *Syntax: FOR-Anweisung*

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

Ausführung

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.
2. Bei jedem Durchlauf wird überprüft, ob die Bedingung

   ```
   |Startwert| <= |Endwert|
   ```

   erfüllt ist. Ist die Bedingung erfüllt, wird die Anweisungsfolge ausgeführt, ist die Bedingung nicht erfüllt, wird die Schleife und damit die Anweisungsfolge übersprungen.

Hinweis

Beachten Sie, daß die Anweisung END_FOR mit einem Semikolon abzuschließen ist.

Anfangszuweisung

Für die Bildung des Startwertes der Laufvariablen steht die in Bild 15-5 dargestellte Anfangszuweisung zur Verfügung.

Anfangszuweisung

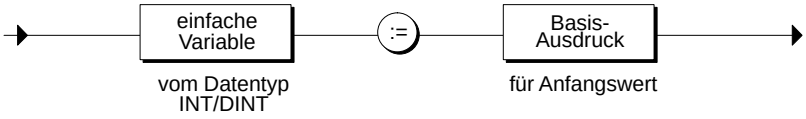


Bild 15-5 *Syntax: Anfangszuweisung*

Beispiele:

```
FOR I  := 1 TO 20 DO
FOR I  := 1 TO (Anfang+J) DO
```

Endwert und Schrittweite

Für die Bildung des Endwertes und der gewünschten Schrittweite können Sie jeweils einen Basisausdruck bilden.

Regeln

Für die FOR-Anweisung gelten folgende Regeln:

- Die Angabe von *`BY [Schrittweite]`* kann entfallen. Ist keine Schrittweite spezifiziert, dann beträgt sie `+1`.
- Anfangswert, Endwert und Schrittweite sind Ausdrücke (siehe Kapitel 13). Die Auswertung erfolgt einmalig am Beginn der Ausführung der FOR-Anweisung.
- Eine Veränderung der beiden Werte für Endwert und Schrittweite während der Ausführung der Schleife ist nicht erlaubt.

Beispiel

Beispiel 15-3 veranschaulicht den Gebrauch der FOR-Anweisung:

```
FUNCTION_BLOCK SUCHEN
VAR
      INDEX       : INT;
      KENNWORT    : ARRAY [1..50] OF STRING;
END_VAR

BEGIN
FOR INDEX:= 1 TO 50 BY 2 DO
      IF KENNWORT [INDEX] = 'KEY' THEN
      EXIT;
      END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

Beispiel 15-3 FOR-Anweisung

15.5 WHILE-Anweisung

Prinzip

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

WHILE-Anweisung

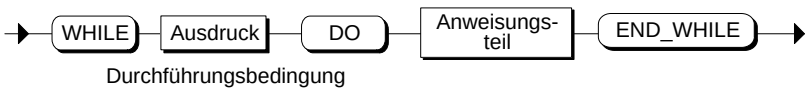


Bild 15-6 *Syntax: WHILE-Anweisung*

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

Ausführung

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 auch schon bei der ersten Auswertung der Fall sein.

Hinweis

Beachten Sie, daß die Anweisung END_WHILE mit einem Semikolon abzuschließen ist.

Beispiel

Beispiel 15-4 veranschaulicht den Gebrauch der WHILE-Anweisung:

```
FUNCTION_BLOCK SUCHEN
VAR
      INDEX       : INT;
      KENNWORT    : ARRAY [1..50] OF STRING;
END_VAR
BEGIN
INDEX:= 1;
WHILE INDEX <= 50 AND KENNWORT[INDEX] <> 'KEY' DO
      INDEX:= INDEX + 2;
END_WHILE;
END_FUNCTION_BLOCK
```

Beispiel 15-4 WHILE-Anweisung

15.6 REPEAT-Anweisung

Prinzip

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.

REPEAT-Anweisung

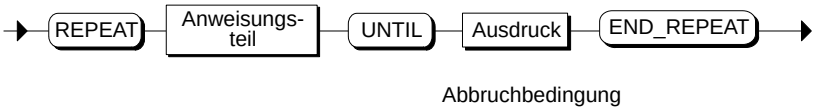


Bild 15-7 *Syntax: REPEAT-Anweisung*

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

Hinweis

Beachten Sie, daß die Anweisung END_REPEAT mit einem Semikolon abzuschließen ist.

Beispiel

Beispiel 15-5 veranschaulicht den Gebrauch der REPEAT-Anweisung

```
FUNCTION_BLOCK SUCHEN
VAR
      INDEX       : INT;
      KENNWORT    : ARRAY [1..50] OF STRING;
END_VAR

BEGIN
INDEX:= 0;
REPEAT
      INDEX:= INDEX + 2;
UNTIL
      INDEX > 50 OR KENNWORT[INDEX] = 'KEY'
END_REPEAT;
END_FUNCTION_BLOCK
```

Beispiel 15-5 REPEAT-Anweisung

15.7 CONTINUE-Anweisung

Prinzip

Eine CONTINUE-Anweisung bewirkt den Abbruch des momentanen Schleifendurchlaufes einer Wiederholungsanweisung (FOR, WHILE oder REPEAT- Anweisung) und das Wiederaufsetzen innerhalb der Schleife.

CONTINUE-Anweisung

Bild 15-8 *Syntax: CONTINUE-Anweisung*

In einer WHILE-Schleife entscheidet die Anfangsbedingung und bei einer REPEAT-Schleife die Endbedingung, ob eine Anweisungsfolge wiederholt wird.

In einer FOR-Anweisung wird direkt nach einer CONTINUE-Anweisung die Laufvariable um die angegebene Schrittweite erhöht.

Beispiel

Beispiel 15-6 veranschaulicht den Gebrauch der CONTINUE-Anweisung:

```
FUNCTION_BLOCK_CONTINUE
VAR
      INDEX :INT;
      FELD  :ARRAY[1..100] OF INT;
END_VAR
BEGIN
INDEX:= 0;
WHILE INDEX <= 100 DO
      INDEX:= INDEX + 1;
      // Wenn FELD[INDEX] gleich INDEX ist,
      // dann wird FELD [INDEX] nicht verändert:
            IF FELD[INDEX] = INDEX THEN
                  CONTINUE;
            END_IF;
            FELD[INDEX]:= 0;
      // Weitere Anweisungen..
      //....
END_WHILE;
END_FUNCTION_BLOCK
```

Beispiel 15-6 CONTINUE-Anweisung

15.8 EXIT-Anweisung

Prinzip

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

EXIT-Anweisung

Bild 15-9 *Syntax: EXIT-Anweisung*

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

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

Beispiel

Beispiel 15-7 veranschaulicht den Gebrauch der EXIT-Anweisung:

```
FUNCTION_BLOCK_EXIT
VAR
      INDEX_1        : INT;
      INDEX_2        : INT;
      INDEX_GESUCHT  : INT;
      KENNWORT       : ARRAY[1..51] OF STRING;
END_VAR
BEGIN
INDEX_2     := 0;
FOR INDEX_1 := 1 TO 51 BY 2 DO
     // Verlassen der FOR-Schleife, wenn
     // KENNWORT[INDEX_1] gleich 'KEY' ist:
     IF KENNWORT[INDEX_1] = 'KEY' THEN
           INDEX_2:= INDEX_1;
           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_GESUCHT:= INDEX_2;
END_FUNCTION_BLOCK
```

Beispiel 15-7 EXIT-Anweisung

15.9 GOTO-Anweisung

Prinzip

Mit einer GOTO-Anweisung können Sie einen Programmsprung realisieren. Sie bewirkt den sofortigen Sprung zu einer angegebenen Sprungmarke und damit zu einer anderen Anweisung innerhalb desselben Bausteines.

GOTO-Anweisungen sollten nur in Sonderfällen, z. B. Fehlerbearbeitung eingesetzt werden. Nach den Regeln der strukturierten Programmierung sollte die GOTO-Anweisung nicht verwendet werden.

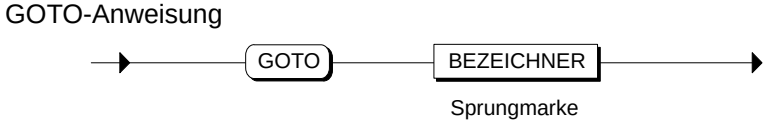


Bild 15-10 *Syntax: GOTO-Anweisung*

Dabei bezeichnet Sprungmarke eine Marke im LABEL/END_LABEL Vereinbarungsblock. Diese Marke ist der Anweisung vorangestellt, die nach dem GOTO als nächste ausgeführt werden soll.

Regeln

Bei der Verwendung der GOTO-Anweisung sind folgende Regeln zu beachten:

- Das Ziel einer Sprunganweisung muß innerhalb desselben Bausteins liegen.
- Das Sprungziel muß eindeutig sein.
- Einsprung in einen Schleifenblock ist nicht zulässig. Aussprung aus einem Schleifenblock ist möglich.

Beispiel

Beispiel 15-8 veranschaulicht den Gebrauch der GOTO-Anweisung:

```
FUNCTION_BLOCK FB3//GOTO_BSP
VAR
        INDEX       : INT;
        A           : INT;
        B           : INT;
        C           : INT;
        KENNWORT    : ARRAY[1..51] OF STRING;
END_VAR
LABEL
        MARKE1,  MARKE2, MARKE3;
END_LABEL
BEGIN
IF A > B THEN GOTO MARKE1;
        ELSIF A > C THEN GOTO MARKE2;
END_IF;
//...
MARKE1  :       INDEX:= 1;
                GOTO MARKE3;
MARKE2  :       INDEX:= 2;
//...
MARKE3  :       ;
//...

END_FUNCTION_BLOCK
```

Beispiel 15-8 Sprunganweisung GOTO

15.10 RETURN-Anweisung

Prinzip

Eine RETURN-Anweisung bewirkt das Verlassen des aktuell bearbeiteten Bausteins (OB, FB, FC) und die Rückkehr zum aufrufenden Baustein bzw. zum Betriebssystem, wenn ein OB verlassen wird.

RETURN-Anweisung

Bild 15-11 *Syntax: RETURN-Anweisung*

Hinweis

Eine RETURN-Anweisung am Ende des Anweisungsteils eines Codebausteins bzw. des Vereinbarungsteils eines Datenbausteins ist redundant, da diese automatisch ausgeführt wird.

Aufruf von Funktionen und Funktionsbausteinen

16

Übersicht

Sie können aus einem SCL-Baustein heraus Funktionen (FC) oder Funktionsbausteine (FB) aufrufen. Aufrufbar sind:

- Funktionen und Funktionsbausteine, die in SCL erstellt wurden.
- Funktionen und Funktionsbausteine, die in einer anderen STEP 7-Sprache (z. B AWL, KOP) programmiert sind.
- Systemfunktionen (SFC) und Systemfunktionsbausteine (SFB), die im Betriebssystem der von Ihnen verwendeten CPU verfügbar sind.

Kapitelübersicht

16.1 Aufruf und Parameterübergabe

Parameter-übergabe

Beim Aufruf von Funktionen oder Funktionsbausteinen 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.

Prinzip

Im folgenden Beispiel eines Funktionsaufrufs werden je ein Eingangs-, Durchgangs-, und Ausgangsparameter angegeben.

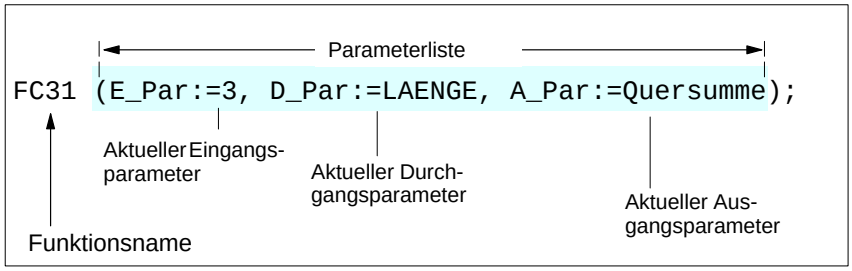


Bild 16-1 *Prinzip der Parameterübergabe*

Die Angabe der Parameter hat die Form einer Wertzuweisung (siehe Bild 16-2). Durch diese Wertzuweisung weisen Sie den Parametern, die Sie im Vereinbarungsteil des aufgerufenen Bausteins definiert haben (Formalparametern), einen Wert (Aktualparameter) zu.

Aktualparameter		Formalparameter
3	⇒	E_Par
LAENGE	⇔	D_Par
Quersumme	⇐	A_Par

Bild 16-2 *Wertzuweisung innerhalb der Parameterliste*

Formalparameter

Die Formalparameter sind die Parameter, die der Baustein beim Aufruf erwartet. Sie sind lediglich ”Platzhalter” für die Aktualparameter, die dem Baustein beim Aufruf übergeben werden. Diese Parameter haben Sie im Vereinbarungsteil eines Bausteins (FB oder FC) definiert.

Tabelle 16-1 Zulässige Vereinbarungsblöcke für Formalparameter

Vereinbarungsblöcke	Daten	Schlüsselwort
Parameterblock	Eingangsparameter	VAR_INPUT *Vereinbarungsliste* END_VAR
	Ausgangsparameter	VAR_OUTPUT *Vereinbarungsliste* END_VAR
	Durchgangsparameter	VAR_IN_OUT *Vereinbarungsliste* END_VAR

16.2 Aufruf von Funktionsbausteinen (FB oder SFB)

Globale und lokale Instanz

Beim Aufruf eines Funktionsbausteins können Sie mit SCL

- sowohl globale Instanz-Datenbausteine
- als auch lokale Instanzbereiche des aktuellen Instanz-Datenbausteins

benutzen. Der Aufruf eines FBs als lokale Instanz unterscheidet sich vom Aufruf als globale Instanz in der Speicherung der Daten. Die Daten werden hier nicht in einem gesonderten DB abgelegt, sondern in dem Instanz-Datenbaustein des aufrufenden FBs eingeschachtelt.

FB-Aufruf

FB: Funktionsbaustein
SFB: Systemfunktionsbaustein

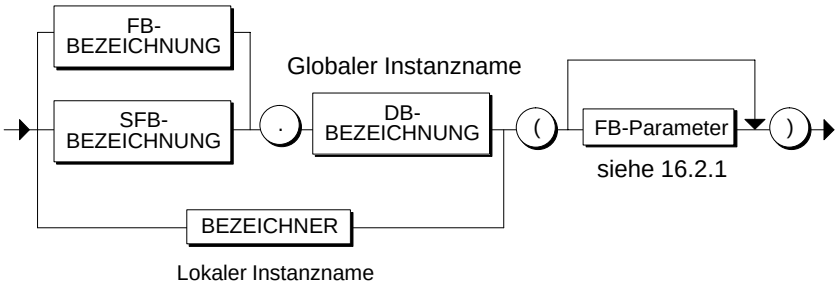


Bild 16-3 *Syntax: FB-Aufruf*

Aufruf als globale Instanz

Der Aufruf erfolgt in einer Aufrufanweisung unter Angabe:

- des Namens des Funktionsbausteins bzw. Systemfunktionsbausteins (FB- oder SFB-Bezeichnung),
- des Instanz-Datenbausteins (DB-Bezeichnung),
- sowie der Parameterversorgung (FB-Parameter).

Ein Aufruf einer globalen Instanz kann absolut oder symbolisch definiert sein.

Absoluter Aufruf:

```
FB10.DB20 (X1:=5,X2:=78,......);
```

Parameterversorgung

Symbolischer Aufruf:

```
ANTRIEB.EIN (X1:=5,X2:=78,......);
```

Bild 16-4 *Aufruf von FB10 mit Instanz-Datenbaustein DB20*

Aufruf als lokale Instanz

Der Aufruf erfolgt in einer Aufrufanweisung unter Angabe:

- des lokalen Instanznamens (BEZEICHNER)
- der Parameterversorgung (FB-Parameter)

Ein Aufruf einer lokalen Instanz ist immer symbolisch, z. B.:

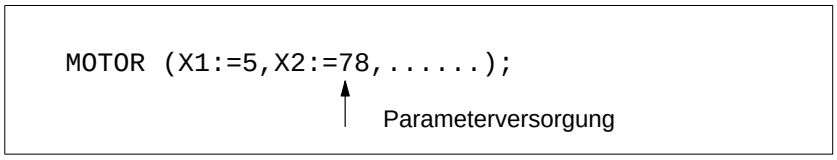


Bild 16-5 *Aufruf einer lokalen Instanz*

16.2.1 FB-Parameter

Prinzip

Beim Aufruf eines Funktionsbausteins – als globale oder lokale Instanz – müssen Sie in der Parameterliste unterscheiden zwischen:

- den Eingangsparametern und
- den Durchgangsparametern

eines FB. In beiden Fällen weisen Sie über **Wertzuweisungen** den Formalparametern die Aktualparameter zu:

```
Formalparameter      Aktualparameter
    E_Par       ⇐     3        //Eingangszuweisung
    D_Par       ⇔     LAENGE   //Durchgangszuweisung
```

Bild 16-6 *Wertzuweisung in der Parameterliste*

Die Ausgangsparameter werden nicht beim Aufruf eines FB angegeben, sondern nach dem Aufruf versorgt.

Die Syntax der FB-Parameterangaben ist beim Aufruf globaler und lokaler Instanzen gleich.

FB-Parameter

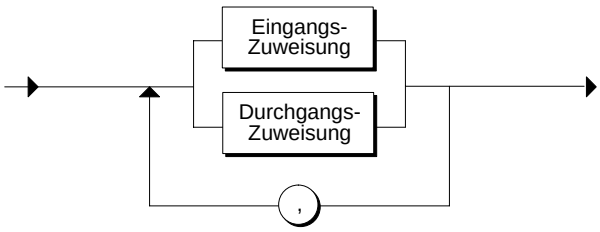


Bild 16-7 *Syntax: FB-Parameter*

Beispiel

Ein Aufruf mit je einer Zuweisung eines Eingangs- und eines Durchgangsparameters könnte z. B. wie folgt aussehen:

```
FB31.DB77(E_Par:=3, D_Par:=LAENGE);
```

Regeln

Zur Parameterversorgung gelten folgende Regeln:

- Die Reihenfolge der Zuweisungen ist beliebig.
- Datentyp von Formal- und Aktualparameter müssen übereinstimmen.
- Die einzelnen Zuweisungen sind durch Komma getrennt.
- Ausgangszuweisungen sind in FB-Aufrufen nicht möglich. Der Wert eines vereinbarten Ausgangsparameters ist in den Instanzdaten gespeichert. Dort ist er für den Zugriff von allen FBs aus verfügbar. Um ihn zu lesen, müssen Sie von einem FB aus einen Zugriff definieren (siehe Kapitel 14.8).

Ergebnis nach dem Aufruf

Nach dem Bausteindurchlauf

- sind die übergebenen aktuellen Eingangsparameter unverändert.
- sind die übergebenen aber veränderten Werte der Durchgangsparameter aktualisiert. Eine Ausnahme bilden die Durchgangsparameter eines elementaren Datentyps (siehe dazu Kapitel 16.2.3).
- können die Ausgangsparameter vom aufrufenden Baustein aus dem globalen Instanz-Datenbaustein oder dem lokalen Instanzbereich gelesen werden. Siehe dazu das Beispiel 16-3.

16.2.2 Eingangszuweisung (FB)

Prinzip

Durch Eingangszuweisungen werden den formalen Eingangsparametern Aktualparameter zugewiesen. Der FB kann diese Aktualparameter **nicht** verändern. Die Zuweisung von aktuellen Eingangsparametern ist optional. Wird kein Aktualparameter angegeben, dann bleiben die Werte des letzten Aufrufs erhalten.

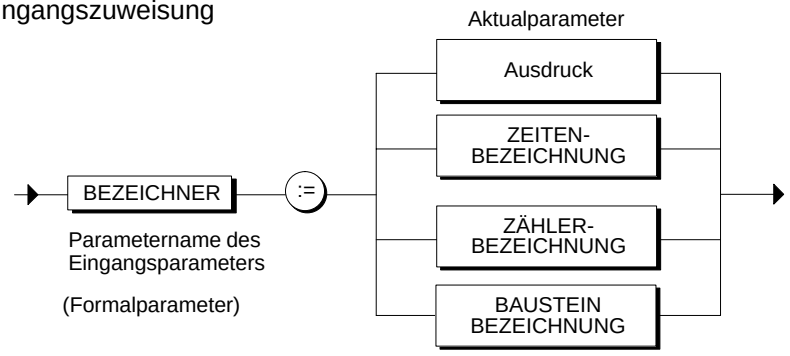


Bild 16-8 *Syntax: Eingangszuweisung*

Mögliche Aktualparameter

In Eingangszuweisungen sind folgende Aktualparameter möglich:

Tabelle 16-2 Aktualparameter in Eingangszuweisungen

Aktual-Parameter	Erläuterung
Ausdruck	• Arithmetischer, logischer oder Vergleichs-Ausdruck • Konstante • Erweiterte Variable
ZEITEN-/ZÄHLER-Bezeichnung	Sie legen eine bestimmte Zeit oder einen bestimmten Zähler fest, der bei der Bearbeitung eines Bausteins verwendet werden soll (siehe auch Kapitel 17).
BAUSTEIN-Bezeichnung	Sie legen einen bestimmten Baustein fest, der als Eingangsparameter verwendet werden soll. Die Bausteinart (FB, FC, DB) wird in der Deklaration des Eingangsparameters festgelegt. Bei der Parameterversorgung geben Sie die Bausteinnummer des Bausteins an. Dieses kann sowohl die absolute als auch die symbolische sein (siehe auch Kapitel 9).

16.2.3 Durchgangszuweisung (FB)

Prinzip

Durchgangszuweisungen dienen dazu, den formalen Durchgangsparametern des aufgerufenen FB Aktualparameter zuzuweisen.

Im Gegensatz zu Eingangsparametern kann der aufgerufene FB Durchgangsparameter jedoch verändern. Der neue Wert des Parameters, der bei der Abarbeitung des FBs entsteht, wird in den Aktualparameter zurückgeschrieben. Der ursprüngliche Wert wird dabei überschrieben.

Wenn im aufgerufenen FB Durchgangsparameter vereinbart sind, müssen diese beim ersten Aufruf versorgt werden. Die Angabe von Aktualparametern ist danach optional.

Durchgangszuweisung

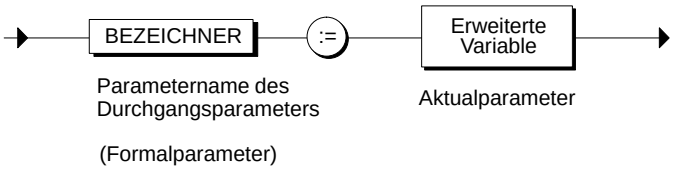


Bild 16-9 *Syntax: Durchgangszuweisung*

Aktualparameter einer Durchgangszuweisung

Da der zugewiesene Aktualparameter während des FB-Durchlaufs als Durchgangsparameter verändert werden kann, muß er eine Variable sein. Ein Eingangsparameter ist in Durchgangsanweisungen aus diesem Grund nicht zuweisbar (der neue Wert könnte nicht zurückgeschrieben werden).

Tabelle 16-3 Aktualparameter in Durchgangszuweisungen

Aktual-Parameter	Erläuterung
Erweiterte Variable	Folgende Typen der erweiterten Variablen stehen zur Verfügung: einfache Variablen und Parameter Zugriff auf Absolutvariablen Zugriff auf Datenbausteine Funktions-Aufrufe (siehe auch Kapitel 14).

Besonderheiten

Beachten Sie die folgenden Besonderheiten:

- Nach dem Bausteindurchlauf wird der veränderte Wert des Durchgangsparameters aktualisiert. Eine Ausnahme bilden die Durchgangsparameter eines **elementaren** Datentyps. Die Aktualisierung erfolgt hier nur, wenn beim Aufruf ein Aktualparameter angegeben wurde.
- Bei Durchgangsparametern von einem **nicht-elementaren** Datentyp sind als Aktualparameter nicht erlaubt:
  - FB-Durchgangsparameter
  - FC-Parameter
- **ANY-Parameter:** Grundsätzlich gelten hier auch die beiden ersten Aussagen. Zusätzlich gilt, daß Konstanten als Aktualparameter **nicht** zulässig sind.

16.2.4 Beispiel für den Aufruf einer globalen Instanz

Prinzip

Ein Funktionsbaustein mit einer FOR-Schleife könnte wie in Beispiel 16-1 aussehen. In den Beispielen wird angenommen, daß in der Symboltabelle für den FB17 das Symbol TEST vereinbart ist.

```
FUNCTION_BLOCK TEST
    VAR_INPUT
      ENDWERT:   INT; //Eingangsparameter
    END_VAR
    VAR_IN_OUT
      IQ1    :  REAL; //Durchgangsparameter
    END_VAR
    VAR_OUTPUT
      CONTROL:  BOOL; //Ausgangsparameter
    END_VAR
    VAR
      INDEX  :  INT;
    END_VAR
BEGIN
    CONTROL:= FALSE;
    FOR INDEX:= 1 TO ENDWERT DO
      IQ1:= IQ1 * 2;
      IF IQ1 > 10000 THEN
         CONTROL:= TRUE;
      END_IF;
    END_FOR;
END_FUNCTION_BLOCK
```

Beispiel 16-1 Beispiel eines FB

Aufruf

Zum Aufrufen dieses FB können Sie eine der folgenden Varianten wählen. Vorausgesetzt wird, daß VARIABLE1 im aufrufenden Baustein als REAL-Variable vereinbart ist

```
      //Absoluter Aufruf, globale Instanz:
FB17.DB10 (ENDWERT:=10, IQ1:= VARIABLE1);

      //Symbolischer Aufruf, globale Instanz:
TEST.TEST_1 (ENDWERT:= 10, IQ1:= VARIABLE1) ;
```

Beispiel 16-2 Beispiel FB-Aufruf mit Instanz-Datenbaustein

Ergebnis

Nach der Ausführung des Bausteins steht der für den Durchgangsparameter IQ1 ermittelte Wert in VARIABLE1 zur Verfügung.

Ausgangwert lesen

Die möglichen Varianten, den Ausgangsparameter CONTROL zu lesen, sollen anhand zweier Beispiele erläutert werden:

```
//Der Zugriff auf den Ausgangsparameter
//erfolgt durch:
    ERGEBNIS:= DB10.CONTROL;
//Sie können den Ausgangsparameter aber auch
//bei einem anderen FB-Aufruf direkt zur
//Versorgung eines Eingangsparameters heranziehen:
    FB17.DB12 (EIN_1:= DB10.CONTROL);
```

Beispiel 16-3 Ergebnis FB-Aufruf mit Instanz-Datenbaustein

16.2.5 Beispiel für den Aufruf einer lokalen Instanz

Prinzip

Ein Funktionsbaustein mit einer einfachen FOR-Schleife könnte wie in Beispiel 16-1 programmiert sein, wobei angenommen wird, daß in der Symboltabelle für den FB17 das Symbol TEST vereinbart ist.

Aufruf

Diesen FB können Sie unter der Voraussetzung daß VARIABLE1 im aufrufenden Baustein als REAL-Variable deklariert ist, so aufrufen:

```
// Aufruf, lokale Instanz:
TEST_L (ENDWERT:= 10, IQ1:= VARIABLE1) ;
```

Beispiel 16-4 Beispiel FB-Aufruf als lokale Instanz

Dabei muß TEST_L in der Variablenvereinbarung wie folgt deklariert sein:

```
VAR
  TEST_L : TEST;
END_VAR
```

Ausgangsparameter lesen

Der Ausgangsparameter CONTROL kann wie folgt gelesen werden:

```
// Der Zugriff auf den Ausgangsparameter erfolgt
// durch
ERGEBNIS:= TEST_L.CONTROL;
```

Beispiel 16-5 Ergebnis FB-Aufruf als lokale Instanz

16.3 Aufruf von Funktionen

Rückgabewert

Im Gegensatz zu den Funktionsbausteinen liefern Funktionen als Ergebnis den Rückgabewert. Aus diesem Grund können Funktionen wie Operanden behandelt werden. Eine Ausnahme bildet die Funktion mit dem Rückgabewert vom Typ VOID.

In der folgenden Wertzuweisung wird z. B. die Funktion ABSTAND mit bestimmten Parametern aufgerufen:

```
LAENGE:= ABSTAND (X1:=-3, Y1:=2);
Rückgabewert ist ABSTAND!
```

Die Funktion berechnet den Rückgabewert, der den gleichen Namen trägt wie die Funktion, und gibt ihn an den aufrufenden Baustein zurück. Dort ersetzt der Wert den Funktionsaufruf.

Der Rückgabewert kann in folgenden Elementen einer FC oder eines FB verwendet werden:

- in einer Wertzuweisung
- in einem logischen, arithmetischen oder Vergleichsausdruck
- als Parameter für einen weiteren Funktionsbaustein-/Funktions-Aufruf

Eine Ausnahme bilden die Funktionen vom Typ VOID. Sie haben keinen Rückgabewert und können somit nicht in Ausdrücken verwendet werden.

Bild 16-10 veranschaulicht die Syntax eines Funktionsaufrufs:

Funktionsaufruf

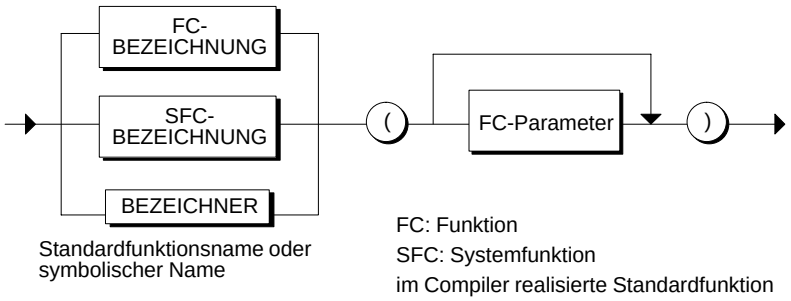


Bild 16-10 *Syntax: Funktionsaufrufs*

Hinweis

Wird in SCL eine Funktion aufgerufen, deren Rückgabewert nicht versorgt wurde, kann es zu einer fehlerhaften Ausführung des Anwenderprogramms kommen.

Bei einer SCL-Funktion kann dieser Fall eintreten, wenn der Rückgabewert zwar versorgt wurde, die entsprechende Anweisung aber nicht durchlaufen wird.
Bei einer AWL-/KOP-/FUP-Funktion kann dieser Fall eintreten, wenn die Funktion ohne Versorgng des Rückgabewertes programmiert wurde oder die entsprechende Anweisung nicht durchlaufen wird.

Aufruf

Der Aufruf einer Funktion erfolgt unter Angabe

- des Funktionsnamens (FC-, SFC-BEZEICHNUNG, BEZEICHNER)
- der Parameterliste.

Beispiel

Der Funktionsname, der den Rückgabewert bezeichnet, kann absolut oder symbolisch angegeben werden, z. B.:

```
FC31 (X1:=5, Q1:= Quersumme)
ABSTAND (X1:=5, Q1:= Quersumme)
```

Ergebnis des Aufrufs

Die Ergebnisse eines Funktionsaufrufs stehen nach dem Aufruf als

- Rückgabewert oder
- als Ausgangs- und Durchgangsparameter (Aktualparameter)

zur Verfügung. Weitere Informationen hierzu finden Sie in Kapitel 18.

16.3.1 FC-Parameter

Prinzip

Im Gegensatz zu Funktionsbausteinen haben Funktionen kein Gedächtnis, in dem sie die Werte der Parameter speichern könnten. Lokale Daten werden während des Durchlaufs der Funktion nur temporär gespeichert. Aus diesem Grund müssen beim Aufruf allen formalen Eingangs-, Durchgangs- und Ausgangsparametern, die im Vereinbarungsteil einer Funktion definiert werden, Aktualparameter zugewiesen werden.

Bild 16-11 zeigt die Syntax der FC-Parameterzuweisung:

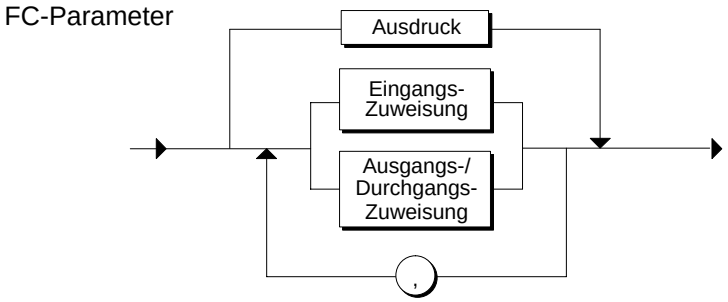


Bild 16-11 *Syntax: FC-Parameter*

Ein Aufruf mit je einer Zuweisung eines Ein-, Aus- und Durchgangsparameters könnte z. B wie folgt aussehen:

```
FC32 (E_Param1:=5,D_Param1:=LAENGE,
      A_Param1:=Quersumme)
```

Regeln

Zur Parameterversorgung gelten folgende Regeln:

- Die Reihenfolge der Zuweisungen ist beliebig.
- Datentyp von Formal- und Aktualparameter müssen übereinstimmen.
- Die einzelnen Zuweisungen sind durch Komma getrennt.

16.3.2 Eingangszuweisung (FC)

Prinzip

Durch Eingangszuweisungen werden den formalen Eingangsparametern, der aufgerufenen Funktion FC, Aktualparameter zugewiesen. Die FC kann mit diesen Aktualparametern arbeiten, sie jedoch nicht verändern. Im Gegensatz zum FB-Aufruf ist diese Zuweisung beim FC-Aufruf **nicht optional**. Eingangszuweisungen haben die folgende Syntax:

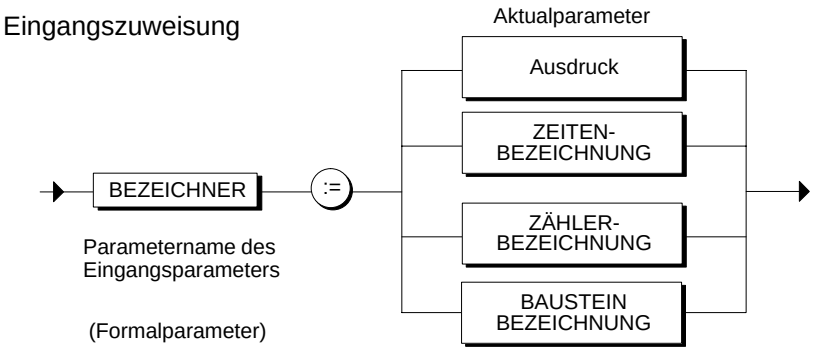


Bild 16-12 *Syntax: Eingangszuweisung*

Aktualparameter in Eingangszuweisungen

In Eingangszuweisungen sind folgende Aktualparameter zuweisbar:

Tabelle 16-4 Aktualparameter in Eingangszuweisungen

Aktualparameter	Erläuterung
Ausdruck	Ein Ausdruck steht für einen Wert und besteht aus Operanden und Operatoren. Folgende Arten von Ausdrücken stehen zur Verfügung: Arithmetischer, logischer oder Vergleichs-Ausdruck Konstante Erweiterte Variable
ZEITEN-/ZÄHLER-Bezeichnung	Sie legen eine bestimmte Zeit oder einen bestimmten Zähler fest, der bei der Bearbeitung eines Bausteins verwendet werden soll. Siehe auch Kapitel 17.
BAUSTEIN-Bezeichnung	Sie legen einen bestimmten Baustein fest, der als Eingangsparameter verwendet werden soll. Die Bausteinart (FB, FC, DB) wird in der Deklaration des Eingangsparameters festgelegt. Bei der Parameterzuweisung geben Sie die Bausteinadresse des Bausteins an. Diese kann sowohl die absolute als auch die symbolische sein. Siehe auch Kapitel 9.

Besonderheiten

Beachten Sie, daß bei formalen FC-Eingangsparametern mit nichtelementarem Datentyp FB-Durchgangsparameter und FC-Parameter als Aktualparameter nicht erlaubt sind.

16.3.3 Ausgangs-/Durchgangszuweisung (FC)

Prinzip

In einer Ausgangszuweisung legen Sie fest, wohin die Ausgangswerte, die bei der Abarbeitung einer Funktion entstehen, geschrieben werden. Mit einer Durchgangszuweisung weisen Sie einem Durchgangsparameter einen Aktualwert zu.

Bild 16-13 zeigt die Syntax von Ausgangs- und Durchgangszuweisungen:

Ausgangs-/Durchgangszuweisung

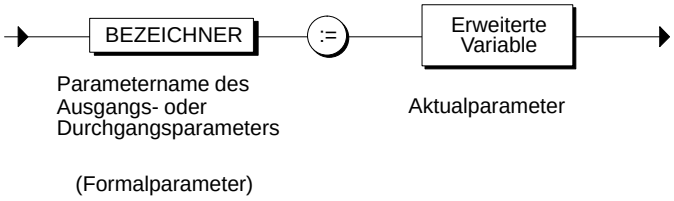


Bild 16-13 *Syntax: Ausgangs-/Durchgangszuweisung*

Aktualparameter in Ausgangs-/ Durchgangs-zuweisungen

Die Aktualparameter in Ausgangs- und Durchgangszuweisungen müssen eine Variable sein, da die Funktion Werte in die Parameter schreiben soll. Ein Eingangsparameter ist in Durchgangsanweisungen aus diesem Grund nicht zuweisbar (der Wert könnte nicht geschrieben werden).In Ausgangs- und Durchgangszuweisungen ist also nur die erweiterte Variable zuweisbar:

Tabelle 16-5 Aktualparameter in Ausgangs-/Durchganszuweisungen

Aktualparameter	Erläuterung
Erweiterte Variable	Folgende Typen der erweiterten Variablen stehen zur Verfügung: einfache Variablen und Parameter Zugriff auf Absolutvariablen Zugriff auf Datenbausteine Funktions-Aufrufe (siehe auch Kapitel 14)

Besonderheiten

Beachten Sie die folgenden Besonderheiten:

- Nach dem Bausteindurchlauf wird der veränderte Wert des Durchgangsparameters aktualisiert.
- Bei Durchgangsparametern von einem **nichtelementaren** Datentyp sind als Aktualparameter nicht erlaubt:
  - FB-Eingangsparameter
  - FB-Durchgangsparameter oder
  - FC-Parameter
- **ANY-Parameter:** Grundsätzlich gilt hier auch die erste Aussage. Bei Durchgangsparametern von einem **nichtelementaren** Datentyp sind als Aktualparameter nicht erlaubt:
  - FB-Eingangsparameter
  - FC-Eingangsparameter

  Zusätzlich gilt, daß Konstanten als Aktualparameter **nicht** zulässig sind. Wenn als Funktionsergebnis (Rückgabewert) der Typ ANY vereinbart ist, gilt außerdem:
  - Alle ANY-Parameter müssen mit Operanden versorgt werden, deren Datentyp innerhalb einer Typklasse liegt. Als Typklasse wird dabei z. B. die Menge der numerischen Datentypen (INT, DINT, REAL) oder die Menge der Bit-Datentypen (BOOL, BYTE, WORD, DWORD) verstanden. Die anderen Datentypen bilden jeweils eine eigene Typklasse.
  - Der SCL-Compiler geht davon aus, daß sich der Datentyp des aktuellen Funktionsergebnisses als der mächtigste Typ unter den Aktualparameter, die den ANY-Parametern zugewiesen sind, bestimmen läßt.
    Mit dem Funktionsergebnis sind dann alle Operationen erlaubt, die für diesen Datentyp definiert sind.
- **POINTER-Parameter:** Grundsätzlich gilt hier auch die erste Aussage. Bei Durchgangsparametern von einem **nichtelementaren** Datentyp sind als Aktualparameter nicht erlaubt:
  - FB-Eingangsparameter
  - FC-Eingangsparameter

16.3.4 Beispiel für einen Funktionsaufruf

Prinzip

Eine Funktion ABSTAND zur Berechnung des Abstandes zweier Punkte (X1,Y1) und (X2,Y2) in der Ebene bei Verwendung kartesischer Koordinaten könnte folgendermaßen aussehen (In den Beispielen ist immer angenommen, daß in einer Symboltabelle für den FC37 das Symbol ABSTAND vereinbart ist.):

```
FUNCTION ABSTAND: REAL
      VAR_INPUT
            X1: REAL;
            X2: REAL;
            Y1: REAL;
            Y2: REAL;
      END_VAR
      VAR_OUTPUT
            Q2: REAL;
      END_VAR
      BEGIN
            ABSTAND:= SQRT
            ( (X2-X1)**2 + (Y2-Y1)**2 );
            Q2:= X1+X2+Y1+Y2;
END_FUNCTION
```

Beispiel 16-6 Abstandsberechnung

Zur weiteren Verwendung eines Funktionswertes stehen Ihnen unter anderem folgende Möglichkeiten offen:

in einer Wertzuweisung, z. B.:

```
LAENGE:= ABSTAND (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Quersumme);
```

in einem arithmetischen oder logischen Ausdruck, z. B.:

```
RADIUS + ABSTAND (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Quersumme)
```

in der Parameterversorgung eines Bausteines, der aufgerufen wird, z. B.:

```
FB32 (DISTANZ:= ABSTAND (X1:=-3, Y1:=2, X2:=8.9,
Y2:=7.4, Q2:=Quersumme);
```

Beispiel 16-7 Werteberechnung innerhalb einer FC

16.4 Implizit definierte Parameter

Übersicht

Implizit definierte Parameter sind solche, die Sie verwenden können, ohne sie zuvor in einem Baustein zu vereinbaren. SCL stellt zwei so definierte Parameter zur Verfügung:

- den Eingangsparameter `EN` und
- den Ausgangsparameter `ENO`

Beide Parameter sind vom Datentyp BOOL und sind `im` Bereich bausteintemporäre Daten abgelegt.

Eingangsparameter EN

Jeder Funktionsbaustein und jede Funktion besitzt den implizit definierten Eingangsparameter EN. Wenn EN gleich TRUE ist, wird der aufgerufene Baustein ausgeführt, andernfalls nicht. Die Versorgung des Parameters EN ist optional.

Beachten Sie, daß EN nicht im Vereinbarungsteil eines Bausteines bzw. einer Funktion deklariert werden darf.

Da EN ein Eingangsparameter ist, können Sie EN innerhalb eines Bausteines nicht verändern.

Hinweis

Der Rückgabewert einer Funktion ist nicht definiert, falls die Funktion wegen EN:=FALSE nicht aufgerufen wurde.

Beispiel

Folgendes Beispiel veranschaulicht den Gebrauch des Parameters EN:

```
FUNCTION_BLOCK FB57
VAR
      ERGEBNIS    : REAL;
      MEIN_ENABLE : BOOL;
END_VAR
//...
BEGIN
MEIN_ENABLE:= FALSE;
// Aufruf einer Funktion,
// wobei der EN - Parameter versorgt wird:

ERGEBNIS:= FC85 (EN:= MEIN_ENABLE, PAR_1:= 27);
// FC85 wurde nicht ausgeführt, da MEIN_ENABLE
// gleich FALSE gesetzt wurde
//...
END_FUNCTION_BLOCK
```

Beispiel 16-8 Verwendung von EN

Ausgangs-parameter ENO

Jeder Funktionsbaustein und jede Funktion besitzt den implizit definierten Ausgangsparameter ENO, der vom Datentyp BOOL ist. Am Ende der Ausführung eines Bausteines wird der gerade aktuelle Wert des OK-Flags in ENO abgelegt.

Unmittelbar nach dem Aufruf eines Bausteines können Sie anhand des Wertes von ENO überprüfen, ob alle Operationen im Baustein richtig abgelaufen sind oder ob es zu Fehlern gekommen ist.

Beispiel

Folgendes Beispiel veranschaulicht den Gebrauch des Parameters EN0.

```
FUNCTION_BLOCK FB57
      //...
      //...
BEGIN
// Aufruf eines Funktionsbausteines:
FB30.DB30 (X1:=10, X2:=10.5);

// Überprüfung, ob im aufgerufenen Baustein
// alles in Ordnung abgelaufen ist:

IF ENO THEN
      // alles in Ordnung
      //...
ELSE
      // Fehler aufgetreten,
      // daher Fehlerbehandlung
      //...
END_IF;
      //...
      //...
END_FUNCTION_BLOCK
```

Beispiel 16-9 Verwendung von ENO

Beispiel

Folgendes Beispiel zeigt eine Verkettung von EN und EN0.

```
// EN und ENO können auch verkettet verwendet
// werden:

FB30.DB30(X1:=10, X2:=10.5);

// Die folgende Funktion soll nur
// ausgeführt werden,wenn der FB 30 fehlerfrei
// abgelaufen ist:

ERGEBNIS:= FC 85 (EN:= ENO, PAR_1:= 27);
```

Beispiel 16-10 Verwendung von EN und ENO

Zähler und Zeiten

17

Übersicht

In SCL können Sie den Programmablauf abhängig von einer Zeitangabe oder einem Zählerstand steuern.

STEP 7 stellt dazu standardmäßige Zähler- und Zeitfunktionen bereit, die Sie in Ihrem SCL Programm verwenden können, ohne sie zuvor vereinbaren zu müssen.

Kapitelübersicht

17.1 Zählfunktionen

Überblick

STEP 7 stellt eine Reihe von Standard-Zählfunktionen zur Verfügung. Diese Zähler können Sie in Ihrem SCL-Programm verwenden, ohne sie zuvor vereinbaren zu müssen. Sie müssen sie lediglich mit den erforderlichen Parametern versorgen. STEP 7 bietet folgende Zählfunktionen an:

- Aufwärtszähler (Counter Up)
- Abwärtszähler (Counter Down)
- Auf- und Abwärtszähler (Counter Up Down)

Aufruf

Zählfunktionen werden aufgerufen wie Funktionen. Die Funktionsbezeichnung kann überall anstelle eines Operanden in einem Ausdruck eingesetzt werden, solange der Typ des Funktionsergebnisses mit dem des ersetzten Operanden kompatibel ist.

Tabelle 17-1 Funktionsnamen der Zählfunktionen

Funktionsname	Bedeutung
S_CU	Aufwärtszähler (Counter Up)
S_CD	Abwärtszähler (Counter Down)
S_CUD	Auf- und Abwärtszähler (Counter Up Down)

Funktionwert

Der Funktionwert (Rückgabewert), der an die Aufrufstelle zurückgegeben wird, ist der aktuelle Zählwert (BCD-Format) im Datentyp WORD. Informationen hierzu finden Sie im Kapitel 17.1.1.

Aufrufparameter

Die Aufrufparameter mit ihrer Bezeichnung und Bedeutung für alle 3 Zählfunktionen finden Sie in Tabelle 17-2. Generell sind folgende Arten von Parametern zu unterscheiden:

- Steuerparameter (z. B. Setzen, Rücksetzen, Zählrichtung angeben)
- Vorbesetzungswert für einen Zählerstand
- Statusausgang (zeigt ob ein End-Zählerstand erreicht ist)
- Zählerstand in binärer Form.

Tabelle 17-2 Aufrufparameter

Bezeichner	Parameter	Datentyp	Beschreibung
C_NO		COUNTER	Zählerkennummer (ZAEHLER-BEZEICHNUNG); Bereich ist von der CPU abhängig.
CU	Eingang	BOOL	Eingang CU: Vorwärtszählen
CD	Eingang	BOOL	Eingang CD: Rückwärtszählen
S	Eingang	BOOL	Eingang für Voreinstellung des Zählers
PV	Eingang	WORD	Wert im Bereich zwischen 0 und 999 für das Setzen des Zählers (eingegeben als 16#<Wert>, wobei Wert im BCD-Format)
R	Eingang	BOOL	Rücksetzeingang
Q	Ausgang	BOOL	Status des Zählers
CV	Ausgang	WORD	Zählerstand (binär)

Beispiel

Der im Beispiel 17-1 genannte Aufruf bewirkt, daß bei der Funktionsberechnung ein globaler Speicherbereich vom Typ COUNTER mit Namen Z12 reserviert wird.

```
Zählwert:= S_CUD (C_NO :=Z12,
                  CU    :=E0.1,
                  CD    :=E0.0,
                  S     :=E0.2 & E0.3,
                  PV    :=120,
                  R     :=FALSE,
                  Q     :=actFlag ,
                  CV    :=binVal);
```

Beispiel 17-1 Aufruf einer Zählfunktion abwärts

Dynamischer Aufruf

Anstelle der absoluten Zähler-Nummer (z. B. C_NO:=Z10), können Sie auch eine Variable mit dem Datentyp COUNTER beim Aufruf angeben. Das hat den Vorteil, daß Sie den Zähleraufruf dynamisch gestalten können, indem Sie dieser Variablen bei jedem Aufruf eine andere absolute Nummer zuweisen.

```
Beispiel:
FUNCTION_BLOCK ZAEHLER;
VAR_INPUT
 MeinZaehler: Counter;
END_VAR
:
currVAL:=S_CD (C_NO:=MeinZaehler,........);
```

Regeln

Da die Parameterwerte (z. B. CD:=E0.0) global gespeichert sind, ist ihre Angabe in bestimmten Fällen optional. Bei der Parameterversorgung sind folgende allgemeine Regeln zu beachten:

- Der Parameter für die Zählerbezeichnung C_NO muß beim Aufruf immer versorgt werden.
- Je nach Zählerfunktion muß entweder der Parameter CU (Aufwärtszähler) oder der Parameter CD (Abwärtszähler) versorgt werden.
- Die Angabe der Parameter PV (Vorbesetzwert) und S (Setzen) kann paarweise entfallen.
- Der Ergebniswert im BCD-Format ist immer der Funktionswert.

Hinweis

Die Namen der Funktionen und Parameter sind in SIMATIC- und IEC-Mnemonik gleich. Nur die Zählerbezeichnung ist von der Mnemonik abhängig: *SIMATIC:* Z und *IEC:* C

Beispiel Aufruf von Zählfunktionen

Beispiel 17-2 veranschaulicht den Aufruf der Zählfunktionen:

```
FUNCTION_BLOCK FB1


VAR
 currVal, binVal: word;
 actFlag: bool;
END_VAR
```

```
BEGIN
currVal  :=S_CD(C_NO:=Z10, CD:=TRUE, S:=TRUE,
               PV:=100, R:=FALSE, CV:=binVal,
               Q:=actFlag);
```

```
currVal  :=S_CU(C_NO:=Z11, CU:=M0.0, S:=M0.1,
               PV:=16#110, R:=M0.2, CV:=binVal,
               Q:=actFlag);
```

```
currVal  :=S_CUD(C_NO:=Z12, CD:=E0.0,
               CU:=E0.1,S:=E0.2 & E0.3, PV:=120,
               R:=FALSE,CV:=binVal, Q:=actFlag);
```

```
currVal  :=S_CD(C_NO:=Z10,CD:=FALSE,
               S:=FALSE,
               PV:=100, R:=TRUE, CV:=binVal,
               Q:=actFlag);


END_FUNCTION_BLOCK
```

Beispiel 17-2 Aufruf von Zählfunktionen

17.1.1 Eingabe und Auswertung des Zählerwerts

Übersicht

Für die Eingabe des Vorbesetzungswertes bzw. für die Auswertung des Funktionsergebnisses benötigen Sie die interne Darstellung des Zählerwerts (siehe Bild 17-1).

Wenn der Zähler gesetzt wird (Parameter S), wird der von Ihnen festgelegte Wert in den Zähler geschrieben. Der Wertebereich liegt zwischen 0 und 999. Sie können den Zählerwert innerhalb dieses Bereichs verändern, indem Sie die Operationen Vorwärts-/Rückwärtszählen, Vorwärtszählen und Rückwärtszählen verwenden.

Format

Bild 17-1 veranschaulicht die Bit-Konfiguration des Zählerwertes:

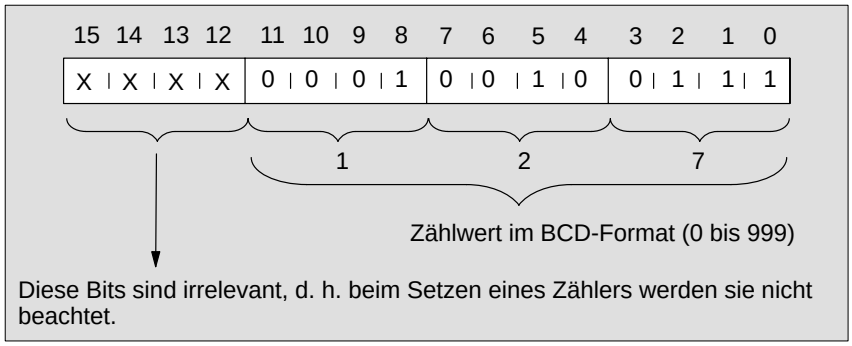


Bild 17-1 *Bit-Konfiguration für einen Zählerwert*

Eingabe

Mit folgenden Formaten können Sie einen vordefinierten Zählerwert laden:

- Dezimal als Integer-Wert: z. B. 295, sofern dieser Wert einer gültigen BCD-Codierung entspricht.
- Im BCD-Format (Eingabe als Hexadezimalkonstante): z. B. 16#127

Auswertung

Das Ergebnis können Sie in zwei verschiedenen Formaten auswerten:

- Als Funktionsergebnis (Typ WORD): im BCD-Format
- Als Ausgangsparameter CV (Typ WORD): binär

17.1.2 Aufwärtszählen (Counter Up)

Beschreibung

Mit dem Zähler Counter Up können Sie nur Aufwärts-Zähloperationen durchführen.

Funktionsweise

Tabelle 17-3 Funktionsweise Aufwärtszähler

Operation	Funktionsweise
Vorwärts-zählen	Der Wert des Zählers wird um ”1” weitergestellt, wenn der Signalzustand an Eingang **CU** von ”0” auf ”1” wechselt und der Zählwert kleiner als 999 ist.
Zähler setzen	Wenn der Signalzustand an Eingang **S** von ”0” auf ”1” wechselt, wird der Zähler mit dem Wert des Eingangs **PV** gesetzt. Ein solcher Signalwechsel ist immer erforderlich, um einen Zähler zu setzen.
Rücksetzen	Der Zähler wird rückgesetzt, wenn Eingang **R** = 1 ist. Das Rücksetzen des Zählers setzt den Zählwert auf ”0”.
Zähler abfragen	Eine Signalzustandsabfrage an Ausgang **Q** ergibt ”1”, wenn der Zählwert größer als ”0” ist. Die Abfrage ergibt ”0”, wenn der Zählwert gleich ”0” ist.

17.1.3 Abwärtszählen (Counter Down)

Beschreibung

Mit dem Zähler Counter Down können Sie nur Abwärts-Zähloperationen durchführen.

Funktionsweise

Tabelle 17-4 Funktionsweise Abwärtszähler

Operation	Funktionsweise
Abwärts-zählen	Der Wert des Zählers wird um ”1” vermindert, wenn der Signalzustand an Eingang **CD** von ”0” auf ”1” wechselt und der Zählwert größer als ”0” ist.
Zähler setzen	Wenn der Signalzustand an Eingang **S** von ”0” auf ”1” wechselt, wird der Zähler mit dem Wert des Eingangs **PV** gesetzt. Ein solcher Signalwechsel ist immer erforderlich, um einen Zähler zu setzen.
Rücksetzen	Der Zähler wird rückgesetzt, wenn Eingang **R** = 1 ist. Das Rücksetzen des Zählers setzt den Zählwert auf ”0”.
Zähler abfragen	Eine Signalzustandsabfrage an Ausgang **Q** ergibt ”1”, wenn der Zählwert größer als ”0” ist. Die Abfrage ergibt ”0”, wenn der Zählwert gleich ”0” ist.

17.1.4 Auf- / Abwärtszählen (Counter Up Down)

Beschreibung

Mit dem Zähler Counter Up Down können Sie sowohl Auf- als auch Abwärts-Zähloperationen ausführen. Bei Gleichzeitigkeit von Vorwärts- und Rückwärts-Zählimpulsen werden beide Operationen bearbeitet.
Der Zählwert bleibt unverändert.

Funktionsweise

Tabelle 17-5 Funktionsweise Auf-/Abwärtszähler

Operation	Funktionsweise
Aufwärts-zählen	Der Wert des Zählers wird um ”1” erhöht, wenn der Signalzustand an Eingang **CU** von ”0” auf ”1” wechselt und der Zählwert kleiner als 999 ist.
Abwärts-zählen	Der Wert des Zählers wird um ”1” vermindert, wenn der Signalzustand an Eingang **CD** von ”0” auf ”1” wechselt und der Zählwert größer als ”0” ist.
Zähler setzen	Wenn der Signalzustand an Eingang **S** von ”0” auf ”1” wechselt, wird der Zähler mit dem Wert des Eingangs **PV** gesetzt. Ein solcher Signal-wechsel ist immer erforderlich, um einen Zähler zu setzen.
Rücksetzen	Der Zähler wird rückgesetzt, wenn Eingang **R** = 1 ist. Das Rücksetzen des Zählers setzt den Zählwert auf ”0”.
Zähler abfragen	Eine Signalzustandsabfrage an Ausgang **Q** ergibt ”1”, wenn der Zähl-wert größer als ”0” ist. Die Abfrage ergibt ”0”, wenn der Zählwert gleich ”0” ist.

17.1.5 Beispiel für die Funktion S_CD (Abwärtszähler)

Parameter-belegung

Tabelle 17-6 zeigt die Parameterbelegung der Beispielfunktion S_CD.

Tabelle 17-6 Aufrufparameter

Parameter	Beschreibung
C_NO	`MEINZAEHLER`
CD	Eingang E0.0
S	`SETZEN`
PV	`VORBESETZUNG 16#0089`
R	`RUECKSETZEN`
Q	`A0.7`
CV	`BIN_WERT`

Beispiel

Beispiel 17-3 zeigt ein Beispiel der Zählfunktion S_CD:

```
FUNCTION_BLOCK ZAEHLEN
VAR_INPUT
 MEINZAEHLER: COUNTER;
END_VAR
VAR_OUTPUT
 ERGEBNIS: INT;
END_VAR
VAR
 SETZEN           : BOOL;
 RUECKSETZEN      : BOOL;
 BCDWERT          : WORD; //Zaehlerstand BCD
 BINWERT          : WORD; //Zaehlerstand binär
 VORBESETZUNG     : WORD;
END_VAR
BEGIN
 A0.0:= 1;
 SETZEN:= E0.2;
 RUECKSETZEN:= E0.3;
 VORBESETZUNG:= 16#0089;
 BCDWERT:= S_CD
           (C_NO := MEINZAEHLER,//abwärtsz.
           CD    := E0.0,
           S     := SETZEN,
           PV    := VORBESETZUNG,
           R     := RUECKSETZEN,
           CV    := BINWERT,
           Q     := A0.7);
ERGEBNIS:=WORD_TO_INT(BINWERT);//Weiterverarb.
                 //als Ausgangsparameter
AW4:= BCDWERT;    //An Ausgabe zur Anzeige
END_FUNCTION_BLOCK
```

Beispiel 17-3 Beispiel für eine Zählfunktion

17.2 Zeitfunktionen (TIMER)

Überblick

Zeiten sind Funktionselemente in Ihrem Programm, die zeitgesteuerte Abläufe ausführen und überwachen. STEP 7 stellt eine Reihe von Standard-Zeitfunktionen zur Verfügung, auf die Sie mit SCL zugreifen können. Mit Zeitoperationen können Sie in Ihrem Programm:

- Wartezeiten einstellen
- Überwachungszeiten ermöglichen
- Impulse erzeugen
- Zeiten messen

Aufruf

Die Zeitfunktionen werden in der gleichen Weise wie die Zählfunktionen aufgerufen. Die Funktionsbezeichnung kann überall anstelle eines Operanden in einem Ausdruck eingesetzt werden, solange der Typ des Funktionsergebnisses mit dem des ersetzten Operanden kompatibel ist.

Tabelle 17-7 STEP 7-Zeitfunktionen

Funktionsname	Bedeutung
S_PULSE	Impuls (Pulse)
S_PEXT	Verlängerter Impuls (Pulse Extended)
S_ODT	Einschaltverzögerung (On Delay Time)
S_ODTS	Speichernde Einschaltverzögerung (Stored On Delay Time)
S_OFFDT	Ausschaltverzögerung (Off Delay Time)

Funktionwert

Der Funktionwert (Rückgabewerte) der an die Aufrufstelle zurückgegeben wird, ist ein Zeitwert vom Datentyp S5TIME. Informationen finden Sie in Kapitel 17.2.1

Aufrufparameter

Die Parameter, die versorgt werden müssen, sind bei der Beschreibung der jeweiligen Standardfunktion tabellarisch erläutert. Die Namen mit den zugehörigen Datentypen für alle 5 Zeitfunktionen finden Sie in Tabelle 17-8.

Generell sind folgende Arten von Parametern zu unterscheiden:

- Steuerparameter (z. B. Setzen, Rücksetzen)
- Vorbesetzungswert für die Startzeit
- Statusausgang (zeigt ob die Zeit noch läuft)
- Rest-Zeitwert in binärer Form

Tabelle 17-8 Aufrufparameter

Parameter	Datentyp	Beschreibung
T_NO	TIMER	Kennummer der Zeit; Bereich ist von der CPU abhängig.
S	BOOL	Starteingang
TV	S5TIME	Voreinstellung Zeitwert (BCD-Format)
R	BOOL	Rücksetzeingang
Q	BOOL	Status der Zeit
BI	WORD	Rest-Zeitwert (binär)

Beispiel

Der im Beispiel 17-4 genannte Aufruf bewirkt bei seiner Abarbeitung, daß ein globaler Speicherbereich vom Typ TIMER mit dem Namen T10 reserviert wird.

```
VERZÖGERUNG:= S_ODT (T_NO :=    T10,
                     S    :=    TRUE,
                     TV   :=    T#1s,
                     R    :=    FALSE,
                     BI   :=    biVal,
                     Q    :=    actFlag
                     );
```

Beispiel 17-4 Aufruf einer Zählfunktion abwärts

Dynamischer Aufruf

Anstelle der absoluten Zeitglied-Nummer (z. B. T10), können Sie auch eine Variable mit dem Datentyp TIMER beim Aufruf angeben. Das hat den Vorteil, daß Sie den Zeitgliedaufruf dynamisch gestalten können, indem Sie dieser Variablen bei jedem Aufruf eine andere absolute Nummer zuweisen.

```
Beispiel:
FUNCTION_BLOCK ZEITGEBER
VAR_INPUT
meineZeit: timer;
END_VAR
:
currTime:=S_ODT (T_NO:=meineZeit,.........)
```

Regeln

Da die Parameterwerte global gespeichert sind, ist ihre Angabe in bestimmten Fällen optional. Die folgenden allgemeinen Regeln sind bei der Parameterversorgung zu beachten:

- Der Parameter für die Zeitgliedbezeichnung T_NO muß beim Aufruf in symbolischer oder absoluter Form versorgt werden.
- Die Angabe der Parameter TV (Vorbesetzwert) und S (Setzen) kann paarweise entfallen.
- Die Parameterentsorgung ist optional. Sie können auf Q und BI mittels einer Wertzuweisung zugreifen.
- Der Ergebniswert im S5TIME-Format ist immer der Funktionswert.

Hinweis

Die Namen der Funktionen sind in SIMATIC- und IEC-Mnemonik gleich.

Beispiel Aufruf von Zeitfunktionen

Beispiel 17-5 veranschaulicht den Aufruf der Zeitfunktionen:

```
FUNCTION_BLOCK FB2

VAR
      currTime    : S5time;
      biVal       : word;
      actFlag     : bool;
END VAR

BEGIN
currTime:= S_ODT (T_NO:=T10, S:=TRUE, TV:=T#1s,
                  R:=FALSE, BI:=biVal,
                  Q:=actFlag);
```

```
currTime:= S_ODTS (T_NO:=T11, S:=M0.0, TV:=T#1s,
                  R:=M0.1, BI:=biVal,
                  Q:=actFlag);
```

```
currTime:=S_OFFDT (T_NO:=T12, S:=E0.1 & actFlag,
                  TV:=T#1s, R:=FALSE, BI:=biVal,
                  Q:=actFlag);
```

```
currTime:= S_PEXT (T_NO:=T13, S:=TRUE,
                  TV:=T#1s, R:=E0.0, BI:=biVal,
                  Q:=actFlag);
```

```
currTime:= S_PULSE (T_NO:=T14, S:=TRUE,
                  TV:=T#1s, R:=FALSE, BI:=biVal,
                  Q:=actFlag);
END_FUNCTION_BLOCK
```

Beispiel 17-5 Aufruf von Zeitfunktionen

17.2.1 Eingabe und Auswertung des Zeitwerts

Überblick

Für die Eingabe des Vorbesetzungswertes bzw. für die Auswertung des Funktionsergebnisses im BCD-Code benötigen Sie die interne Darstellung des Zeitwerts (siehe Bild 17-2).

Das Aktualisieren der Zeit vermindert den Zeitwert um jeweils eine Einheit in einem Intervall, der von der Zeitbasis festgelegt wurde. Der Zeitwert wird solange vermindert, bis er gleich ”0” ist. Der Zeitbereich umfaßt 0 bis 9 990 Sekunden.

Format

Bild 17-2 zeigt die interne Darstellung des Zeitwerts.

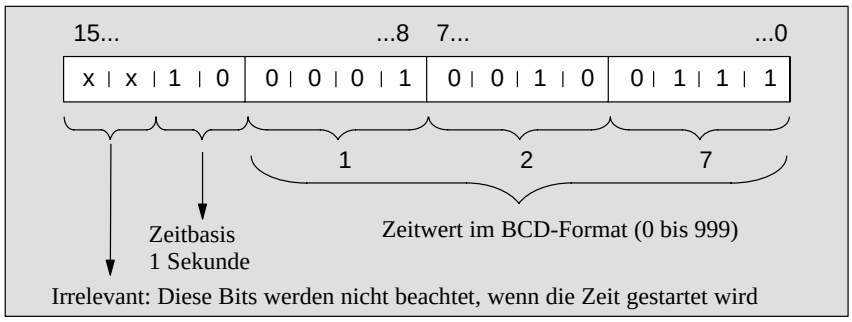


Bild 17-2 *Darstellung des Zeitwerts*

Eingabe

Mit folgenden Darstellungen können Sie einen vordefinierten Zeitwert laden:

- In Stufendarstellung: `TIME#aH_bbM_ccS_dddMS`
- In Dezimaldarstellung: `TIME#2.4H`

Auswertung

Das Ergebnis können Sie in zwei verschiedenen Formaten auswerten:

- Als Funktionsergebnis (Typ S5TIME): im BCD-Format
- Als Ausgangsparameter (Zeitwert ohne Zeitbasis vom Typ WORD): binär.

Zeitbasis

Die Bits 12 und 13 des Timerworts enthalten die Zeitbasis binär-codiert. Die Zeitbasis definiert das Intervall, in dem der Zeitwert um eine Einheit vermindert wird (siehe Tabelle 17-9 und Bild 17-2). Die kleinste Zeitbasis beträgt 10 ms; die größte 10 s.

Tabelle 17-9 Zeitbasis und Binärcode

Zeitbasis	Binärcode für Zeitbasis
10 ms	00
100 ms	01
1 s	10
10 s	11

Hinweis

Da Zeitwerte nur in einem Zeitintervall gespeichert werden, werden Werte, die keine genauen Vielfache des Zeitintervalls sind, abgeschnitten.

Werte, deren Auflösung für den gewünschten Bereich zu groß ist, werden abgerundet, so daß der gewünschte Bereich erzielt wird, **nicht** jedoch die gewünschte Auflösung.

17.2.2 Zeit als Impuls starten

Beschreibung

Die maximale Zeit, in der das Ausgangssignal auf ”1” bleibt, ist gleich dem programmierten Zeitwert t.

Tritt während der Laufzeit des Zeitgliedes am Eingang der Signalzustand 0 auf, wird der Ausgang Q auf ”0” gesetzt (d.h. die Zeit wird angehalten). Dies bedeutet eine vorzeitige Beendigung der Laufzeit.

Bild 17-3 veranschaulicht die Funktionsweise des Zeitglieds ”Zeit als Impuls starten”:

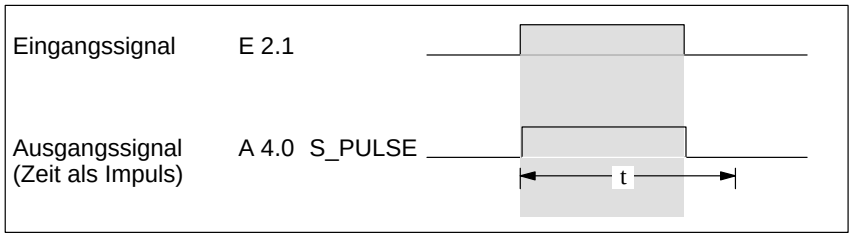


Bild 17-3 *Zeitglied ”Zeit als Impuls starten”*

Funktionsweise

Tabelle 17-10 Funktionsweise ”Zeit als Impuls starten”

Operation	Funktionsweise
Zeit starten	Die Operation ”Zeit als Impuls starten” startet eine angegebene Zeit, wenn der Signalzustand am Starteingang **S** von ”0” auf ”1” wechselt. Um die Zeit freizugeben, ist immer ein Signalwechsel erforderlich.
Laufzeit festlegen	Die Zeit läuft solange mit dem Wert weiter, der an Eingang **TV** angegeben ist, bis die programmierte Zeit abgelaufen ist und der Eingang S = 1 ist.
Laufzeit vorzeitig beenden	Wechselt Eingang S von ”1” auf ”0”, bevor der Zeitwert abgelaufen ist, wird die Zeit angehalten.
Rücksetzen	Die Zeit wird zurückgesetzt, wenn der Rücksetzeingang **R** von ”0” auf ”1” wechselt, während die Zeit läuft. Durch diesen Wechsel werden auch der Zeitwert und die Zeitbasis auf Null zurückgesetzt. Der Signalzustand ”1” an Eingang R hat keinen Einfluß, wenn die Zeit nicht läuft.
Signalzustand abfragen	Solange die Zeit läuft, ergibt eine Signalzustandsabfrage nach ”1” an Ausgang **Q** das Ergebnis ”1”. Bei vorzeitiger Beendigung der Laufzeit ergibt eine Signalzustandsabfrage an Ausgang Q das Ergebnis ”0”.
Aktuellen Zeitwert abfragen	Der aktuelle Zeitwert kann am Ausgang **BI** und durch den Funktionswert S_PULSE abgefragt werden.

17.2.3 Zeit als verlängerten Impuls starten

Beschreibung

Das Ausgangssignal bleibt für die programmierte Zeit (t) auf ”1”, unabhängig davon, wie lange das Eingangssignal auf ”1” bleibt. Eine erneute Auslösung des Startimpulses bewirkt einen erneuten Ablauf der Zeitdauer, so daß der Ausgangsimpuls zeitlich verlängert wird (Nachtriggerung).

Bild 17-4 veranschaulicht die Funktionsweise des Zeitgliedes ”Zeit als verlängerten Impuls starten”.

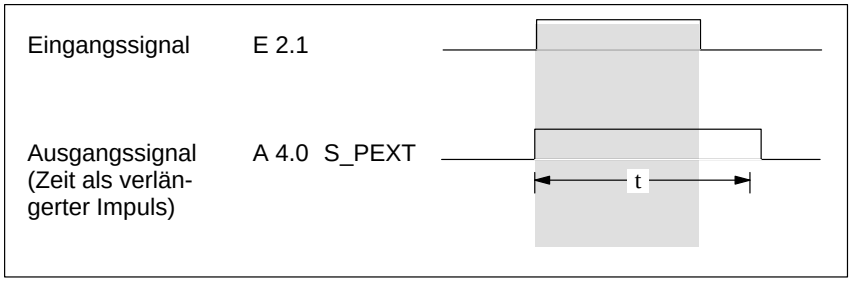


Bild 17-4 *Zeitglied ”Zeit als verlängerten Impuls starten”*

Funktionsweise

Tabelle 17-11 Funktionsweise ”Zeit als verlängerten Impuls starten”

Operation	Funktionsweise
Zeit starten	Die Operation ”Zeit als verlängerten Impuls starten” startet eine angegebene Zeit, wenn der Signalzustand am Starteingang **S** von ”0” auf ”1” wechselt. Um die Zeit freizugeben, ist immer ein Signalwechsel erforderlich.
Laufzeit erneut starten	Wechselt der Signalzustand am Eingang **S** noch während der Laufzeit erneut auf ”1”, wird die Zeit mit dem angegebenen Zeitwert neu gestartet
Laufzeit voreinstellen	Die Zeit läuft solange mit dem Wert weiter, der an Eingang **TV** angegeben ist, bis die programmierte Zeit abgelaufen ist
Rücksetzen	Die Zeit wird zurückgesetzt, wenn der Rücksetzeingang **R** von ”0” auf ”1” wechselt, während die Zeit läuft. Durch diesen Wechsel werden auch der Zeitwert und die Zeitbasis auf Null zurückgesetzt. Der Signalzustand ”1” an Eingang R hat keinen Einfluß, wenn die Zeit nicht läuft.
Signalzustand abfragen	Solange die Zeit läuft, ergibt eine Signalzustandsabfrage nach ”1” an Ausgang **Q** das Ergebnis ”1”, unabhängig von der Länge des Eingangssignals.
Aktuellen Zeitwert abfragen	Der aktuelle Zeitwert kann am Ausgang **BI** und durch den Funktionswert S_PEXT abgefragt werden.

17.2.4 Zeit als Einschaltverzögerung starten

Beschreibung

Das Ausgangssignal wechselt nur von ”0” auf ”1”, wenn die programmierte Zeit abgelaufen ist, und das Eingangssignal noch immer ”1” beträgt. D.h. der Ausgang wird verzögert eingeschaltet. Eingangssignale, deren Zeitdauer kürzer als die der programmierten Zeit sind, erscheinen am Ausgang nicht.

Bild 17-5 veranschaulicht die Funktionsweise des Zeitgliedes ”Zeit als Einschaltverzögerung starten”.

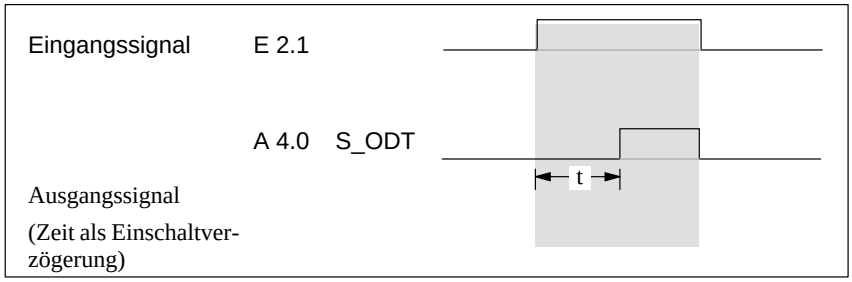


Bild 17-5 *Zeitglied ”Zeit als Einschaltverzögerung starten”*

Funktionsweise

Tabelle 17-12 Funktionsweise ”Zeit als Einschaltverzögerung starten”

Operation	Funktionsweise
Zeit starten	Die Operation ”Zeit als Eingangsverzögerung starten” startet eine angegebene Zeit, wenn der Signalzustand am Starteingang **S** von ”0” auf ”1” wechselt. Um die Zeit freizugeben, ist immer ein Signalwechsel erforderlich.
Zeit anhalten	Wechselt der Signalzustand an Eingang **S** von ”1” auf ”0”, während die Zeit läuft, wird sie angehalten.
Laufzeit festlegen	Die Zeit läuft mit dem Wert weiter, der an Eingang **TV** angegeben ist, solange der Signalzustand an Eingang S = 1 ist.
Rücksetzen	Die Zeit wird zurückgesetzt, wenn der Rücksetzeingang **R** von ”0” auf ”1” wechselt, während die Zeit läuft. Durch diesen Wechsel werden auch der Zeitwert und die Zeitbasis auf Null zurückgesetzt. Die Zeit wird auch dann zurückgesetzt, wenn R = 1 ist, während die Zeit nicht läuft.
Signalzustand abfragen	Eine Signalzustandsabfrage nach ”1” an Ausgang **Q** ergibt ”1”, wenn die Zeit fehlerfrei abgelaufen ist und Eingang S noch immer ”1” ist. Wurde die Zeit angehalten, ergibt eine Signalzustandsabfrage nach ”1” immer ”0”. Eine Signalzustandsabfrage nach ”1” an Ausgang Q ergibt auch dann ”0”, wenn die Zeit nicht läuft und das VKE (Verknüpfungsergebnis, siehe **/232/**) an Eingang S noch immer ”1” beträgt.
Aktuellen Zeitwert abfragen	Der aktuelle Zeitwert kann am Ausgang **BI** und durch den Funktionswert S_ODT abgefragt werden.

17.2.5 Zeit als speichernde Einschaltverzögerung starten

Beschreibung

Das Ausgangssignal wechselt nur von ”0” auf ”1”, wenn die programmierte Zeit abgelaufen ist, unabhängig davon, wie lange das Eingangssignal auf ”1” bleibt.

Bild 17-6 veranschaulicht die Funktionsweise des Zeitgliedes ”Zeit als speichernde Einschaltverzögerung starten”.

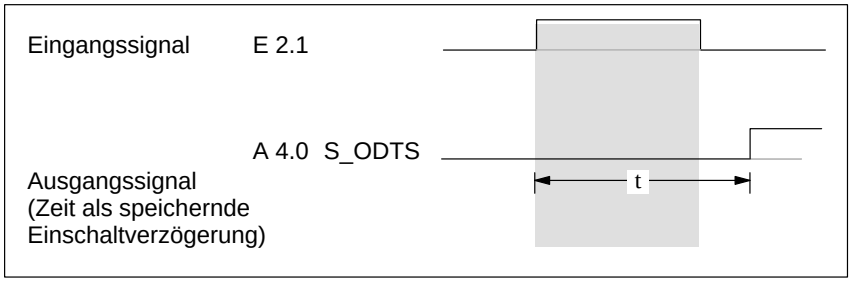


Bild 17-6 *Zeitglied ”Zeit als speichernde Einschaltverzögerung starten”*

Funktionsweise

Tabelle 17-13 Funktionsweise ”Zeit als speichernde Einschaltverzögerung starten”

Operation	Funktionsweise
Zeit starten	Die Operation ”Zeit als speichernde Eingangsverzögerung starten” startet eine angegebene Zeit, wenn der Signalzustand am Starteingang **S** von ”0” auf ”1” wechselt. Um die Zeit freizugeben, ist immer ein Signalwechsel erforderlich.
Zeit neu starten	Die Zeit wird mit dem angegebenen Wert neu gestartet, wenn Eingang **S** von ”0” auf ”1” wechselt, während die Zeit läuft.
Laufzeit festlegen	Die Zeit läuft auch dann mit dem Wert weiter, der an Eingang **TV** angegeben ist, wenn der Signalzustand an Eingang **S** noch vor Ablauf der Zeit auf ”0” wechselt.
Rücksetzen	Wechselt der Rücksetzeingang **R** von ”0” auf ”1”, wird die Zeit unabhängig vom VKE (Verknüpfungsergebnis siehe **/232/**) an Eingang S zurückgesetzt.
Signalzustand abfragen	Eine Signalzustandsabfrage nach ”1” an Ausgang **Q** ergibt nach Ablauf der Zeit unabhängig vom Signalzustand an Eingang S das Ergebnis ”1”.
Aktuellen Zeitwert abfragen	Der aktuelle Zeitwert kann am Ausgang **BI** und durch den Funktionswert S_ODTS abgefragt werden.

17.2.6 Zeit als Ausschaltverzögerung starten

Beschreibung

Bei einem Signalzustandswechsel von ”0” nach ”1” am Starteingang S erscheint am Ausgang Q der Zustand ”1”. Wechselt der Zustand am Starteingang von ”1” nach ”0”, wird die Zeit gestartet. Erst nach Ablauf der Zeitdauer nimmt der Ausgang den Signalzustand ”0” an. Der Ausgang wird also verzögert abgeschaltet.

Bild 17-7 veranschaulicht die Funktionsweise des Zeitgliedes ”Zeit als Ausschaltverzögerung starten”.

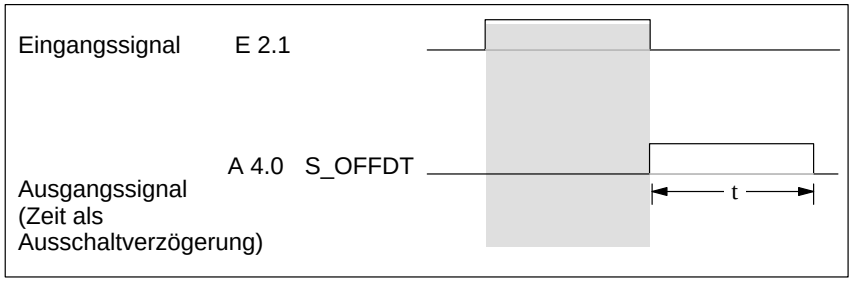


Bild 17-7 *Zeitglied ”Zeit als Ausschaltverzögerung starten”*

Funtktionsweise

Tabelle 17-14 Funktionsweise ”Zeit als Ausschaltverzögerung starten”

Operation	Funktionsweise
Zeit starten	Die Operation Zeit als Ausschaltverzögerung starten startet die angegebene Zeit, wenn der Signalzustand am Starteingang **S** von ”1” auf ”0” wechselt. Es ist immer ein Signalwechsel erforderlich, um die Zeit freizugeben.
Zeit neu starten	Die Zeit wird wieder neu gestartet, wenn der Signalzustand an Eingang **S** erneut von ”1” auf ”0” wechselt (z.B nach dem Rücksetzen).
Laufzeit festlegen	Die Zeit läuft mit dem Wert, der an Eingang **TV** angegeben ist.
Rücksetzen	Wechselt der Rücksetzeingang **R** von ”0” auf ”1”, während die Zeit läuft, wird die Zeit zurückgesetzt.
Signalzustand abfragen	Eine Signalzustandsabfrage nach ”1” an Ausgang **Q** ergibt ”1”, wenn der Signalzustand an Eingang S = 1 ist oder die Zeit läuft.
Aktuellen Zeitwert abfragen	Der aktuelle Zeitwert kann am Ausgang **BI** und durch den Funktionswert S_OFFDT abgefragt werden.

17.2.7 Programmbeispiel für einen verlängerten Impuls

Beispiel S_PEXT

Beispiel 17-6 zeigt ein Programm für die Anwendung der Zeifunktion ”verlängerter Impuls”.

```
FUNCTION_BLOCK ZEITGEBER
VAR_INPUT
 MEINEZEIT: TIMER;
END_VAR
VAR_OUTPUT
 ERGEBNIS: S5TIME;
END_VAR
VAR
 SETZEN           : BOOL;
 RUECKSETZEN      : BOOL;
 BCDWERT          : S5TIME;//Zeitb. u. Restwert
                          //BCD codiert
 BINWERT          : WORD; //Zeitwert binär
 VORBESETZUNG     : S5TIME;
END_VAR
BEGIN
 A0.0:= 1;
 SETZEN:= E0.0;
 RUECKSETZEN:= E0.1;
 VORBESETZUNG:= T#25S;

 BCDWERT:= S_PEXT(T_NO:= MEINEZEIT,
                  S   := SETZEN,
                  TV  := VORBESETZUNG,
                  R   := RUECKSETZEN,
                  BI  := BINWERT,
                  Q   := A0.7);

ERGEBNIS:=BCDWERT; //Weiterverarbeitung
                   //als Ausgangsparameter
AW4:= BINWERT      //An Ausgabe zur Anzeige
END_FUNCTION_BLOCK
```

Beispiel 17-6 Beispiel einer Zeitfunktion

17.2.8 Auswahl des richtigen Zeitglieds

Bild 17-8 bietet einen Überblick über die fünf verschiedenen Zeiten, die in diesem Abschnitt beschrieben wurden. Diese Übersicht soll Ihnen helfen, die für Ihre Zwecke adäquaten Zeitgeber auszuwählen.

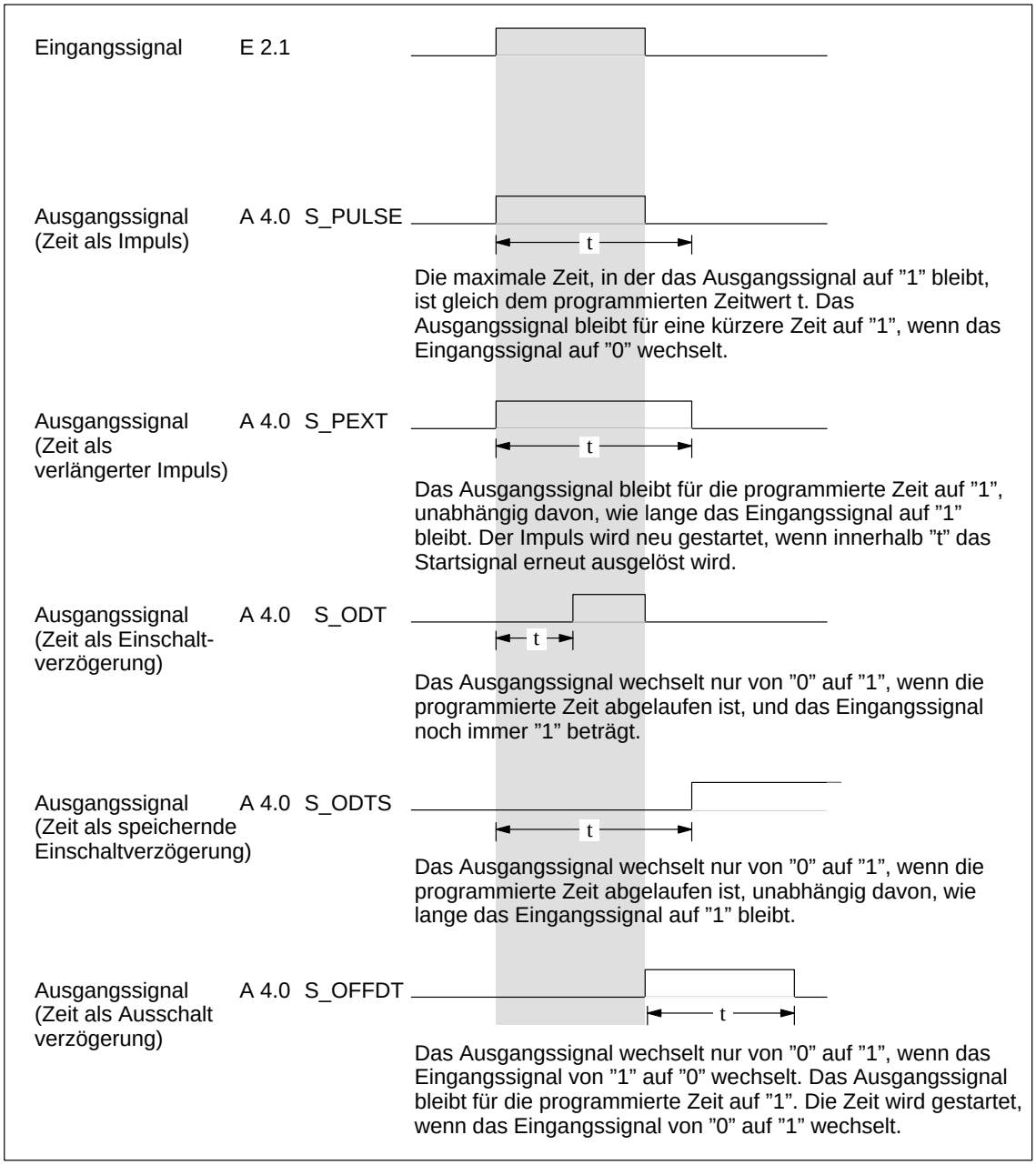


Bild 17-8 *Auswahl des richtigen Zeitgebers*

SCL-Standardfunktionen 18

Übersicht

SCL stellt Ihnen für die Lösung häufig auftretender Aufgaben eine Reihe von Standardfunktionen zur Verfügung, die Sie in Ihren SCL-Bausteinen aufrufen können.

Kapitelübersicht

18.1 Konvertierung von Datentypen

Übersicht

Wenn Sie zwei Operanden ungleichen Datentyps miteinander verknüpfen oder Variablen Ausdrücke zuweisen, müssen Sie im Einzelfall die Verträglichkeit der beteiligten Datentypen beachten. In folgenden Fällen sind falsche Ergebnisse zu erwarten:

- beim Wechsel in eine andere Typklasse, z.B. von einem Bitdatentyp in einen numerischen Datentyp
- beim Wechsel innerhalb einer Typklasse, wenn der Zieldatentyp weniger mächtig als der Quelldatentyp ist.

Deshalb müssen Sie hier eine **explizite** Datentyp-Konvertierung durchführen. Die notwendigen Informationen finden Sie in Kapitel 18.2.

Wenn diese beiden Fälle nicht zutreffen, erzwingt der Compiler eine automatische Konvertierung in ein gemeinsames Format. Dies wird im folgenden als **implizite** Datentyp-Konvertierung bezeichnet.

Implizite Datentyp-Konvertierungen

Innerhalb der in Tabelle 18-1 definierten Klassen von Hilfs-Datentypen wird vom Compiler eine implizite Datentyp-Konvertierungen in der angegebenen Reihenfolge durchgeführt. Als gemeinsames Format zweier Operanden ist jeweils der kleinste Standardtyp definiert, dessen Wertebereich beide Operanden beinhaltet. So ist z. B. das gemeinsame Format von Byte und Integer - Integer.

Beachten Sie bitte auch, daß bei einer Datentyp-Konvertierung innerhalb der Klasse `ANY_BIT` führende Bits auf `0` gesetzt werden.

Tabelle 18-1 Reihenfolge impliziter Datentyp-Konvertierungen

Klassen	Reihenfolge der Konvertierung
ANY_BIT	`BOOL` ⇒ `BYTE` ⇒ `WORD` ⇒ `DWORD`
ANY_NUM	`INT` ⇒ `DINT` ⇒ `REAL`

Das folgende Beispiel soll die implizite Konvertierung von Datentypen verdeutlichen:

```
FUNCTION_BLOCK FB10
VAR
      PID_REGLER_1:BYTE;
      PID_REGLER_2:WORD;
END_VAR
BEGIN
IF (PID_REGLER_1 <> PID_REGLER_2) THEN
      // ...
(* In der Bedingung obiger IF / THEN - Anweisung
wird PID_REGLER_1 implizit in eine Variable vom
Datentyp WORD konvertiert. *)
END_FUNCTION_BLOCK
```

Beispiel 18-1 Implizite Datentyp-Konvertierung

18.2 Standardfunktionen für Datentyp-Konvertierungen

Explizite Datentyp-konvertierung

Die explizite Datentyp-Konvertierung führen Sie mit Standardfunktionen durch. Die Standardfunktionen finden Sie in den Tabellen 18-2 und 18-3.

Funktionsaufruf

Die allgemeinen Funktionsaufrufe sind in Kapitel 16 ausführlich beschrieben.

- Eingangsparameter
  Jede Funktion zur Konvertierung eines Datentyps hat genau einen Eingangsparameter mit dem Namen `IN`. Da es sich um eine Funktion mit nur einem Parameter handelt, müssen Sie nur den Aktualparameter angeben.
- Funktionswert
  Der Funktionswert ist immer der Rückgabewert der Funktion. Die beiden Tabellen zeigen die Regeln, nach denen ein Datentyp konvertiert wird. In Tabelle 18-3 ist auch angegeben, ob die jeweilige Funktion das `OK`-Flag beeinflußt.
- Namensgebung
  Da die Datentypen des Eingangsparameters und des Funktionswertes aus dem jeweiligen Funktionsnamen hervorgehen, sind sie in den Tabellen 18-2 und 18-3 nicht gesondert aufgelistet: z.B. bei Funktion `BOOL_TO_BYTE` ist der Datentyp des Eingangsparameters `BOOL`, der Datentyp des Funktionswertes `BYTE`.

Liste der Konvertierungs-Funktionen (Klasse A)

Tabelle 18-2 stellt die Datentyp-Konvertierungs-Funktionen der Klasse A dar. Diese Funktionen setzt der Compiler implizit ab oder Sie können sie explizit angeben. Das Ergebnis ist immer definiert.

Tabelle 18-2 Datentyp-Konvertierungs-Funktionen der Klasse A

Funktionsname	Konvertierungsregel
BOOL_TO_BYTE	Ergänzung führender Nullen
BOOL_TO_DWORD	
BOOL_TO_WORD	
BYTE_TO_DWORD	
BYTE_TO_WORD	
CHAR_TO_STRING	Transformation in einen String (der Länge 1), der das gleiche Zeichen enthält.
DINT_TO_REAL	Transformation in `REAL` entsprechend IEEE-Norm. Der Wert kann sich - wegen der anderen Genauigkeit bei `REAL` - ändern.
INT_TO_DINT	Das höherwertige Wort des Funktionswertes wird bei einem negativen Eingangsparameter mit `16#FFFF`, sonst mit Nullen aufgefüllt. Der Wert bleibt gleich.
INT_TO_REAL	Transformation in `REAL` entsprechend IEEE-Norm. Der Wert bleibt gleich.
WORD_TO_DWORD	Ergänzung führender Nullen

Liste der Konvertierungs-Funktionen (Klasse B)

Tabelle 18-3 stellt die Datentyp-Konvertierungs-Funktionen der Klasse B dar. Diese Funktionen müssen Sie explizit angeben. Das Ergebnis kann auch undefiniert sein, wenn die Größe des Zieldatentyps unzureichend ist.

Sie können diesen Fall entweder selbst überprüfen, indem Sie eine Grenzprüfung davorschalten, oder die Überprüfung vom System durchführen lassen, indem Sie bei der Übersetzung die OK-Option einschalten. Das System setzt dann in den Fällen, in denen das Ergebnis undefiniert ist, die OK-Variable auf FALSE. Für die Auswertung müssen Sie selbst sorgen.

Tabelle 18-3 Datentyp-Konvertierungs-Funktionen der Klasse B

Funktionsname	Konvertierungsregel	OK
BYTE_TO_BOOL	Kopieren des niedrigstwertigen Bits	J
BYTE_TO_CHAR	Übernahme des Bitstrings	N
CHAR_TO_BYTE	Übernahme des Bitstrings	N
CHAR_TO_INT	Der im Eingangsparameter vorhandene Bitstring wird in das niedrigerwertige Byte des Funktionswertes eingetragen. Das höherwertige Byte wird mit Nullen aufgefüllt.	N
DATE_TO_DINT	Übernahme des Bitstrings	N
DINT_TO_DATE	Übernahme des Bitstrings	J
DINT_TO_DWORD	Übernahme des Bitstrings	N
DINT_TO_INT	Kopieren des Bits für das Vorzeichen; Der im Eingangsparameter vorhandene Wert wird im Datentyp INT interpretiert. Wenn der Wert kleiner als `-32_768` oder größer als `32_767` ist, dann wird die OK-Variable gleich FALSE gesetzt.	J
DINT_TO_TIME	Übernahme des Bitstrings	N
DINT_TO_TOD	Übernahme des Bitstrings	J
DWORD_TO_BOOL	Kopieren des niedrigstwertigen Bits	J
DWORD_TO_BYTE	Kopieren der 8 niedrigstwertigen Bits	J
DWORD_TO_DINT	Übernahme des Bitstrings	N
DWORD_TO_REAL	Übernahme des Bitstrings	N
DWORD_TO_WORD	Kopieren der 16 niedrigstwertigen Bits	J
INT_TO_CHAR	Übernahme des Bitstrings	J
INT_TO_WORD	Übernahme des Bitstrings	N
REAL_TO_DINT	Runden des IEEE-REAL-Wertes auf DINT. Wenn der Wert kleiner als `-2_147_483_648` oder größer als `2_147_483_647` ist, dann wird die OK-Variable gleich FALSE gesetzt.	J
REAL_TO_DWORD	Übernahme des Bitstrings	N
REAL_TO_INT	Runden des IEEE-REAL-Wertes auf INT. Wenn der Wert kleiner als `-32_768` oder größer als `32_767` ist, dann wird die `OK`-Variable gleich FALSE gesetzt.	J

Tabelle 18-3 Datentyp-Konvertierungs-Funktionen der Klasse B

Funktionsname	Konvertierungsregel	OK
STRING_TO_CHAR	Kopieren des ersten Zeichens des Strings. Wenn der STRING nicht eine Länge von 1 hat, dann wird die OK-Variable gleich FALSE gesetzt.	J
TIME_TO_DINT	Übernahme des Bitstrings	N
TOD_TO_DINT	Übernahme des Bitstrings	N
WORD_TO_BOOL	Kopieren des niedrigstwertigen Bits	J
WORD_TO_BYTE	Kopieren der niedrigstwertigen 8 Bits	J
WORD_TO_INT	Übernahme des Bitstrings	N
WORD_TO_BLOCK_DB	Das Bitmuster von WORD wird als Datenbausteinnummer interpretiert.	N
BLOCK_DB_TO_WORD	Die Datenbausteinnummer wird als Bitmuster von WORD interpretiert.	N

Hinweis

Sie haben außerdem die Möglichkeit, **IEC-Funktionen** zur Datentyp-Konvertierung zu nutzen. Informationen zu einzelnen IEC-Funktionen finden Sie in **/235/**.

Beispiele für explizite Konvertierung

Im Beispiel 18-2 ist eine explizite Konvertierung notwendig, da der Zieldatentyp weniger mächtig ist als der Quelldatentyp.

```
FUNCTION_BLOCK FB10
VAR
      SCHALTER    : INT;
      REGLER      : DINT;
END_VAR

BEGIN
SCHALTER := DINT_TO_INT (REGLER);
(* INT ist weniger mächtig als DINT *)
// ...
END_FUNCTION_BLOCK
```

Beispiel 18-2 Zieldatentyp paßt nicht zum Quelldatentyp

Im Beispiel 18-3 ist eine explizite Datentyp-Konvertierung notwendig, da der Datentyp REAL für einen arithmetischen Ausdruck mit dem Operartor MOD nicht zulässig ist:

```
FUNCTION_BLOCK FB20
    VAR
      intwert:INT:=17;
      KONV2 := INT;
    END_VAR

BEGIN
    KONV2 := intwert MOD REAL_TO_INT (2.3);
    (* MOD darf nur auf Daten vom Typ INT oder
    DINT angewendet werden. *)
// ...
END_FUNCTION_BLOCK
```

Beispiel 18-3 Konvertierung wegen unzulässigem Datentyp

Im Beispiel 18-4 ist eine Konvertierung notwendig, da nicht der richtige Datentyp für einen logischen Operator vorliegt. Der Operator NOT darf nur auf Daten vom Typ BOOL, BYTE, WORD oder DWORD angewendet werden.

```
FUNCTION_BLOCK FB30
    VAR
      intwert  : INT := 17;
      KONV1    : WORD;
    END_VAR

BEGIN
    KONV1 := NOT INT_TO_WORD(intwert);
    (* NOT darf nicht auf Daten vom Typ INT
     angewendet werden. *)
// ...
END_FUNCTION_BLOCK
```

Beispiel 18-4 Konvertierung wegen unzulässigem Datentyp

Das Beispiel 18-5 zeigt die Konvertierung bei Ein- und Ausgaben an die Peripherie:

```
FUNCTION_BLOCK FB40
    VAR
      radius_ein  : WORD;
      radius      : INT;
    END_VAR

BEGIN
    radius_ein  := EB0;
    radius      := WORD_TO_INT(radius_ein);
(* Konvertierung bei Wechsel in eine andere Typklasse.
Wert kommt von der Eingabe und wird zur Weiterberechnung
konvertiert. *)

    radius  := Radius(flaeche:= Kreisdaten.flaeche);
    AB0     := WORD_TO_BYTE(INT_TO_WORD(radius));
(* Radius wird rückgerechnet aus der Fläche und liegt in
Integer vor. Zur Ausgabe wird der Wert zuerst in eine
andere Typklasse (INT_TO_WORD) und dann in einen weniger
mächtigen Typ (WORD_TO_BYTE) konvertiert. *)
    // ...
END_FUNCTION_BLOCK
```

Beispiel 18-5 Konvertierung bei Ein- und Ausgaben

Funktionen zum Runden und Abschneiden

Zu den Datentyp-Konvertierungen zählen auch die Funktionen zum Runden und Abschneiden von Zahlen. Tabelle 18-4 zeigt die Namen, Datentypen (für den Eingangsparameter und den Funktionswert) und Aufgaben dieser Funktionen.

Tabelle 18-4 Funktionen zum Runden und Abschneiden

Funktionsname	Datentyp Eingangsparameter	Datentyp Funktionswert	Aufgabe
ROUND	REAL	DINT	Runden (Bilden einer DINT-Zahl)
TRUNC	REAL	DINT	Abschneiden (Bilden einer DINT-Zahl)

Die unterschiedliche Wirkungsweise demonstrieren die folgenden Beispiele:

```
• ROUND (3.14)     // Hier wird abgerundet,
                   // daher ist das Ergebnis: 3

• ROUND (3.56)     // Hier wird aufgerundet,
                   // daher ist das Ergebnis: 4

• TRUNC (3.14)     // Hier wird abgeschnitten,
                   // daher ist das Ergebnis: 3

• TRUNC (3.56)     // Hier wird abgeschnitten,
                   // daher ist das Ergebnis: 3
```

18.3 Numerische Standardfunktionen

Funktionalität

Jede numerische Standardfunktion hat einen Eingangsparameter. Das Ergebnis ist immer der Funktionswert. Jede der Tabellen 18-5, 18-6 bzw. 18-7 spezifiziert eine Gruppe von numerischen Standardfunktionen durch die Funktionsnamen, und die Datentypen. Der Datentyp ANY_NUM steht für INT, DINT oder REAL.

Liste der allgemeinen Funktionen

Allgemeine Funktionen sind die Funktionen zur Berechnung des absoluten Betrages, des Quadrats oder der Quadratwurzel einer Größe.

Tabelle 18-5 Allgemeine Funktionen

Funktionsname	Datentyp Eingangs-parameter	Datentyp Funktionswert	Beschreibung
ABS	ANY_NUM[1)]	ANY_NUM	Absolutbetrag
SQR	ANY_NUM[1)]	REAL	Quadrat
SQRT	ANY_NUM[1)]	REAL	Wurzel

1) *Beachten Sie, daß Eingangsparameter vom Typ `ANY_NUM` intern in Real-Variablen umgewandelt werden.*

Liste der logarithmischen Funktionen

Logarithmische Funktionen sind die Funktionen zur Berechnung eines Exponentialwertes oder eines Logarithmus einer Größe.

Tabelle 18-6 Logarithmische Funktionen

Funktionsname	Datentyp Eingangs-parameter	Datentyp Funktionswert	Beschreibung
EXP	ANY_NUM[1)]	REAL	e hoch `IN`
EXPD	ANY_NUM[1)]	REAL	10 hoch `IN`
LN	ANY_NUM[1)]	REAL	Natürlicher Logarithmus
LOG	ANY_NUM[1)]	REAL	Dekadischer Logarithmus

1) *Beachten Sie, daß Eingangsparameter vom Typ `ANY_NUM` intern in Real-Variablen umgewandelt werden.*

Hinweis

Sie haben außerdem die Möglichkeit, **IEC-Funktionen** als numerische Standardfunktion zu nutzen. Kopieren Sie in diesem Fall die gewünschte Funktion aus der STEP 7-Bibliothek STDLIBS\IEC in Ihr Programmverzeichnis. Informationen zu einzelnen IEC-Funktionen finden Sie in **/235/**.

Liste der trigonometrischen Funktionen

Die in Tabelle 18-7 dargestellten trigonometrischen Funktionen erwarten und berechnen Größen von Winkeln im Bogenmaß.

Tabelle 18-7 Trigonometrische Funktionen

Funktionsname	Datentyp Eingangsparameter	Datentyp Funktionswert	Beschreibung
ACOS	ANY_NUM[1)]	REAL	Arcus-Cosinus
ASIN	ANY_NUM[1)]	REAL	Arcus-Sinus
ATAN	ANY_NUM[1)]	REAL	Arcus-Tangens
COS	ANY_NUM[1)]	REAL	Cosinus
SIN	ANY_NUM[1)]	REAL	Sinus
TAN	ANY_NUM[1)]	REAL	Tangens

1) *Beachten Sie, daß Eingangsparameter vom Typ `ANY_NUM` intern in Real-Variablen umgewandelt werden.*

Beispiele

Tabelle 18-8 zeigt mögliche Aufrufe von numerischen Standardfunktionen und die jeweiligen Ergebnisse:

Tabelle 18-8 Aufrufe numerischer Standardfunktionen

Aufruf	Ergebnis
ERGEBNIS := ABS (-5);	5
ERGEBNIS := SQRT (81.0);	9
ERGEBNIS := SQR (23);	529
ERGEBNIS := EXP (4.1);	60.340 ...
ERGEBNIS := EXPD (3);	1_000
ERGEBNIS := LN (2.718_281);	1
ERGEBNIS := LOG (245);	2.389_166 ...
PI := 3. 141 592; ERGEBNIS := SIN (PI / 6);	0.5
ERGEBNIS := ACOS (0.5);	1.047_197 (=PI / 3)

18.4 Bitstring-Standardfunktionen

Funktionalität

Jede Bitstring Standardfunktion hat zwei Eingangsparameter, die durch IN bzw. N bezeichnet werden. Das Ergebnis ist immer der Funktionswert. Tabelle 18-9 zeigt die Funktionsnamen und die Datentypen der zwei Eingangsparameter und des Funktionswertes. Es bedeuten:

- Eingangsparameter IN: Puffer, in dem die Bitschiebeoperationen durchgeführt werden.
- Eingangsparameter N: Anzahl der Rotationen bei den Umlaufpufferfunktionen ROL und ROR bzw. die Anzahl der zu schiebenden Stellen bei SHL und SHR.

Liste der Funktionen

Tabelle 18-9 zeigt die möglichen Bitstring-Standardfunktionen.

Tabelle 18-9 Bitstring-Standardfunktionen

Funktionsname	Datentyp Eingangsparameter `IN`	Datentyp Eingangsparameter `N`	Datentyp Funktionswert	Aufgabe
ROL	BOOL	INT	BOOL	Der im Parameter `IN` vorhandene Wert wird um soviele Bitstellen nach links rotiert, wie der Inhalt des Parameters `N` angibt.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
ROR	BOOL	INT	BOOL	Der im Parameter `IN` vorhandene Wert wird um soviele Bitstellen nach rechts rotiert, wie der Inhalt des Parameters `N` angibt.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
SHL	BOOL	INT	BOOL	In dem im Parameter `IN` vorhandenen Wert werden soviele Bitstellen nach links geschoben und soviele Bitstellen auf der rechten Seite durch 0 ersetzt, wie der Inhalt des Parameters `N` angibt.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	
SHR	BOOL	INT	BOOL	In dem im Parameter `IN` vorhandenen Wert werden soviele Bitstellen nach rechts geschoben und soviele Bitstellen auf der linken Seite durch 0 ersetzt, wie der Inhalt des Parameters `N` angibt.
	BYTE	INT	BYTE	
	WORD	INT	WORD	
	DWORD	INT	DWORD	

Hinweis

Sie haben außerdem die Möglichkeit, **IEC-Funktionen** für Bitstringoperationen zu nutzen. Kopieren Sie in diesem Fall die gewünschte Funktion aus der STEP 7-Bibliothek STDLIBS\IEC in Ihr Programmverzeichnis. Informationen zu einzelnen IEC-Funktionen finden Sie in **/235/**.

Beispiele

Tabelle 18-10 zeigt mögliche Aufrufe von Bitstring-Standardfunktionen und die jeweiligen Ergebnisse.

Tabelle 18-10 Aufrufe von Bitstring-Standardfunktionen

Aufruf	ERGEBNIS
ERGEBNIS := ROL (IN:=2#1101_0011, N:=5); // IN := 211 dezimal	2#0111_1010 (= 122 dezimal)
ERGEBNIS := ROR (IN:=2#1101_0011, N:=2); // IN := 211 dezimal	2#1111_0100 (= 244 dezimal)
ERGEBNIS := SHL (IN:=2#1101_0011, N:=3); // IN := 211 dezimal	2#1001_1000 (= 152 dezimal)
ERGEBNIS := SHR (IN:=2#1101_0011, N:=2); // IN := 211 dezimal	2#0011_0100 (= 52 dezimal)

Aufrufschnittstelle

19

Übersicht

Die S7-CPUs enthalten in das Betriebssystem integrierte System- und Standardfunktionen, die Sie bei der Programmierung in SCL nutzen können. Im einzelnen sind dies:

- Organisationsbausteine (OB)
- Systemfunktionen (SFC)
- Systemfunktionsbausteine (SFB)

Kapitelübersicht

19.1 Aufrufschnittstelle

Übersicht

Sie können die Bausteine symbolisch oder absolut aufrufen. Dazu benötigen Sie entweder den symbolischen Namen, der in der Symboltabelle vereinbart sein muß, oder die Nummer für die absolute Bezeichnung des Bausteins.

Beim Aufruf müssen Sie den **Formalparametern**, deren Namen und Datentypen bei der Erstellung des parametrierbaren Bausteins festgelegt wurden, die **Aktualparameter** zuordnen, mit dessen Werten der Baustein zur Laufzeit Ihres Programms arbeitet.

Alle Informationen, die Sie dazu benötigen finden Sie im Referenzhandbuch **/235/** beschrieben. Dieses Handbuch gibt Ihnen einen Überblick über die grundsätzlich bei S7 verfügbaren Funktionen sowie – als Nachschlageinformation – detaillierte Schnittstellenbeschreibungen für die Nutzung in Ihrem Anwenderprogramm.

Beispiel SFC 31

Die folgenden Befehlszeilen ermöglichen den Aufruf der Systemfunktion SFC 31 (Uhrzeitalarm abfragen):

```
FUNCTION_BLOCK FB20

VAR
      Ergebnis: INT;
END_VAR
BEGIN
      // ...
      Ergebnis:= SFC31 (OB_NR:=10,STATUS:=MW100);
      // ...
      // ...
END_FUNCTION_BLOCK
```

Beispiel 19-1 Abfrage des Uhrzeitalarms

Ergebnisse

Der Funktionswert ist vom Typ Integer. Wenn sein Wert > = 0 ist, so ist der Baustein fehlerfrei abgelaufen, ist der Wert < 0, so ist ein Fehler aufgetreten. Nach dem Aufruf können Sie den implizit definierten Ausgangsparameter ENO auswerten.

Bedingter Aufruf

Für einen bedingten Aufruf müssen Sie den vordefinierten **Eingangsparameter EN** mit 0 belegen (z.B. über den Eingang E0.3), dann wird der Baustein nicht aufgerufen. Wird EN mit 1 belegt, so wird die Funktion aufgerufen. Der **Ausgangsparameter ENO** wird in diesem Fall auch auf ”1” gesetzt (sonst auf ”0”), falls während der Bearbeitung des Bausteins kein Fehler auftritt.

Hinweis

Bei Funktionsbausteinen bzw. Systemfunktionsbausteinen muß die Information, die bei einer Funktion mit dem Funktionswert übergeben werden kann, in Ausgangsparameter gespeichert werden. Diese werden anschließend über den Instanz-Datenbaustein ausgewertet. Weitere Information siehe Kapitel 16.

19.2 Übergabeschnittstelle zu OB

Organisations-bausteine

Organisationsbausteine bilden die Schnittstelle zwischen dem Betriebssystem der CPU und dem Anwenderprogramm. Mit Hilfe von OB können Programmteile gezielt zur Ausführung gebracht werden:

- beim Anlauf der CPU
- in zyklischer oder auch zeitlich getakteter Ausführung
- zu bestimmten Uhrzeiten oder Tagen
- nach Ablauf einer vorgegebenen Zeitdauer
- beim Auftreten von Fehlern
- beim Auftreten von Prozeß- oder Kommunikationsalarmen

Organisationsbausteine werden entsprechend der ihnen zugeordneten Priorität bearbeitet.

Verfügbare OB

Nicht alle CPUs können alle in S7 verfügbaren OB bearbeiten. Sie entnehmen den Datenblättern zu Ihrer CPU, welche OB Ihnen zur Verfügung stehen.

Weitere Informationen

Weitere Informationen entnehmen Sie der Online-Hilfe sowie den folgenden Handbüchern:

- **/70/** Handbuch: *Automatisierungssystem S7-300, Aufbauen, CPU-Daten* Dieses Handbuch enthält die Datenblätter, die den Leistungsumfang der verschiedenen S7-300 CPUs beschreiben. Dazu gehören auch die möglichen Startereignisse für jeden OB.
- **/100/** Installationshandbuch: *Automatisierungssystem S7-400, M7-400, Aufbauen.* Dieses Handbuch enthält die Datenblätter, die den Leistungsumfang der verschiedenen S7-400 CPUs beschreiben. Dazu gehören auch die möglichen Startereignisse für jeden OB.

Anhänge

Formale Sprachbeschreibung

A

Sprach-beschreibung von SCL

Basis für die Sprachbeschreibung in den einzelnen Kapiteln sind Syntaxdiagramme. Sie geben Ihnen einen guten Einblick in den syntaktischen (d.h. grammatikalischen) Aufbau von SCL. Eine vollständige Zusammenstellung aller Diagramme mit den Sprachelementen finden Sie in Anhang B und C.

Kapitelübersicht

A.1 Übersicht

Was ist ein 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 eingeführten Regeln aufbauen.

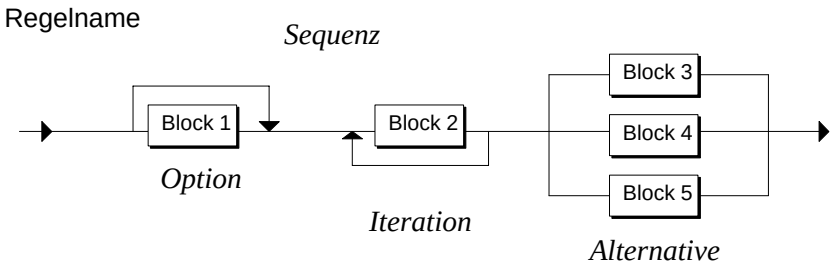


Bild A-1 *Beispiel eines Syntaxdiagramms*

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

- Sequenz: Folge von Blöcken
- Option: Überspringbarer Zweig
- Iteration: Wiederholung von Zweigen
- Alternative: Verzweigung

Welche Arten von Blöcken gibt es?

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

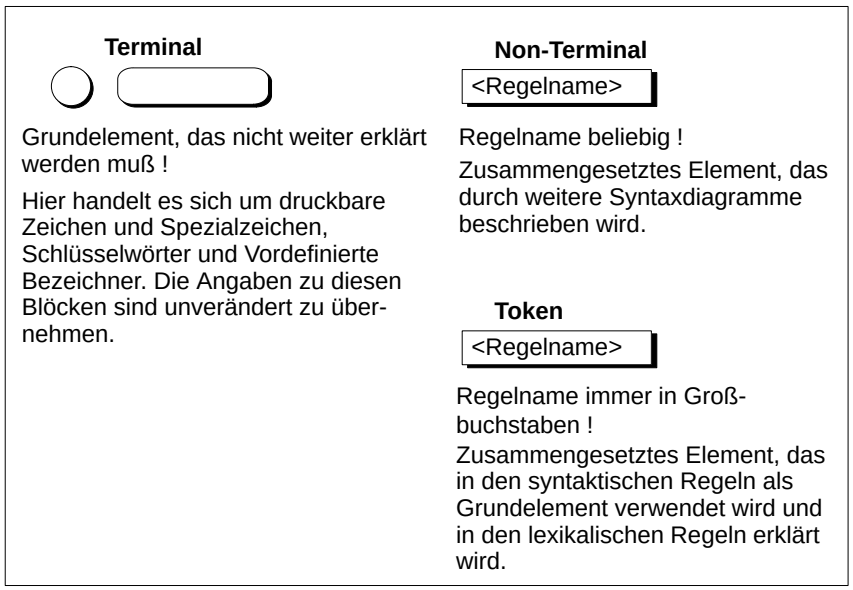


Bild A-2 *Symbolarten von Blöcken*

Regeln

Die Regeln, die Sie für den Aufbau Ihres SCL-Programms verwenden können, sind in die Stufen **lexikalische** und **syntaktische** Regeln eingeteilt.

Lexikalische Regeln

Die lexikalischen Regeln beschreiben die Struktur der Elemente (Token), 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.

BEZEICHNER

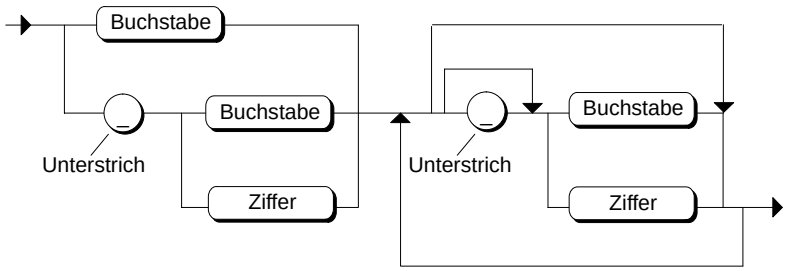


Bild A-3 *Beispiel für lexikalische Regel*

Das Beispiel zeigt die lexikalische Regel BEZEICHNER. Sie beschreibt den Aufbau eines Bezeichners (Namen), z. B.:

```
MESS_FELD_12
SOLLWERT_B_1
```

Syntaktische Regeln

Aufbauend auf den lexikalischen Regeln wird in den syntaktischen Regeln die Struktur von SCL beschrieben. Im Rahmen dieser Regeln können Sie Ihr SCL-Programm formatfrei erstellen:

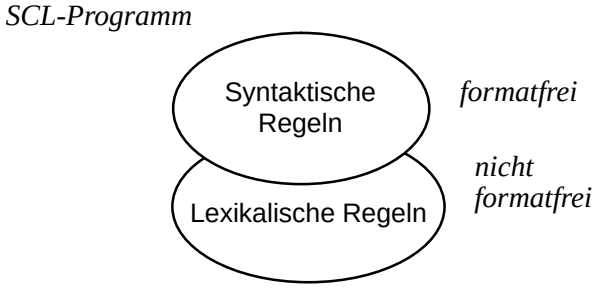


Bild A-4 *Regelstufen und Formatfreiheit*

Formales

Jede Regel hat einen Regelnamen, der vorangestellt ist. Wenn die Regel in einer übergeordneten Regel verwendet wird, so taucht der Regelname in einem Rechteck auf.

Ist der Regelname in Großbuchstaben geschrieben, so handelt es sich um ein Token, das in den lexikalischen Regeln beschrieben wird.

Semantik

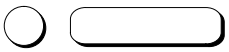
In den Regeln kann nur der formale Aufbau der Sprache dargestellt werden. Die Bedeutung, d.h. Semantik, geht nicht immer daraus hervor. Deshalb wird an wichtigen Stellen Zusatzinformation neben die Regeln geschrieben. Beispiele dafür sind:

- bei gleichartigen Elementen mit unterschiedlicher Bedeutung wird ein Zusatznamen angegeben: z. B. in Regel Datumsangabe bei DEZIMAL-ZIFFERNFOLGE Jahr, Monat oder Tag. Der Name deutet auf die Verwendung hin.
- Wichtige Einschränkungen werden neben den Regeln vermerkt: z. B. bei Symbol, daß es in der Symboltabelle definiert werden muß.

A.2 Übersicht Terminale

Definition

Ein Terminal ist ein Grundelement, das nicht durch eine weitere Regel, sondern verbal erklärt wird. In den Syntaxdiagrammen finden Sie es als folgendes Symbol:



Das Terminal wird durch ein Oval oder Kreis dargestellt. Entweder wird der Inhalt direkt angegeben oder der Name (in Großbuchstaben).

Dieser definiert die Menge der einsetzbaren ASCII-Zeichen.

Bild A-5 *Symbole für Terminale*

Übersicht

In den Kapitel A.3 bis A.4 werden die Verwendungsarten von einzelnen Zeichen dargestellt. Es sind:

- Buchstaben, Ziffern, druckbare Zeichen und Spezialzeichen
- Formatierungs- und Trennzeichen in den lexikalischen Regeln
- Präfixe für Literale
- Formatierungs- und Trennzeichen in den syntaktischen Regeln
- Operatoren

In den Kapitel A.5 und A.6 geht es um Schlüsselwörter und vordefinierte Bezeichner aus Zeichenketten. Die Tabellen sind alphabetisch geordnet. Bei Unterschieden zwischen SIMATIC und IEC Mnemonik wird die entsprechende IEC Mnemonik dazu angegeben:

- Schlüsselwörter und vordefinierte Bezeichner
- Operandenkennzeichen und Bausteinschlüsselwörter

A.3 Terminale der lexikalischen Regeln

Übersicht

In den folgenden Tabellen sind die Terminale spezifiziert durch die Angabe der Menge der Zeichensatzelemente aus dem ASCII-Zeichensatz.

Buchstaben und Ziffern

Buchstaben und Ziffern sind die hauptsächlich verwendeten Zeichen. Der BEZEICHNER (siehe Kapitel A-3) besteht z. B. aus Buchstaben, Ziffern und Unterstrich.

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

Druckbare Zeichen und Spezialzeichen

Der vollständige, erweiterte ASCII Zeichensatz ist benutzbar in Strings, Kommentaren und Symbolen.

Tabelle A-2 Druckbare Zeichen und Spezialzeichen

Zeichen	Untergruppe	Zeichensatzelemente
druckbares Zeichen	abhängig vom verwendeten Zeichencode. Bei ASCII-Code z. B. ab Dezimaläquivalent 31, ohne DEL und ohne die folgenden Ersatzzeichen:	alle druckbaren Zeichen
Ersatzzeichen	Dollarzeichen	$
	Apostroph	’
Steuerzeichen	$P oder $p	Seitenumbruch (formfeed, page)
	$L oder $l	Zeilenumbruch (linefeed)
	$R oder $r	Wagenrücklauf (carriage return)
	$T oder $t	Tabulator
Ersatzdarstellung im Hexacode	$hh	beliebige Zeichen, aufnehmbar als Hexacode (hh)

A.4 Formatierungs-, Trennzeichen und Operatoren

In den lexikalischen Regeln

In Tabelle A-3 finden Sie die Verwendung einzelner Zeichen des ASCII-Zeichensatzes als Formatierungs- und Trennzeichen im Bereich der lexikalischen Regeln (siehe Anhang B).

Tabelle A-3 Formatierungs- und Trennzeichen in den lexikalischen Regeln

Zeichen	Beschreibung
:	Trennzeichen zwischen Stunden, Minuten und Sekunden Attribute
.	Trennzeichen für Absolute Adressierung Realzahl- und Zeitintervall-darstellung
’ ’	Zeichen und Zeichenkette
” ”	Einleitungszeichen für Symbol nach den Regeln der Symboltabelle
_ Unterstrich	Trennzeichen für Zahlenwerte in Literalen kann in BEZEICHNERN vorkommen
$	Fluchtsymbol zur Angabe von Steuerzeichen oder Ersatzzeichen
$> $<	Stringunterbrechung, falls der String nicht in eine Zeile paßt oder falls Kommentare eingefügt werden sollen.

Für Literale

In Tabelle A-4 finden Sie die Verwendung einzelner Zeichen und Zeichenketten für Literale im Bereich der lexikalischen Regeln (siehe Anhang B). Die Tabelle gilt für SIMATIC- und IEC-Mnemonik.

Tabelle A-4 Mnemonik für Literale, alphabetisch geordnet

Präfix	Kennzeichen für	Lexikalische Regel
2#	INTEGERLITERAL	Binärziffernfolge
8#	INTEGER LITERAL	Oktalziffernfolge
16#	INTEGER LITERAL	Hexadezimalziffernfolge
D#	Zeitangabe	DATUM
DATE#	Zeitangabe	DATUM
DATE_AND_TIME#	Zeitangabe	DATUM UND ZEIT
DT#	Zeitangabe	DATUM UND ZEIT
E	Trennzeichen für REALZAHL-LITERAL	Exponent
e	Trennzeichen für REALZAHL-LITERAL	Exponent
D	Trennzeichen für Zeitintervall (Day)	Tage (Regel: Stufendarstellung)
H	Trennzeichen für Zeitintervall (Hour)	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)
T#	Zeitangabe	ZEITDAUER
TIME#	Zeitangabe	ZEITDAUER
TIME_OF_DAY#	Zeitangabe	TAGESZEIT
TOD#	Zeitangabe	TAGESZEIT

In den syntaktischen Regeln

In Tabelle A-5 finden Sie die Verwendung einzelner Zeichen als Formatierungs- und Trennzeichen im Bereich der syntaktischen Regeln sowie bei Kommentar und Attributen (siehe Anhang B.2 und B.3).

Tabelle A-5 Formatierungs- und Trennzeichen in den syntaktischen Regeln

Zeichen	Beschreibung	Syntaktische Regel, Kommentar oder Attribut
:	Trennzeichen zur Typangabe, bei Anweisung nach Sprungmarke	Variablendeklaration, Instanzdeklaration, Funktion, Anweisungsteil, CASE-Anweisung
;	Abschluß einer Vereinbarung oder Anweisung	Variablendeklaration, Anweisungsteil, DB-Zuweisungsteil, Konstantenblock, Sprungmarkenblock, Komponentendeklaration
,	Trennzeichen für Listen und Sprungmarkenblöcke	Variablendeklaration, Array-Datentyp Spezifikation, Feld-Initialisierungsliste, FB-Parameter, FC-Parameter, Wertliste, Instanzdeklaration
..	Bereichsangabe	Array-Datentyp Spezifikation, Wertliste
.	Trennzeichen für FB- und DB-Name, Absolute Adressierung	FB-Aufruf, strukturierte Variable
( )	Aufruf Funktion und Funktionsbaustein Klammerung in Ausdrücken, Initialisierungsliste für Arrays	Funktionsaufruf, FB-Aufruf, Ausdruck, Feld-Initialisierungsliste, Einfache Multiplikation, Potenzausdruck
[ ]	Array-Vereinbarung, strukturierte Variable Teil Array, Indizierung bei globalen Variablen und Strings	Array-Datentyp Spezifikation, STRING-Datentypspezifikation
(* *)	Blockkommentar	siehe Anhang B
//	Zeilenkommentar	siehe Anhang B
{ }	Attributblock	Angabe von Attributen
%	Einleitung für Direktbezeichner	Um IEC–konform zu programmieren, kann %M4.0 anstatt M4.0 verwendet werden.

Operatoren

In Tabelle A-6 sind alle SCL Operatoren dargestellt, Schlüsselworte, z. B. AND und die üblichen Operatoren als einzelne Zeichen. Diese Tabelle gilt für SIMATIC- und IEC-Mnemonik.

Tabelle A-6 SCL Operatoren

Operator	Beschreibung	Syntaktische Regel
:=	Zuweisungsoperator, Anfangszuweisung, Datentypinitialisierung	Wertzuweisung, DB-Zuweisungsteil, Konstantenblock, Ausgangs-/Durchgangs-Zuweisung, Eingangs-Zuweisung, Durchgangs-Zuweisung
+, -	Arithmetische Operatoren: unäre Operatoren, Vorzeichen	Ausdruck, einfacher Ausdruck, Potenzausdruck
+, -, *, / MOD; DIV	Arithmetische Basisoperatoren	Arithmetischer Basisoperator, einfache Multiplikation
**	Arithmetische Operatoren: Potenzoperator	Ausdruck
NOT	Logische Operatoren: Negation	Ausdruck, Operand
AND, &, OR; XOR,	Logische Basisoperatoren	Logischer Basisoperator
<,>,<=,>=,=,<>	Vergleichsoperator	Vergleichsoperator

A.5 Schlüsselwörter und vordefinierte Bezeichner

Schlüsselwörter und Vordefinierte Bezeichner

In Tabelle A-7 finden Sie Schlüsselwörter von SCL und vordefinierte Bezeichner alphabetisch aufgelistet. Dazu wird eine Beschreibung sowie die syntaktische Regel aus Anhang C angegeben, in der sie als Terminale verwendet werden. Schlüsselwörter sind generell unabhängig von der Mnemonik.

Tabelle A-7 SCL Schlüsselwörter und vordefinierte Bezeichner, alphabetisch geordnet

Schlüsselwörter	Beschreibung	Syntaktische Regel
AND	Logischer Operator	Logischer Basisoperator
ANY	Bezeichnung für ANY Datentyp	Parameter-Datentyp Spezifikation
ARRAY	Einleitung der Spezifikation eines Arrays, danach folgt zwischen ”[” und ”]” die Indexliste	Array-Datentyp Spezifikation
BEGIN	Einleitung Anweisungsteil bei Codebausteinen oder Initialisierungsteil bei Datenbaustein	Organisationsbaustein, Funktion, Funktionsbaustein, Datenbaustein
BLOCK_DB	Bezeichnung für Datentyp BLOCK_DB	Parameter-Datentyp Spezifikation
BLOCK_FB	Bezeichnung für Datentyp BLOCK_FB	Parameter-Datentyp Spezifikation
BLOCK_FC	Bezeichnung für Datentyp BLOCK_FC	Parameter-Datentyp Spezifikation
BLOCK_SDB	Bezeichnung für Datentyp BLOCK_SDB	Parameter-Datentyp Spezifikation
BOOL	Elementarer Datentyp für binäre Daten	Bitdatentyp
BY	Einleitung der Schrittweite	FOR-Anweisung
BYTE	Elementarer Datentyp	Bitdatentyp
CASE	Einleitung Kontrollanweisung zur Selektion	CASE-Anweisung
CHAR	Elementarer Datentyp	Zeichentyp
CONST	Einleitung für Definition von Konstanten	Konstantenblock
CONTINUE	Steueranweisung für FOR-, WHILE und REPEAT-Schleife	CONTINUE-Anweisung
COUNTER	Datentyp für Zähler, nur verwendbar in Parameterblock	Parameter-Datentyp Spezifikation
DATA_BLOCK	Einleitung des Datenbausteins	Datenbaustein
DATE	Elementarer Datentyp für Datum	Zeittyp
DATE_AND_TIME	Zusammengesetzter Datentyp für Datum und Uhrzeit	DATE_AND_TIME
DINT	Elementarer Datentyp für Ganzzahl (Integer) doppelter Genauigkeit	Numerischer Datentyp
DIV	Operator für Division	Arithmetischer Basisoperator, einfache Multiplikation
DO	Einleitung des Anweisungsteils bei FOR-Anweisung	FOR-Anweisung, WHILE-Anweisung
DT	Elementarer Datentyp für Datum und Uhrzeit	DATE_AND_TIME
DWORD	Elementarer Datentyp Doppelwort	Bitdatentyp
ELSE	Einleitung des Falls, wenn keine Bedingung erfüllt ist	IF-Anweisung, CASE-Anweisung
ELSIF	Einleitung der Alternativ-Bedingung	IF-Anweisung
EN	Flag zur Bausteinfreigabe	

Tabelle A-7 SCL Schlüsselwörter und vordefinierte Bezeichner, alphabetisch geordnet

Schlüsselwörter	Beschreibung	Syntaktische Regel
ENO	Fehlerflag des Bausteins	
END_CASE	Abschluß der CASE-Anweisung	CASE-Anweisung
END_CONST	Abschluß bei Definition von Konstanten	Konstantenblock
END_DATA_BLOCK	Abschluß des Datenbausteins	Datenbaustein
END_FOR	Abschluß der FOR-Anweisung	FOR-Anweisung
END_FUNCTION	Abschluß der Funktion	Funktion
END_FUNC-TION_BLOCK	Abschluß des Funktionsbausteins	Funktionsbaustein
END_IF	Abschluß der IF-Anweisung	IF-Anweisung
END_LABEL	Abschluß des Sprungmarkenblocks	Sprungmarkenblock
END_TYPE	Abschluß des UDT	Anwenderdefinierter Datentyp
END_ORGANIZA-TION_BLOCK	Abschluß des Organisationsbausteins	Organisationsbaustein
END_REPEAT	Abschluß der REPEAT-Anweisung	REPEAT-Anweisung
END_STRUCT	Abschluß der Spezifikation einer Struktur	Struktur-Datentyp Spezifikation
END_VAR	Abschluß eines Deklarationsblocks	temporärer Variablenblock, statischer Variablenblock, Parameterblock
END_WHILE	Abschluß der WHILE-Anweisung	WHILE-Anweisung
EXIT	Direkter Ausstieg aus der Schleifenbearbeitung	EXIT
FALSE	Vordefinierte boolesche Konstante: Logische Bedingung nicht erfüllt, Wert gleich 0	
FOR	Einleitung der Kontrollanweisung zur Schleifenbearbeitung	FOR-Anweisung
FUNCTION	Einleitung der Funktion	Funktion
FUNCTION_BLOCK	Einleitung des Funktionsbausteins	Funktionsbaustein
GOTO	Anweisung zur Durchführung eines Sprungs zu einer Sprungmarke	Programmsprung
IF	Einleitung Kontrollanweisung für Selektion	IF-Anweisung
INT	Elementarer Datentyp für Ganzzahl (Integer) einfacher Genauigkeit	Numerischer Datentyp
LABEL	Einleitung des Sprungmarkenblocks	Sprungmarkenblock
MOD	Arithmetischer Operator für Divisionsrest	Arithmetischer Basisoperator, einfache Multiplikation
NIL	Nullpointer	
NOT	Logischer Operator, gehört zu den Unären Operatoren	Ausdruck, Operand
OF	Einleitung der Datentypspezifikation	Array-Datentyp-Spezifikation, CASE-Anweisung
OK	Flag, das aussagt, ob die Anweisungen eines Bausteins fehlerfrei abgearbeitet wurden	
OR	Logischer Operator	Logischer Basisoperator
ORGANIZATION_ BLOCK	Einleitung des Organisationsbausteins	Organisationsbaustein

Tabelle A-7 SCL Schlüsselwörter und vordefinierte Bezeichner, alphabetisch geordnet

Schlüsselwörter	Beschreibung	Syntaktische Regel
POINTER	Zeiger-Datentyp, nur erlaubt in Parameterdeklaration im Parameterblock, wird nicht in SCL bearbeitet	siehe Kapitel 10
REAL	Elementarer Datentyp	Numerischer Datentyp
REPEAT	Einleitung der Kontrollanweisung zur Schleifenbearbeitung	REPEAT-Anweisung
RETURN	Steueranweisung zur Rückkehr von Unterprogramm	RETURN-Anweisung
S5TIME	Elementarer Datentyp für Zeitangaben, spezielles S5-Format	Zeittyp
STRING	Datentyp für Zeichenkette	STRING-Datentyp Spezifikation
STRUCT	Einleitung der Spezifikation einer Struktur, danach folgt Liste der Komponenten	STRUCT-Datentyp Spezifikation
THEN	Einleitung der Folgeaktionen, wenn Bedingung erfüllt	IF-Anweisung
TIME	Elementarer Datentyp für Zeitangaben	Zeittyp
TIMER	Datentyp für Zeitglied, nur verwendbar in Parameterblock	Parameter-Datentyp Spezifikation
TIME_OF_DAY	Elementarer Datentyp für Tageszeit	Zeittyp
TO	Einleitung des Endwerts	FOR-Anweisung
TOD	Elementarer Datentyp für Tageszeit	Zeittyp
TRUE	Vordefinierte Boolesche Konstante: Logische Bedingung erfüllt, Wert ungleich 0	
TYPE	Einleitung des UDT	Anwenderdefinierter Datentyp
VAR	Einleitung eines Deklarationsblocks	statischer Variablenblock
VAR_TEMP	Einleitung eines Deklarationsblocks	temporärer Variablenblock
UNTIL	Einleitung der Abbruchbedingung für REPEAT-Anweisung	REPEAT-Anweisung
VAR_INPUT	Einleitung eines Deklarationsblocks	Parameterblock
VAR_IN_OUT	Einleitung eines Deklarationsblocks	Parameterblock
VAR_OUTPUT	Einleitung eines Deklarationsblocks	Parameterblock
WHILE	Einleitung der Kontrollanweisung zur Schleifenbearbeitung	WHILE-Anweisung
WORD	Elementarer Datentyp Wort	Bitdatentyp
VOID	Kein Rückgabewert bei einem Funktionsaufruf	Funktion
XOR	Logischer Operator	Logischer Basisoperator

A.6 Operandenkennzeichen und Bausteinschlüsselwörter

Globale Systemdaten

In Tabelle A-8 finden Sie die SIMATIC-Mnemonik der SCL Operandenkennzeichen mit Beschreibung alphabetisch aufgelistet:

- Angabe des Operandenkennzeichens:
  Speicher-Präfix (A, E, M, PA, PE) oder Datenbaustein (D)
- Angabe der Größe des Datenelements:
  Größen-Präfix (optional oder B, D, W, X)

Die Mnemonik stellt eine Kombination zwischen dem Operandenkennzeichen (Speicher-Präfix oder D für Datenbaustein) und Größen-Präfix dar. Beides sind lexikalische Regeln. Die Tabelle ist nach der SIMATIC- Mnemonik sortiert, die entsprechende IEC- Mnemonik wird dazu angegeben.

Tabelle A-8 Operandenkennzeichen der globalen Systemdaten

SIMATIC-Mnemonik	IEC-Mnemonik	Speicher-Präfix oder Datenbaustein	Größen-Präfix
A	Q	Ausgang (über Prozeßabbild)	Bit
AB	QB	Ausgang (über Prozeßabbild)	Byte
AD	QD	Ausgang (über Prozeßabbild)	Doppelwort
AW	QW	Ausgang (über Prozeßabbild)	Wort
AX	QX	Ausgang (über Prozeßabbild)	Bit
D	D	Datenbaustein	Bit
DB	DB	Datenbaustein	Byte
DD	DD	Datenbaustein	Doppelwort
DW	DW	Datenbaustein	Wort
DX	DX	Datenbaustein	Bit
E	I	Eingang (über Prozeßabbild)	Bit
EB	IB	Eingang (über Prozeßabbild)	Byte
ED	ID	Eingang (über Prozeßabbild)	Doppelwort
EW	IW	Eingang (über Prozeßabbild)	Wort
EX	IX	Eingang (über Prozeßabbild)	Bit
M	M	Merker	Bit
MB	MB	Merker	Byte
MD	MD	Merker	Doppelwort
MW	MW	Merker	Wort
MX	MX	Merker	Bit
PAB	PQB	Ausgang (Peripherie direkt)	Byte
PAD	PQD	Ausgang (Peripherie direkt)	Doppelwort
PAW	PQW	Ausgang (Peripherie direkt)	Wort
PEB	PIB	Eingang (Peripherie direkt)	Byte
PED	PID	Eingang (Peripherie direkt)	Doppelwort
PEW	PIW	Eingang (Peripherie direkt)	Wort

Baustein-schlüsselwörter

Werden für die absolute Adressierung von Bausteinen verwendet. Die Tabelle ist nach der SIMATIC Mnemonik sortiert, die entsprechende IEC-Mnemonik wird dazu angegeben.

Tabelle A-9 Bausteinschlüsselwörter sowie Zähler und Zeitglied

SIMATIC-Mnemonik	IEC-Mnemonik	Speicher-Präfix oder Datenbaustein
DB	DB	Datenbaustein (Data_Block)
FB	FB	Funktionsbaustein (Function_Block)
FC	FC	Funktion (Function)
OB	OB	Organisationsbaustein (Organization_Block)
SDB	SDB	Systemdatenbaustein (System_Data_Block)
SFC	SFC	Systemfunktion (System_Function)
SFB	SFB	Systemfunktionsbaustein (System_Function_Block)
T	T	Zeitglied (Timer)
UDT	UDT	globaler bzw. anwenderdefinierter Datentyp (Userdefined_Data_Type)
Z	C	Zähler (Counter)

A.7 Übersicht Non-Terminale

Definition

Ein Non-Terminal ist ein zusammengesetztes Element, das durch eine weitere Regel beschrieben wird. Das Non-Terminal wird durch einen Kasten dargestellt. Der Name im Kasten entspricht dem Regelnamen der weiterführenden Regel.

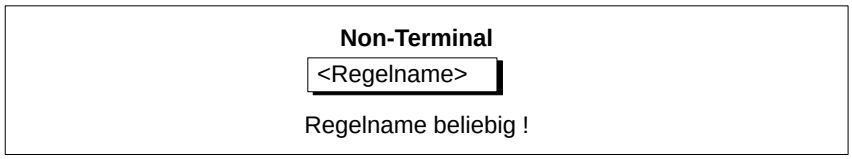


Bild A-6 Non-Terminal

Das Element kommt in den lexikalischen und den syntaktischen Regeln vor.

A.8 Übersicht Token

Definition

Ein Token ist ein zusammengesetztes Element, das in den syntaktischen Regeln als Grundelement verwendet wird und in den lexikalischen Regeln erklärt wird. Das Token wird durch ein Rechteck dargestellt. Der NAME, in Großbuchstaben, entspricht dem Regelnamen der weiterführenden lexikalischen Regeln (ohne Rechteck).

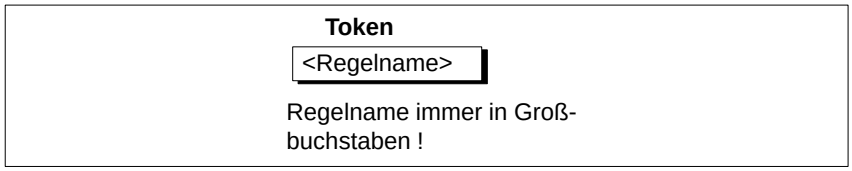


Bild A-7 Token

Übersicht

Die definierten Token stellen Bezeichner dar, die als Ergebnisse der lexikalischen Regeln ermittelt wurden. Diese Token beschreiben die:

- Bezeichner
- Namensvergabe bei SCL
- Vordefinierten Konstanten und Flags

A.9 Bezeichner

Bezeichner in SCL

Mit Bezeichnern können Sie Sprachobjekte von SCL ansprechen. Tabelle A-10 informiert Sie über die Klassen von Bezeichnern.

Tabelle A-10 Gesamtmenge der Bezeichnerarten in SCL

Bezeichnerart	Bemerkungen, Beispiele
Schlüsselwörter	z. B. Steueranweisungen `BEGIN,DO,WHILE`
Vordefinierte Namen	Namen von • Standard-Datentypen (z. B. BOOL, BYTE, INT) • vordefinierte Standardfunktionen z. B. `ABS` • Standardkonstanten `TRUE und FALSE`
Operandenkennzeichen bei Absolutbezeichnern	für globale Systemdaten und Datenbausteine: z. B. `E1.2, MW10, FC20, T5, DB30, DB10.D4.5`
freie wählbare Namen nach der Regel BEZEICHNER	Namen von • vereinbarten Variablen • Strukturkomponenten • Parametern • vereinbarten Konstanten • Sprungmarken
Symbole der Symboltabelle	erfüllen entweder die lexikalische Regel BEZEICHNER oder die lexikalische Regel Symbol, d.h. in Hochkommata eingeschlossen, z. B. ”xyz”

Groß- und Kleinschreibung

Bei den Schlüsselwörtern ist Groß- und Kleinschreibung nicht relevant. Seit der S7-SCL Version 4.0 wird auch bei den vordefinierten Namen, sowie bei den frei wählbaren Namen, z. B. für Variablen, und bei den Symbolen aus der Symboltabelle Groß- und Kleinschreibung nicht mehr unterschieden. Tabelle A-11 gibt Ihnen einen Überblick.

Tabelle A-11 Relevanz von Groß- und Kleinschreibung bei den Bezeichnerarten

Bezeichnerart	case-sensitiv?
Schlüsselwörter	nein
vordefinierte Namen bei Standard-Datentypen	nein
Namen bei vordefinierten Standardfunktionen	nein
vordefinierte Namen bei Standardkonstanten	nein
Operandenkennzeichen bei Absolutbezeichnern	nein
freie Namen	nein
Symbole der Symboltabelle	nein

Die Namen für Standardfunktionen, z. B. BYTE_TO_WORD und ABS können also auch in Kleinbuchstaben geschrieben werden. Genauso die Parameter für Zeit- und Zählfunktionen, z. B. SV, se oder ZV.

A.10 Namensvergabe bei SCL

Vergabe von wählbaren Namen

Für die Vergabe von Namen haben Sie generell 2 Möglichkeiten:

- Sie können Namen innerhalb SCL selbst vergeben. Diese Namen müssen der Regel BEZEICHNER entsprechen (siehe Bild A-8). Die Regel BEZEICHNER können Sie für jeden Namen in SCL benutzen.
- Sie können Namen über STEP 7 mit Hilfe der Symboltabelle einführen. Die Regel für diese Namen ist ebenfalls BEZEICHNER oder als erweiterte Möglichkeit Symbol. Durch die Angabe in Hochkommata kann das Symbol mit allen druckbaren Zeichen (z.B. Leerzeichen) gebildet werden.

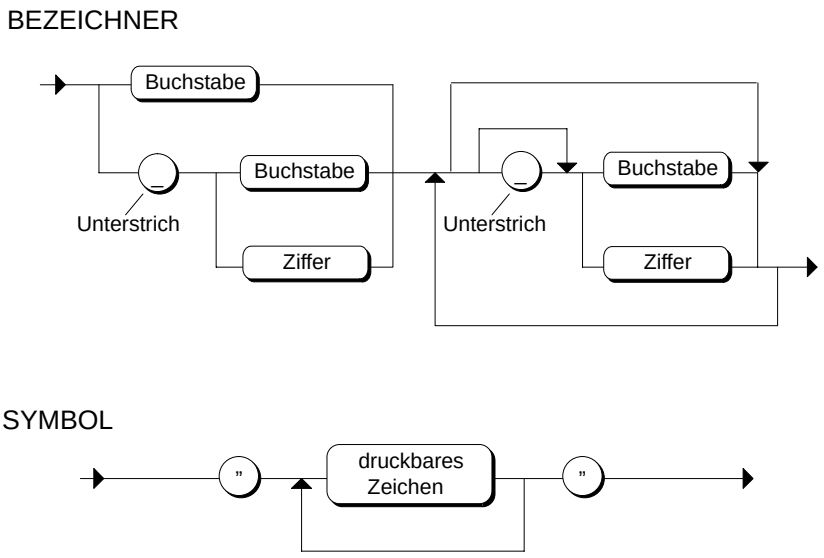


Bild A-8 *Lexikalische Regeln: BEZEICHNER und* Symbol

Regeln bei der Namensvergabe

Beachten Sie bitte folgendes:

- Bei der Namensvergabe wählen Sie am besten eindeutige und aussagekräftige Namen, die zur Verständlichkeit des Programms beitragen.
- Achten Sie darauf, ob der Name schon vom System belegt ist, z. B. durch Bezeichner für Datentypen oder Standardfunktionen.
- Gültigkeitsbereich: Bei Namen, die global gültig sind, erstreckt sich der Gültigkeitsbereich über das gesamte Programm. Lokal gültige Namen gelten nur innerhalb eines Bausteins. Sie haben somit die Möglichkeit gleiche Namen in verschiedenen Bausteinen zu benutzen. Die Tabelle A-12 informiert Sie über die Möglichkeiten, die Sie haben.

Namens-einschränkungen

Bei der Vergabe von Namen müssen Sie einige Einschränkungen beachten.

Die Namen müssen in ihrem Gültigkeitsbereich eindeutig sein, d.h. Namen, die bereits innerhalb eines Bausteins vergeben wurden, dürfen nicht nochmals im selben Baustein benutzt werden. Weiterhin dürfen folgende vom System belegte Namen nicht benutzt werden:

- Namen von Schlüsselwörtern: z. B. CONST, END_CONST, BEGIN
- Namen von Operatoren: z. B. AND, XOR
- Namen von Vordefinierten Bezeichnern: z. B. Namen für Datentypen wie BOOL, STRING, INT
- Namen der vordefinierten Konstanten TRUE und FALSE
- Namen von Standardfunktionen: z. B. ABS, ACOS, ASIN, COS, LN
- Namen von Absolut- bzw. Operandenkennzeichen für globale Systemdaten: z. B. EB, EW, ED, AB, AW, AD MB, MD

Verwendung von BEZEICHNER

Tabelle A-12 zeigt Ihnen für welche Fälle Sie Namen, die der Regel für BEZEICHNER entsprechen, angeben können.

Tabelle A-12 Vorkommen von BEZEICHNER

BEZEICHNER	Beschreibung	Regel
Bausteinname	Symbolischer Name für Baustein	BAUSTEIN-BEZEICHNUNG, Funktionsaufruf
Name für Zeitglied und Zähler	Symbolischer Name für Zeitglied und Zähler	TIMER-BEZEICHNUNG, ZÄHLER-BEZEICHNUNG
Attributname	Name für ein Attribut	Attributzuweisung
Konstantenname	Vereinbarung symbolischer Konstante, Verwendung	Konstantenblock, Konstante
Sprungmarke	Vereinbarung Sprungmarke, Verwendung Sprungmarke	Sprungmarkenblock-Anweisungsteil, GOTO-Anweisung
Variablenname	Vereinbarung temporärer oder statischer Variable	Variablendeklaration, Einfache Variable, Strukturierte Variable
Lokaler Instanzname	Vereinbarung lokaler Instanzen	Instanzdeklaration, FB-Aufrufname

BAUSTEIN-BEZEICHNUNG

In der Regel BAUSTEIN-BEZEICHNUNG können Sie BEZEICHNER und Symbol alternativ einsetzen:

BAUSTEIN-BEZEICHNUNG

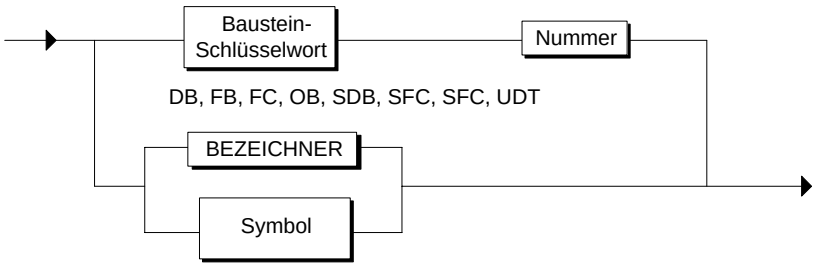


Bild A-9 *Lexikalische Regel: BAUSTEIN-BEZEICHNUNG*

Analog zu BAUSTEIN-BEZEICHNUNG gelten auch die Regeln TIMER-BEZEICHNUNG und COUNTER-BEZEICHNUNG.

A.11 Vordefinierte Konstanten und Flags

Vordefinierte Konstanten und Flags

Beide Tabellen gelten für SIMATIC- und IEC-Mnemonik.

Tabelle A-13 Vordefinierte Konstanten

Mnemonik	Beschreibung
FALSE	Vordefinierte boolesche Konstante (Standardkonstante) mit dem Wert 0. Sie hat die logische Bedeutung, daß eine Bedingung nicht erfüllt ist.
TRUE	Vordefinierte boolesche Konstante (Standardkonstante) mit dem Wert 1. Sie hat die logische Bedeutung, daß eine Bedingung erfüllt ist.

Tabelle A-14 Flags

Mnemonik	Beschreibung
EN	Flag zur Baustein-Freigabe
ENO	Fehlerflag des Bausteins
OK	Flag wird auf FALSE gesetzt, wenn eine Anweisung fehlerhaft bearbeitet wurde.

Lexikalische Regeln

B

Kapitelübersicht

Im Kapitel	finden Sie	auf Seite
B.1	Bezeichnungen	B-2
B.1.1	Literale	B-4
B.1.2	Absolutadressierung	B-9
B.2	Kommentare	B-11
B.3	Bausteinattribute	B-12

Lexikalische Regeln

Die lexikalischen Regeln beschreiben die Struktur der Elemente (Token), 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.

Einteilung

Die lexikalischen Regeln sind in folgende Gruppen unterteilt:

- Bezeichnungen
- Literale
- Absolutadressierung

B.1 Bezeichnungen

Tabelle B-1 Bezeichnungen

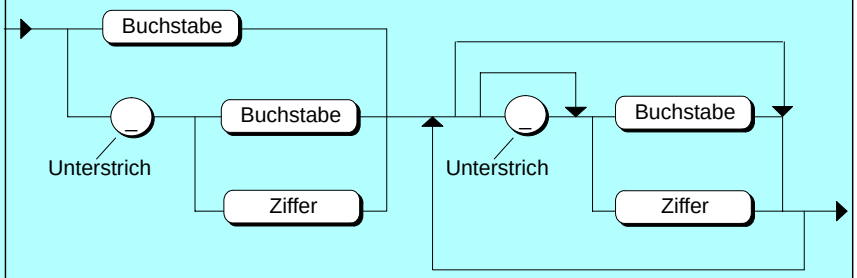
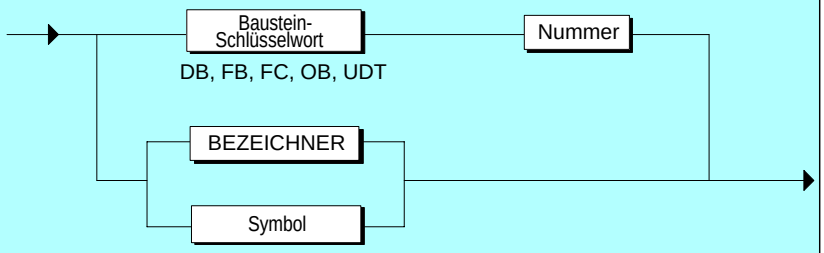
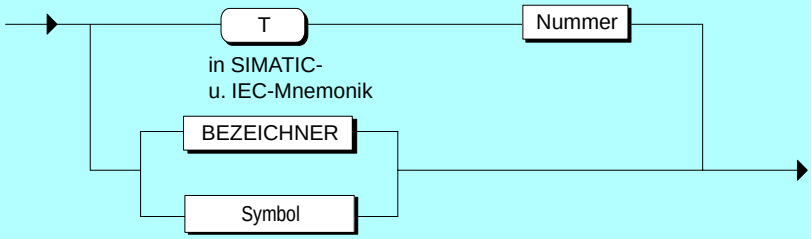
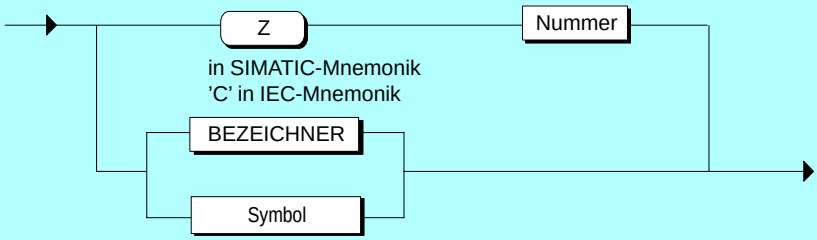
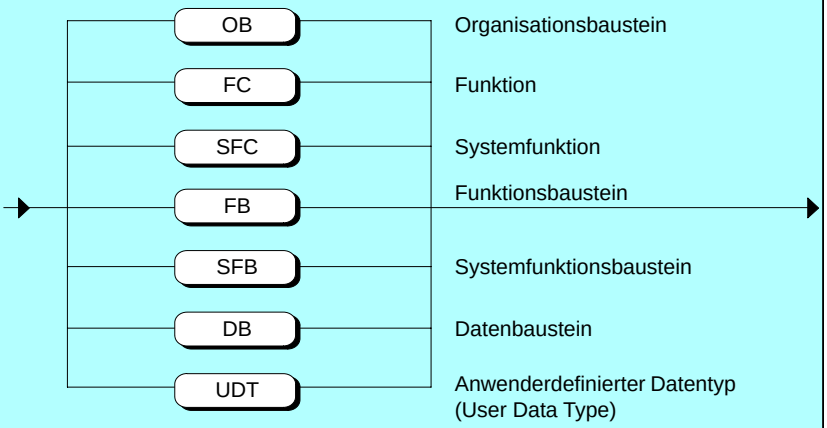
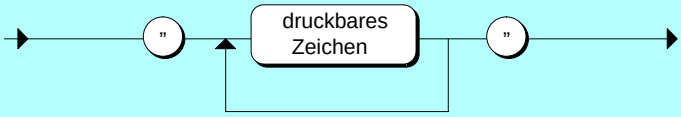
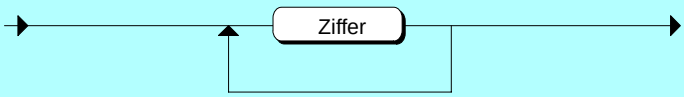
Regel	Syntaxdiagramm
BEZEICHNER	
BAUSTEIN-BEZEICHNUNG	Die Regel gilt auch für folgende Regelnamen: DB - BEZEICHNUNG FB - BEZEICHNUNG FC - BEZEICHNUNG OB - BEZEICHNUNG UDT - BEZEICHNUNG 
TIMERBEZEICH-NUNG	

Tabelle B-1 Bezeichnungen, Fortsetzung

Regel	Syntaxdiagramm
ZÄHLER-BEZEICHNUNG	
Baustein-Schlüssel-wort	
Symbol	
Nummer	

B.1.1 Literale

Tabelle B-2 Literale

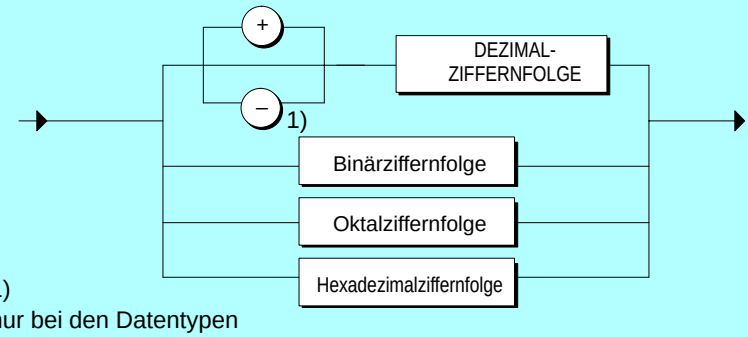
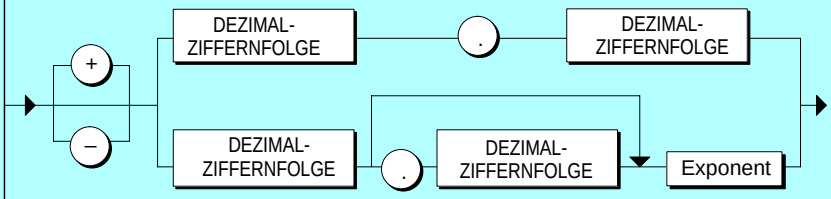
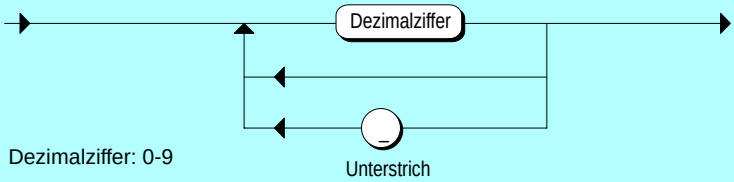
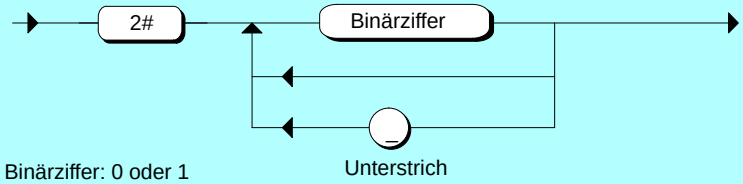
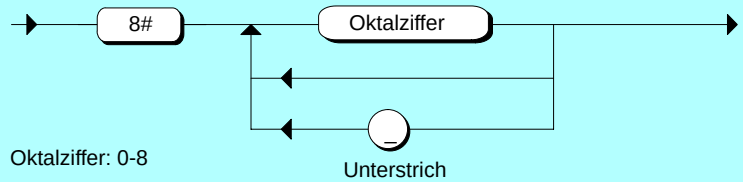
Regel	Syntaxdiagramm
INTEGER-LITERAL	 1) nur bei den Datentypen INT und DINT
REALZAHL- LITERAL	
DEZIMALZIFFERN- FOLGE	
Binärziffernfolge	
Oktalziffernfolge	

Tabelle B-2 Literale, Fortsetzung

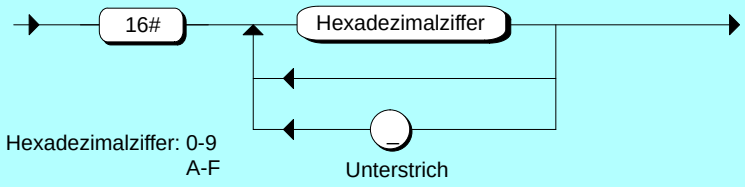
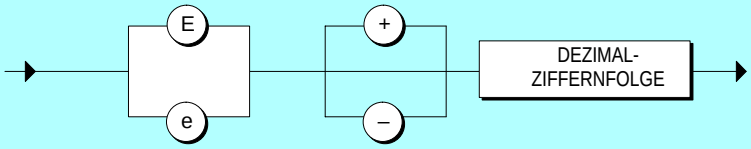
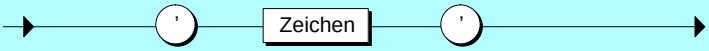
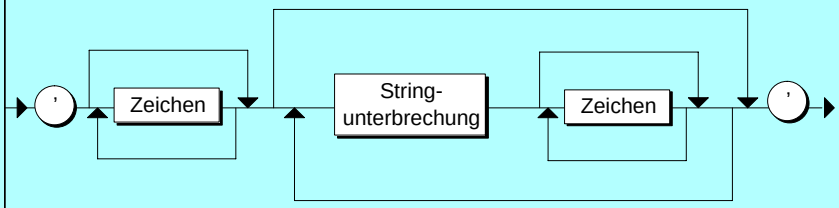
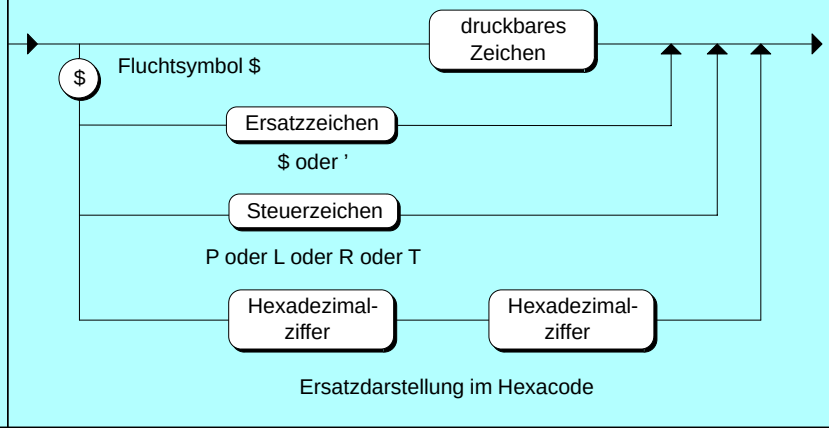
Regel	Syntaxdiagramm
Hexadezimal-ziffernfolge	
Exponent	
CHARACTER-LITERAL	
STRING-LITERAL	
Zeichen	

Tabelle B-2 Literale, Fortsetzung

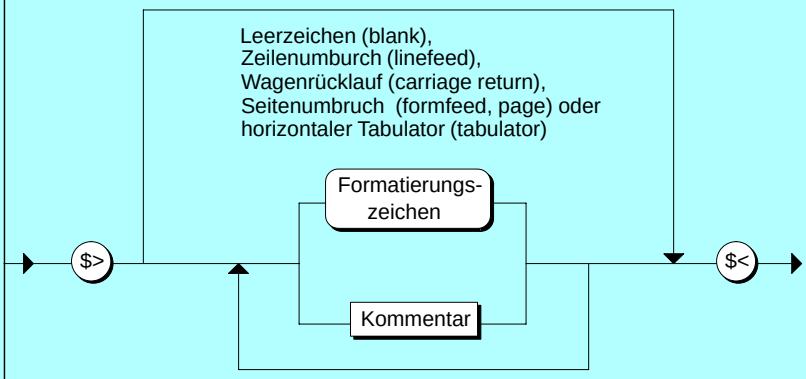
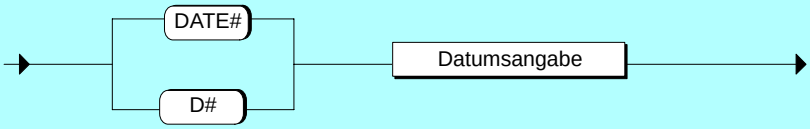
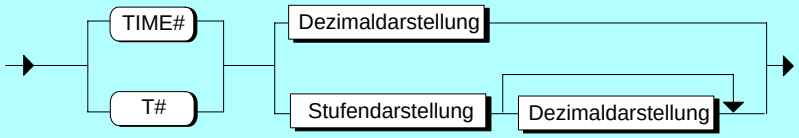
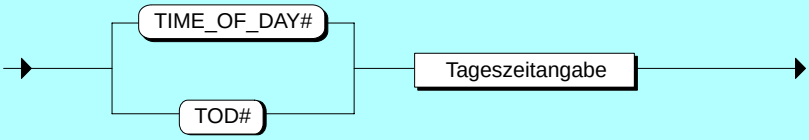
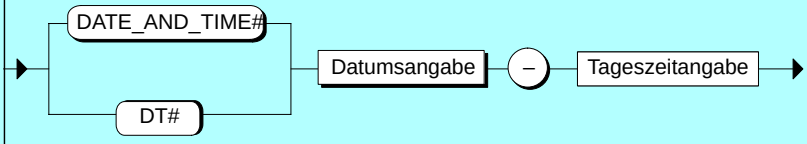
Regel	Syntaxdiagramm
Stringunterbrechung	
DATUM	
ZEITDAUER	 - Jede Zeiteinheit (z. B. Stunden, Minuten) darf nur 1 x angegeben werden. - Die Reihenfolge - Tage, Stunden, Minuten, Sekunden, Millisekunden - ist einzuhalten.
TAGESZEIT	
DATUM UND ZEIT	

Tabelle B-2 Literale, Fortsetzung

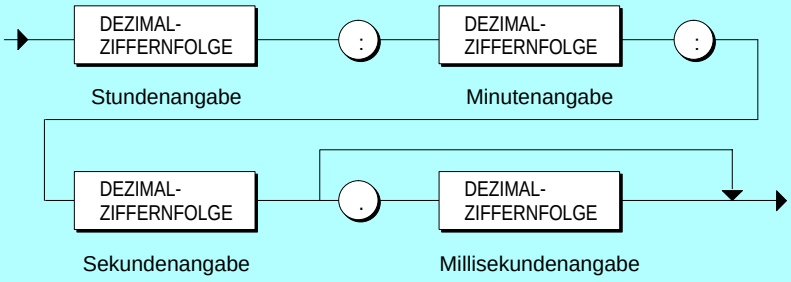
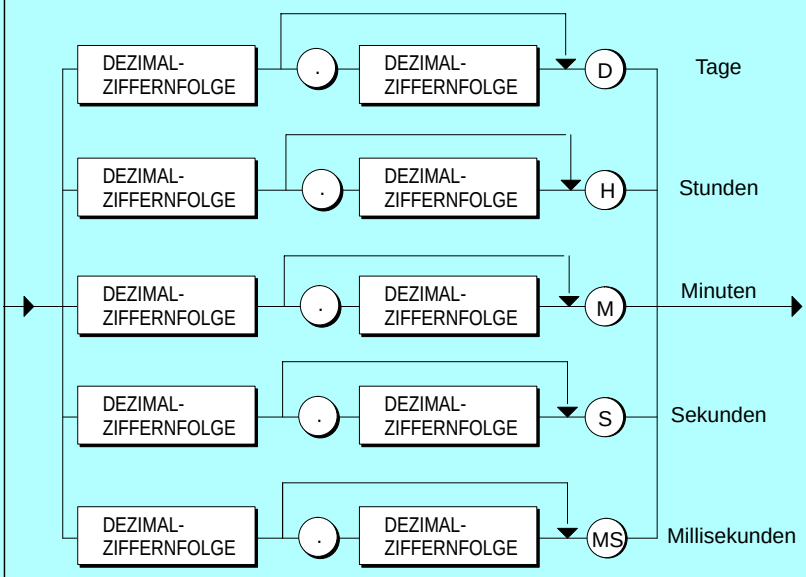
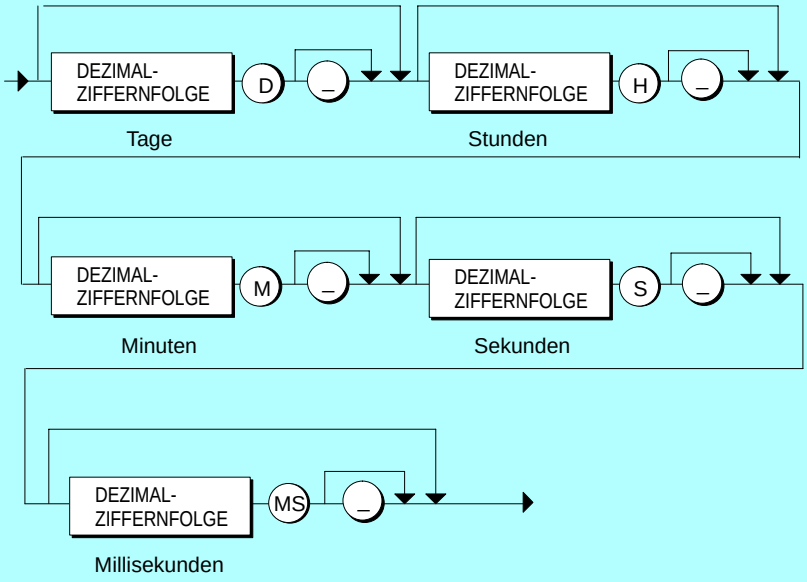
Regel	Syntaxdiagramm
Datumgsangabe	
Tageszeitangabe	

Tabelle B-2 Literale, Fortsetzung

Regel	Syntaxdiagramm
Dezimaldarstellung	 Der Einstieg in die Dezimaldarstellung ist nur bei noch nicht definierten Zeiteinheiten möglich.
Stufendarstellung	 mindestens eine Angabe ist erforderlich!

B.1.2 Absolutadressierung

Tabelle B-3 Absolutadressierung

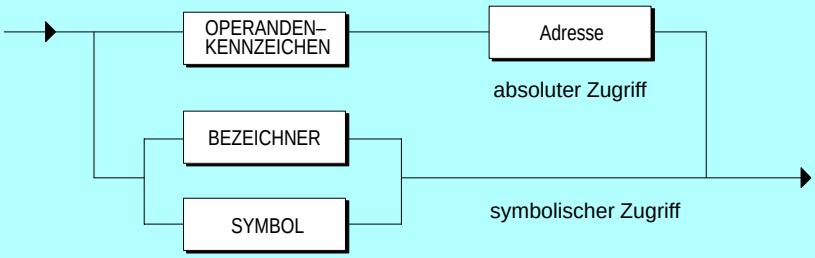
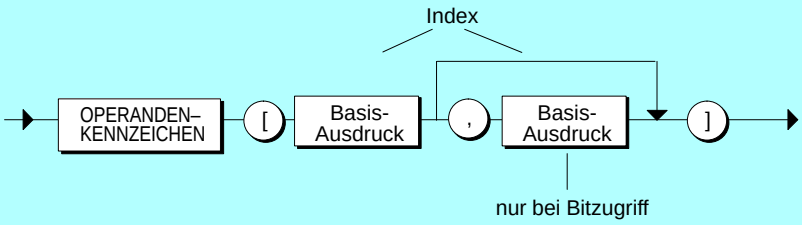
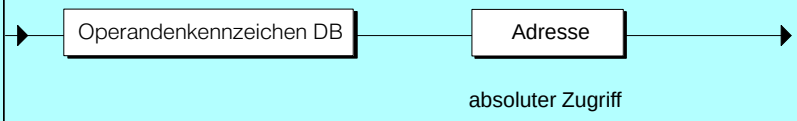
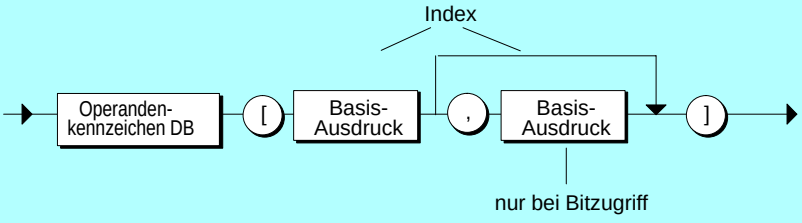
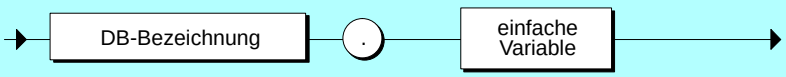
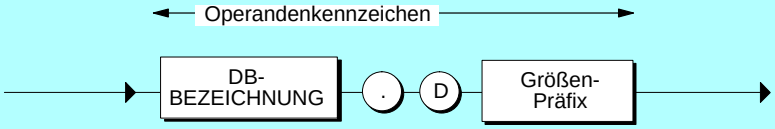
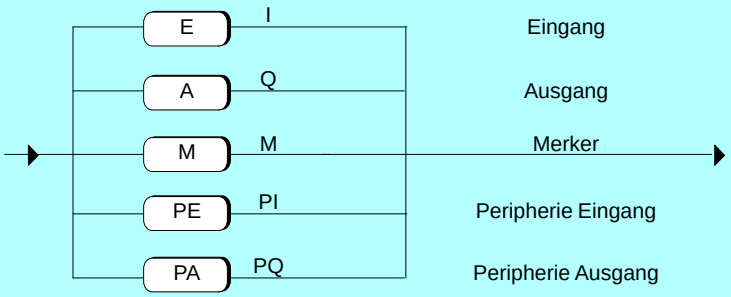
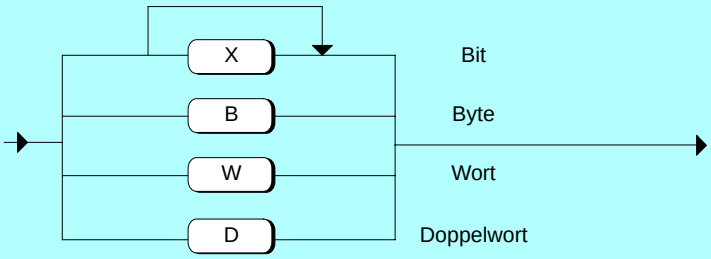
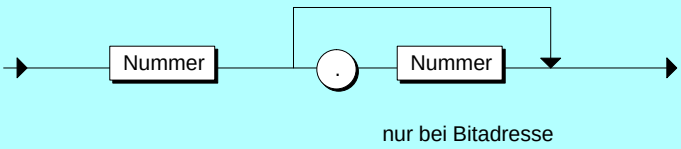
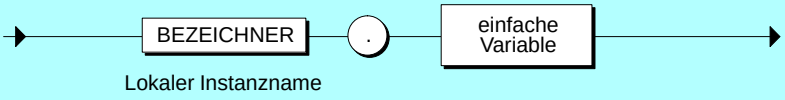
Regel	Syntaxdiagramm
EINFACHER SPEICHERZUGRIFF	
INDIZIERTER SPEICHERZUGRIFF	
OPERANDEN- KENNZEICHEN FÜR SPEICHER	
ABSOLUTER DB-ZUGRIFF	
INDIZIERTER DB-ZUGRIFF	
STRUKTURIERTER DB-ZUGRIFF	

Tabelle B-3 Absolutadressierung, Fortsetzung

Regel	Syntaxdiagramm
Operanden- Kennzeichen DB	
Speicher-Präfix	
Größen-Präfix für Speicher und DB	
Adresse für Speicher und DB	
Zugriff auf lokale Instanz	

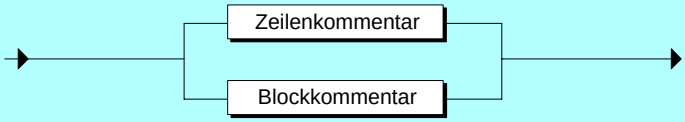
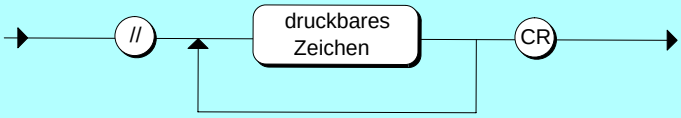
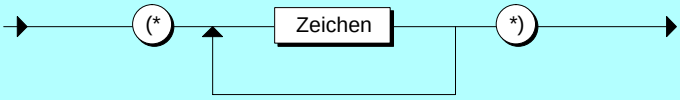
B.2 Kommentare

Was zu beachten ist

Folgende sind die wichtigsten Punkte, die beim Einbau von Kommentaren zu beachten sind:

- Die Schachtelung von Kommentaren ist nicht erlaubt.
- Der Einbau ist an beliebigen Stellen in den syntaktischen Regeln möglich, nicht aber in den lexikalischen Regeln.

Tabelle B-4 Kommentare

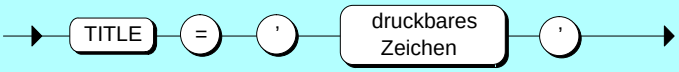
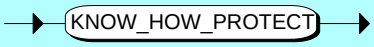
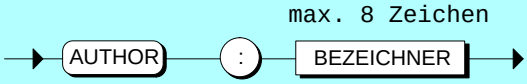
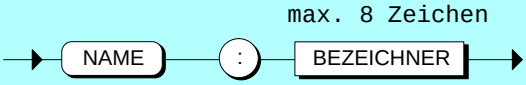
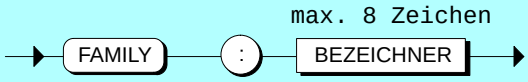
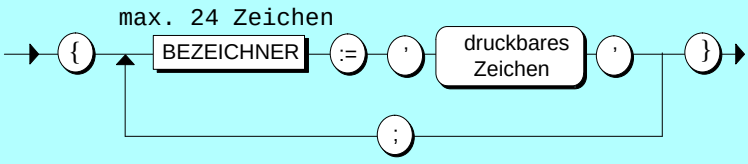
Regel	Syntaxdiagramm
KOMMENTAR	
ZEILENKOMMENTAR	
BLOCKKOMMENTAR	

B.3 Bausteinattribute

Was zu beachten ist

Bausteinattribute können mit folgender Syntax nach der BAUSTEIN-BEZEICHNUNG und vor der Vereinbarung des ersten Variablen- oder Parameterblocks stehen.

Tabelle B-5 Attribute

Regel	Syntaxdiagramm
ÜBERSCHRIFT	
VERSION	
BAUSTEINSCHUTZ	
AUTHOR	
NAME	
BAUSTEINFAMILIE	
Systemattribute für Bausteine	

Syntaktische Regeln

C

Definition der syntaktischen Regeln

Aufbauend auf den lexikalischen Regeln wird in den syntaktischen Regeln die Struktur von SCL beschrieben. Im Rahmen dieser Regeln können Sie Ihr SCL-Programm formatfrei erstellen.

Kapitelübersicht

Formales

Jede Regel hat einen Regelnamen, der vorangestellt ist. Wenn die Regel in einer übergeordneten Regel verwendet wird, so taucht der Regelname in einem Rechteck auf.

Ist der Name im Rechteck in Großbuchstaben geschrieben, so handelt es sich um ein Token, das in den lexikalischen Regeln beschrieben wird.

Über Regelnamen in abgerundeten Rahmen oder Kreisen, finden Sie Informationen im Anhang A.

Was zu beachten ist

Die Eigenschaft formatfrei bedeutet für Sie:

- Einfügen von Formatierungszeichen ist überall möglich
- Block- und Zeilenkommentare können eingefügt werden (siehe Kapitel 7.6)

C.1 Gliederungen von SCL-Quellen

Tabelle C-1 Syntax der SCL-Quellen

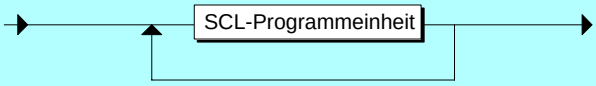
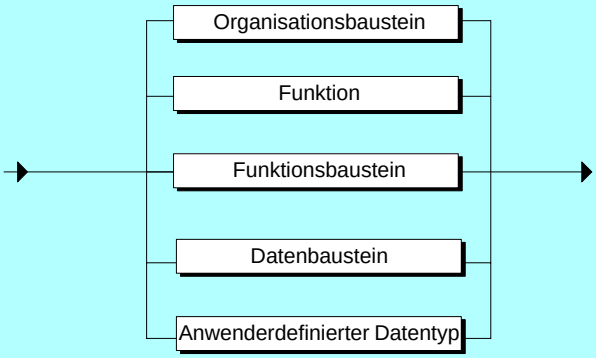
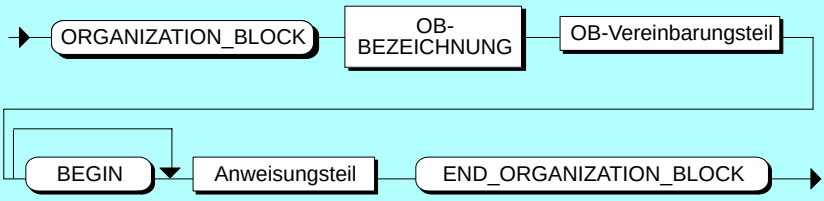
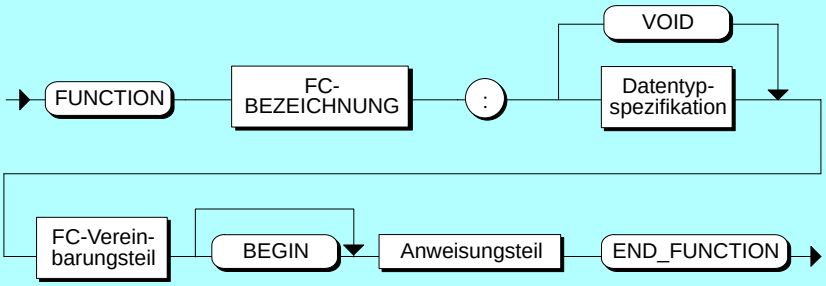
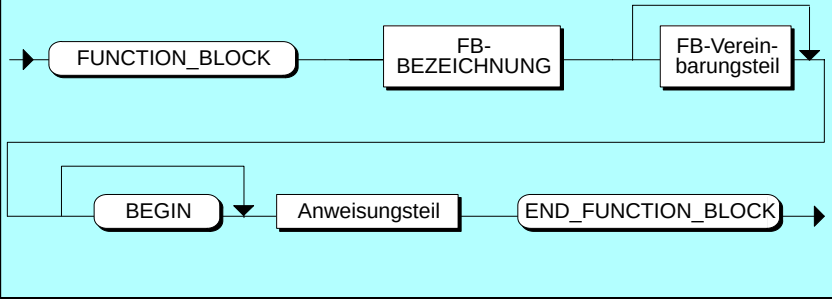
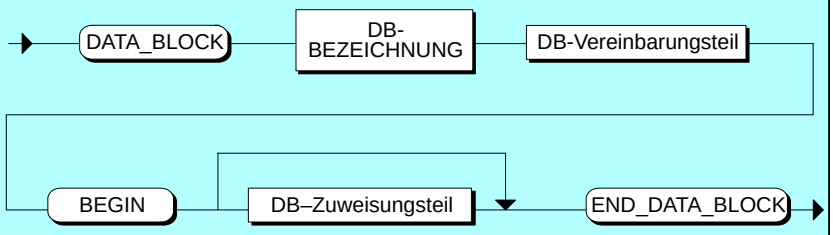
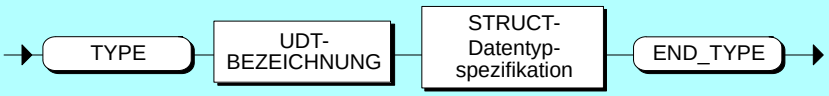
Regel	Syntaxdiagramm
SCL-Programm	
SCL-Programmeinheit	
Organisationsbaustein	
Funktion Beachten Sie, daß bei Funktionen ohne VOID im Anweisungsteil der Rückgabewert dem Funktionsnamen zugewiesen werden muß!	
Funktionsbaustein	

Tabelle C-1 Syntax der SCL-Quellen, Fortsetzung

Regel	Syntaxdiagramm
Datenbaustein	
Anwenderdefinierter Datentyp	

C.2 Aufbau der Vereinbarungsteile

Tabelle C-2 Syntax des Vereinbarungsteils

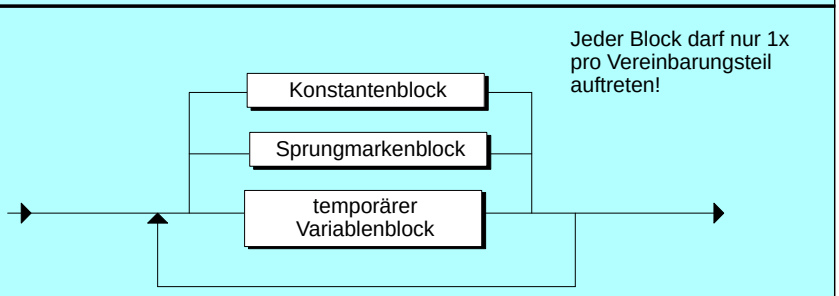
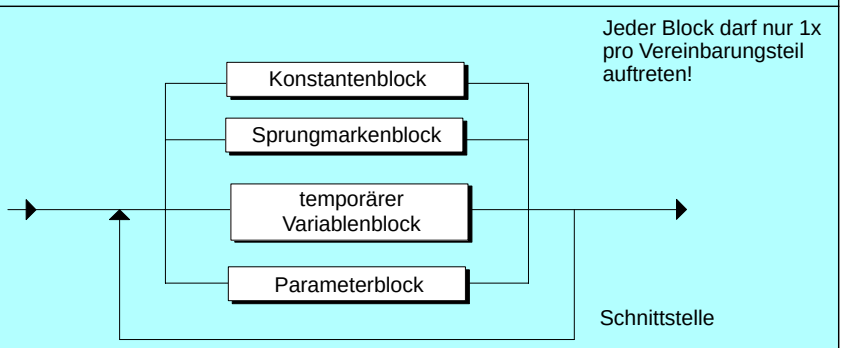
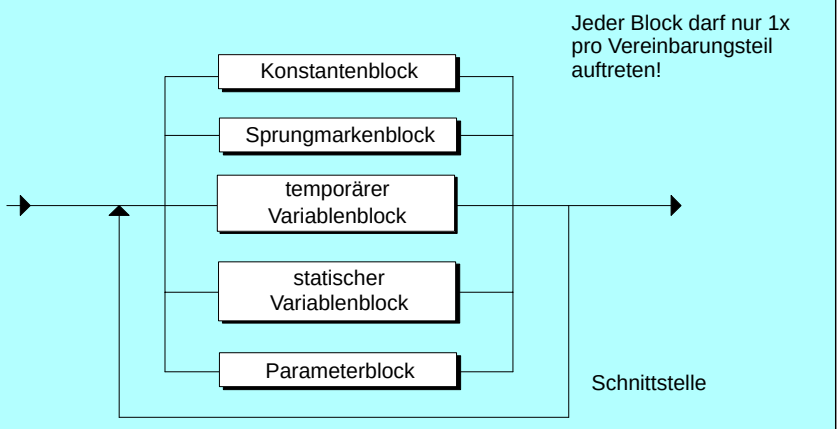
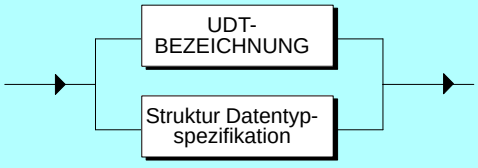
Regel	Syntaxdiagramm
OB-Vereinbarungsteil	
FC-Vereinbarungsteil	
FB-Vereinbarungsteil	
DB-Vereinbarungsteil	

Tabelle C-3 Syntax der Vereinbarungsblöcke

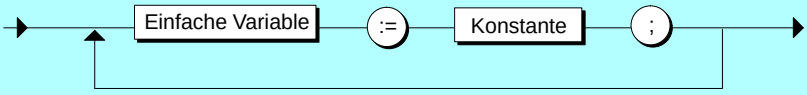
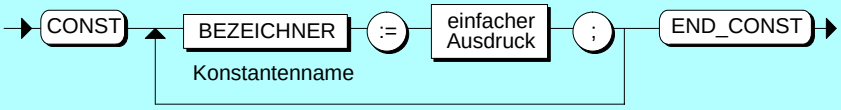
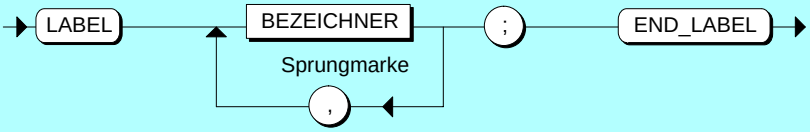
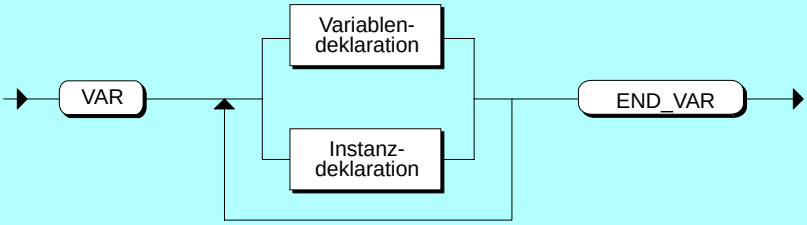
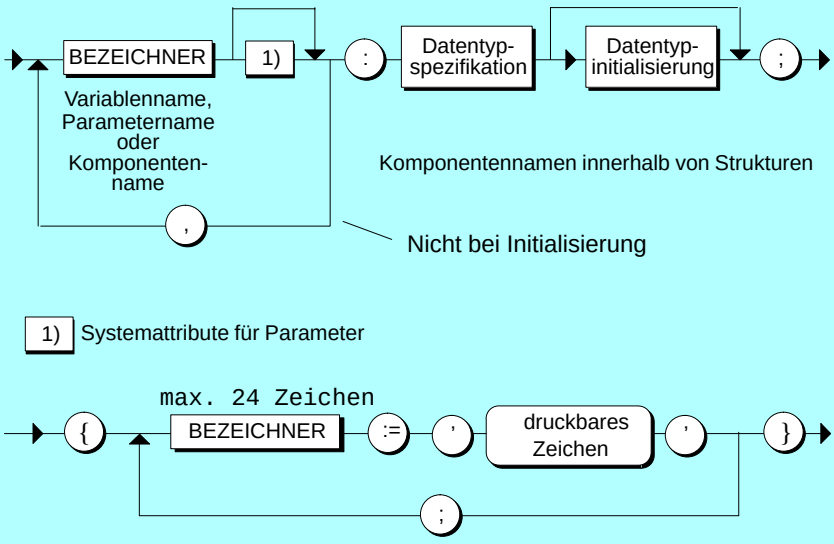
Regel	Syntaxdiagramm
DB-Zuweisungsteil	
Konstantenblock	
Sprungmarkenblock	
Statischer Variablenblock	
Variablendeklaration	

Tabelle C-3 Syntax der Vereinbarungsblöcke, Fortsetzung

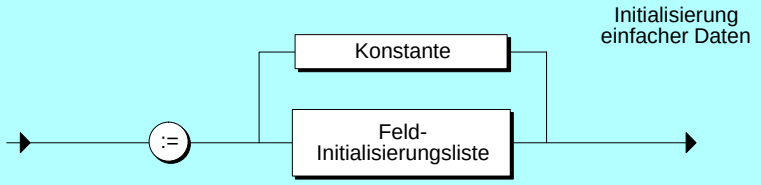
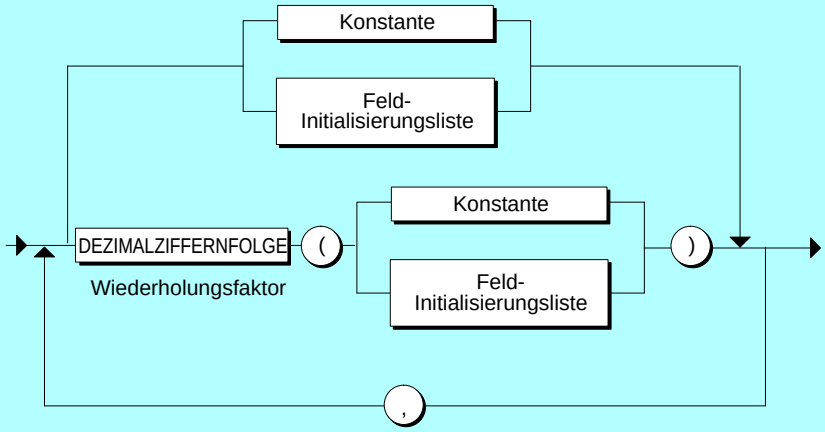
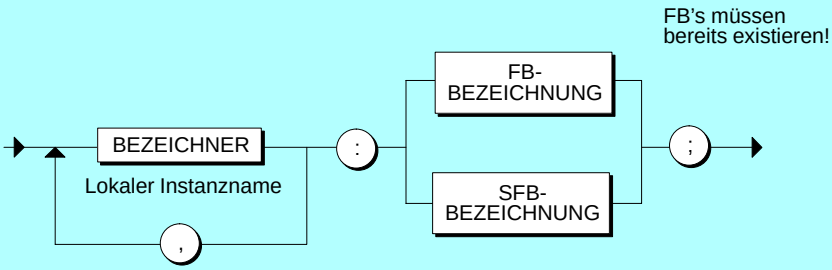
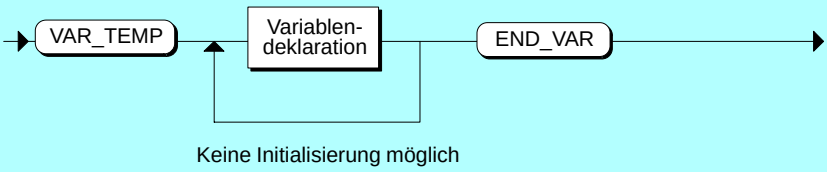
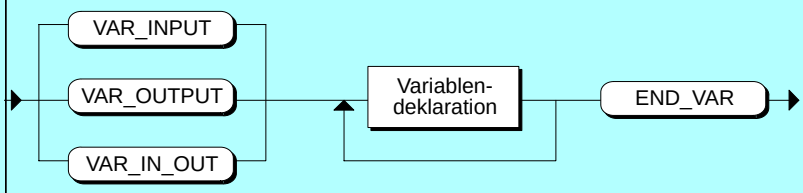
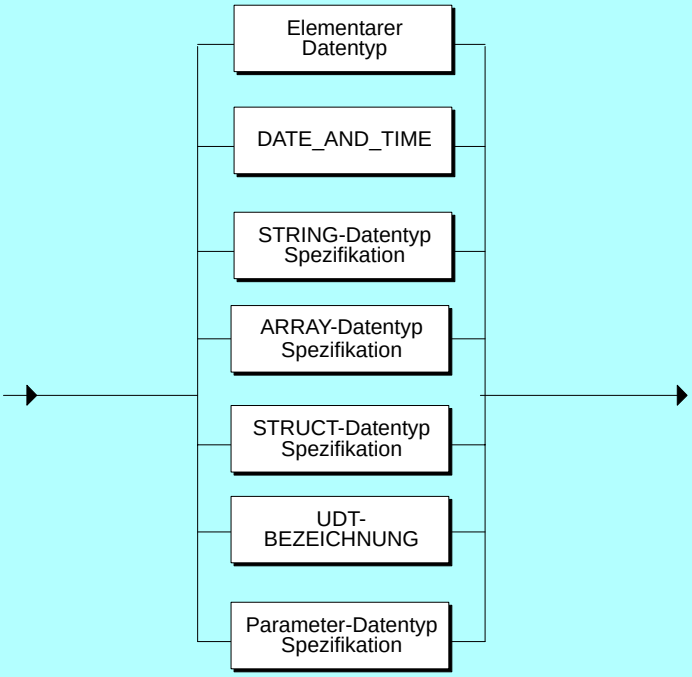
Regel	Syntaxdiagramm
Datentyp-Initialisierung	
Feld-Initialisierungs-liste	
Instanzdeklaration	
Temporärer Variablenblock	

Tabelle C-3 Syntax der Vereinbarungsblöcke, Fortsetzung

Regel	Syntaxdiagramm
Parameterblock	 Initialisierung nur möglich für VAR_INPUT und VAR_OUTPUT
Datentyp-Spezifikation	

C.3 Datentypen in SCL

Tabelle C-4 Syntax der Datentypen im Vereinbarungsteil

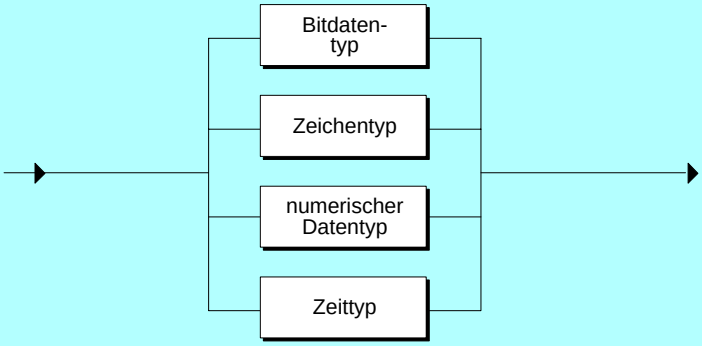
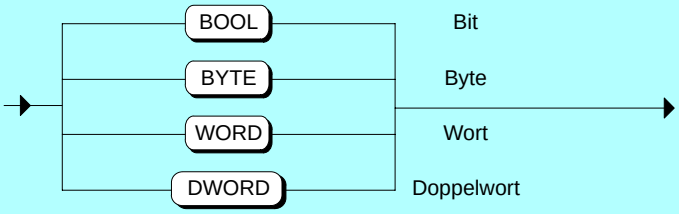
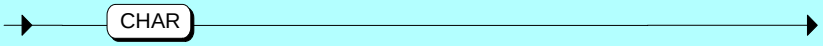
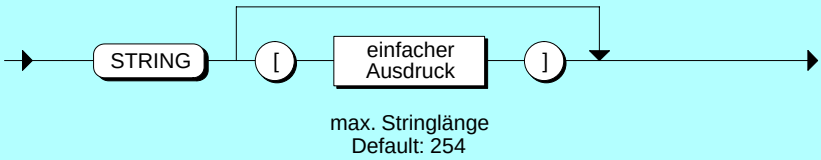
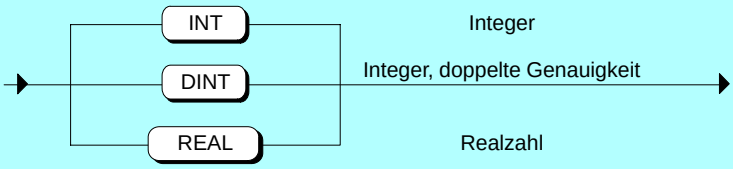
Regel	Syntaxdiagramm
Elementarer Datentyp	
Bitdatentyp	
Zeichentyp	
STRING-Datentyp Spezifikation	
Numerischer Datentyp	

Tabelle C-4 Syntax der Datentypen im Vereinbarungsteil, Fortsetzung

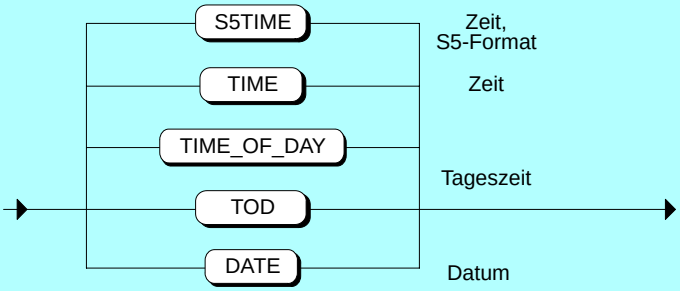
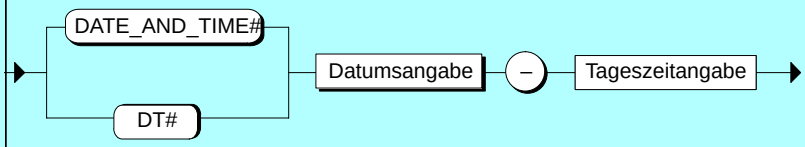
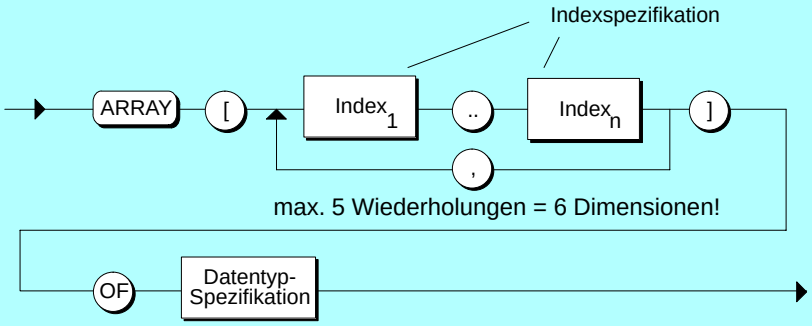
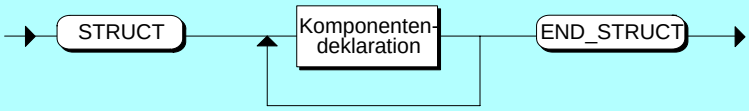
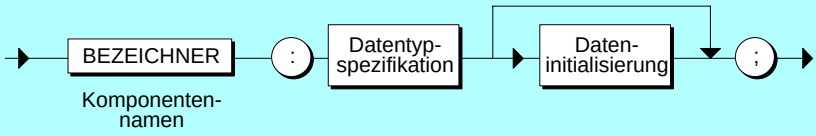
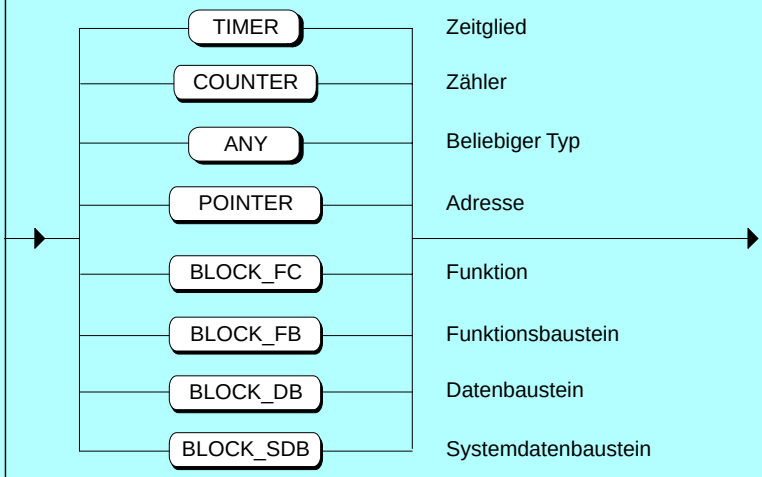
Regel	Syntaxdiagramm
Zeittyp	 siehe auch Kapitel B.1.1
DATE_AND_TIME	
ARRAY-Datentyp Spezifikation	
STRUCT-Datentyp Spezifikation Vergessen Sie nicht, das Schlüsselwort END_STRUCT mit Semikolon abzuschließen !	

Tabelle C-4 Syntax der Datentypen im Vereinbarungsteil, Fortsetzung

Regel	Syntaxdiagramm
Komponentendeklaration	
Parametertyp Spezifikation	

C.4 Anweisungsteil

Tabelle C-5 Syntax des Anweisungsteils

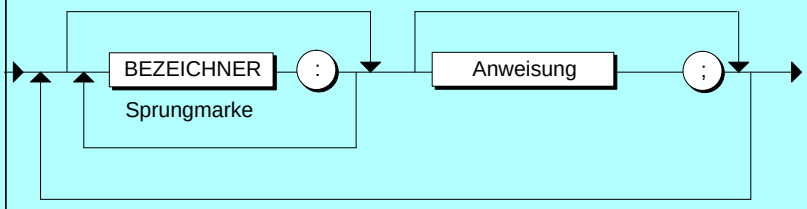
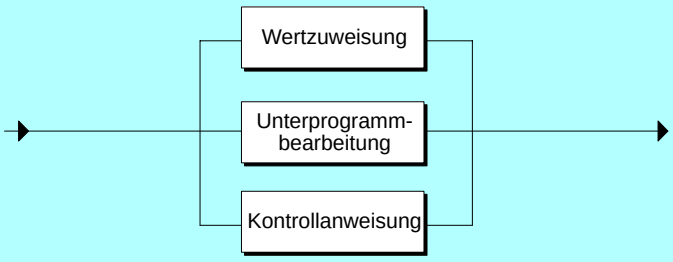
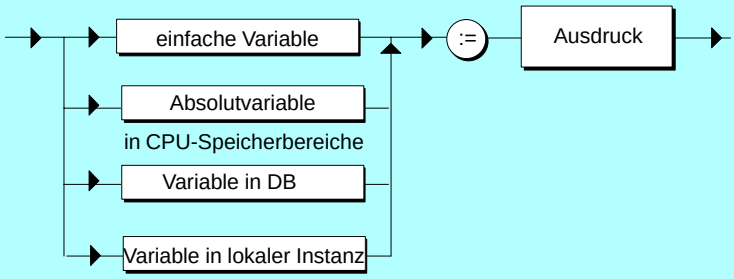
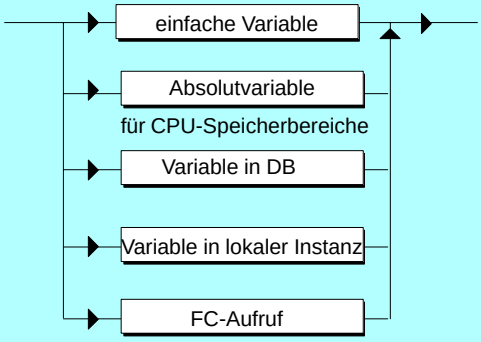
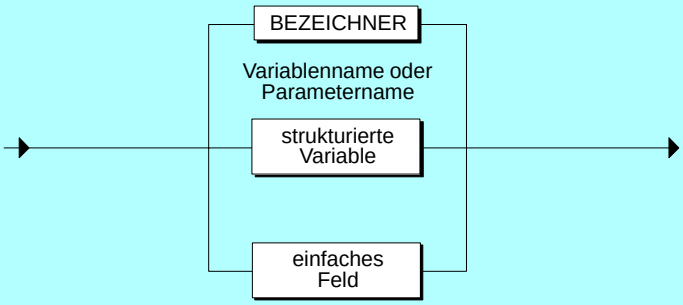
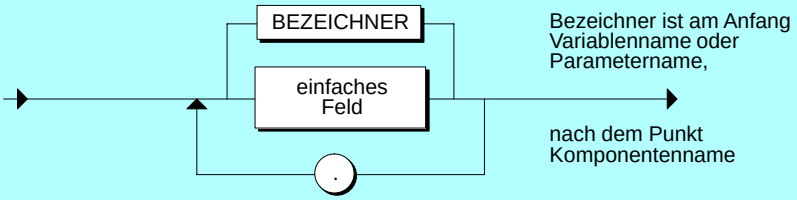
Regel	Syntaxdiagramm
Anweisungsteil	
Anweisung	
Wertzuweisung	
Erweiterte Variable	

Tabelle C-5 Syntax des Anweisungsteils, Fortsetzung

Regel	Syntaxdiagramm
Einfache Variable	
Strukturierte Variable	

C.5 Wertzuweisungen

Tabelle C-6 Syntax der Wertzuweisungen

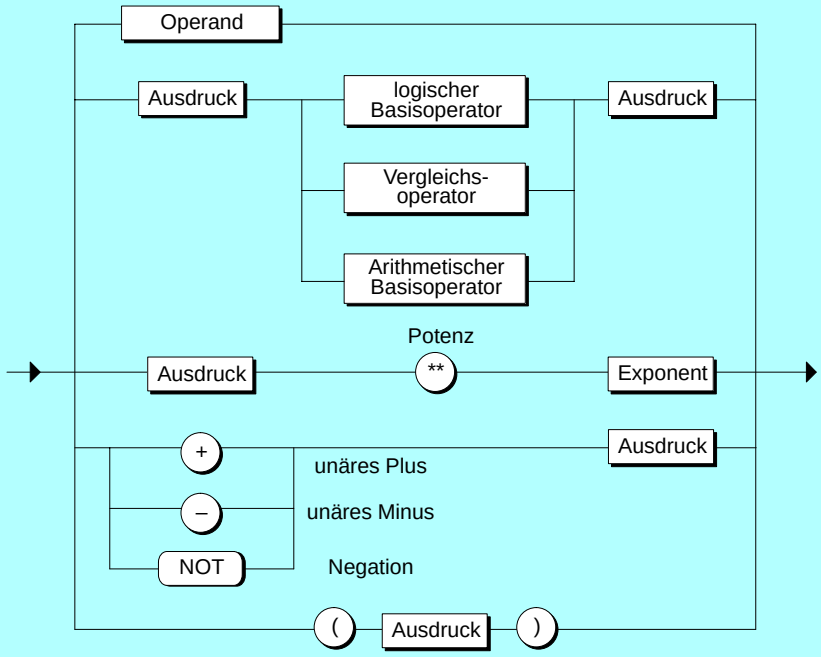
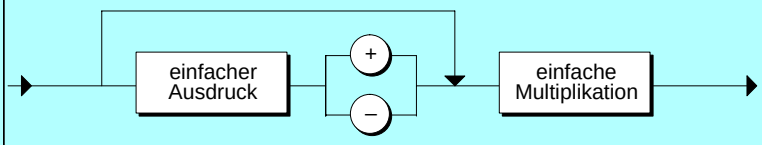
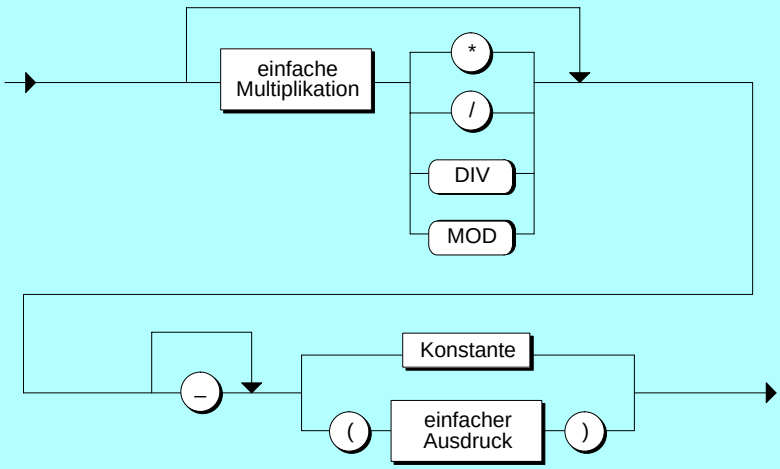
Regel	Syntaxdiagramm
Ausdruck	
Einfacher Ausdruck	
Einfache Multiplikation	

Tabelle C-6 Syntax der Wertzuweisungen, Fortsetzung

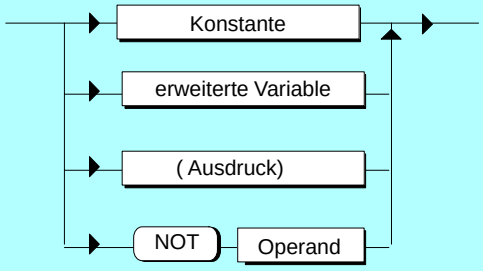
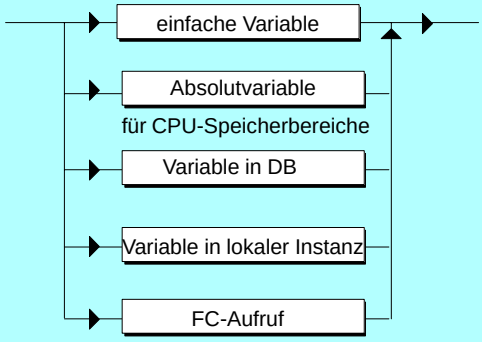
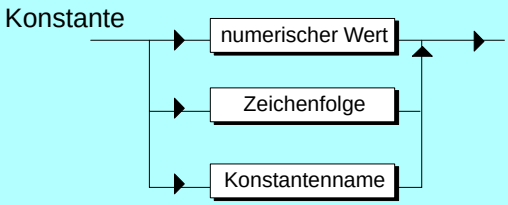
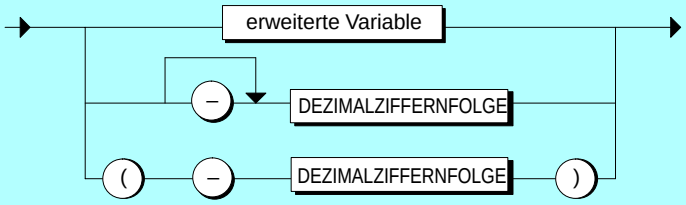
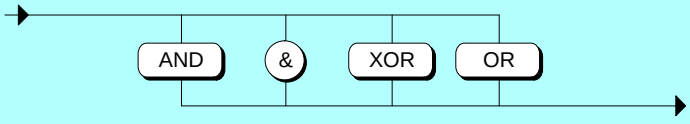
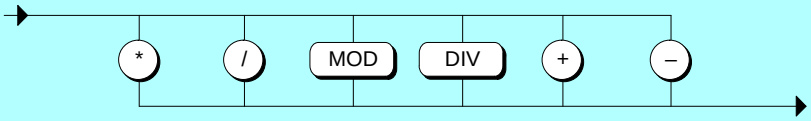
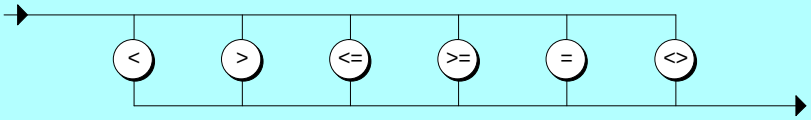
Regel	Syntaxdiagramm
Operand	
Erweiterte Variable	
Konstante	
Exponent	
Logischer Basisoperator	

Tabelle C-6 Syntax der Wertzuweisungen, Fortsetzung

Regel	Syntaxdiagramm
Arithmetischer Basisoperator	
Vergleichsoperator	

C.6 Aufruf von Funktionen und Funktionsbausteinen

Tabelle C-7 Syntax der Aufrufe

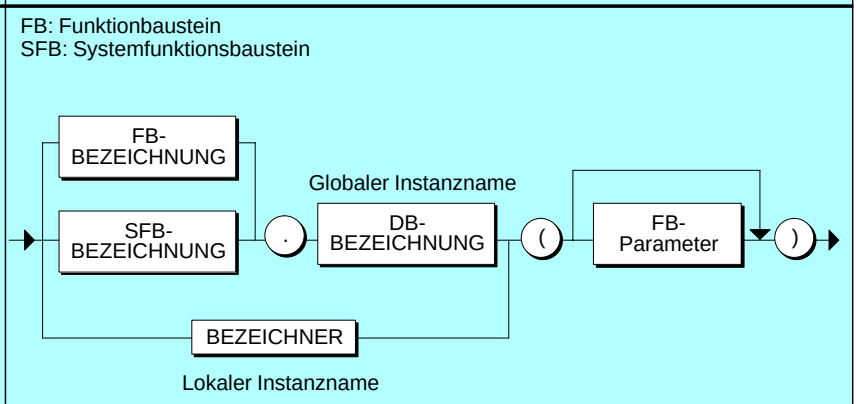
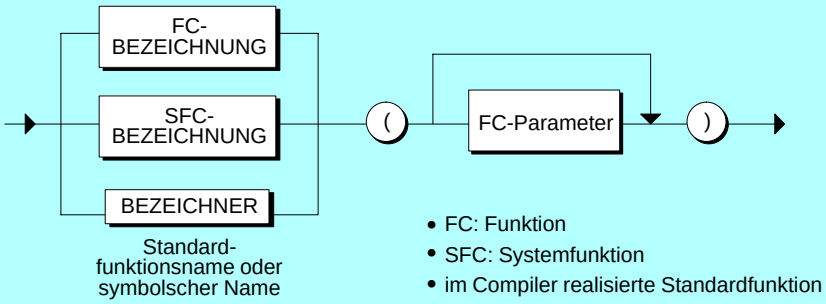
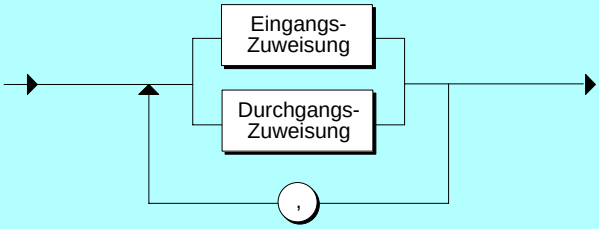
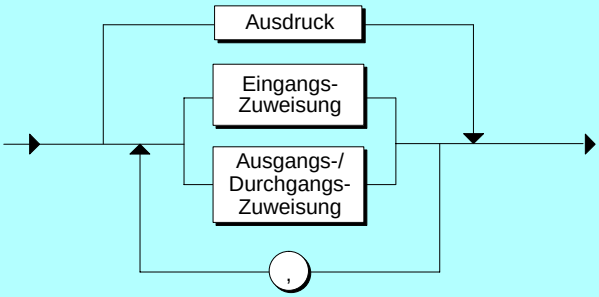
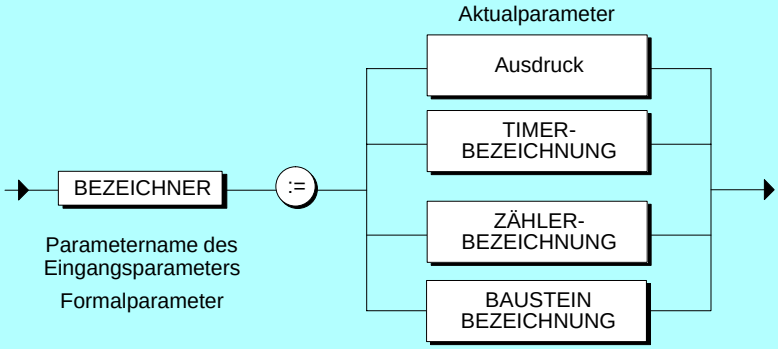
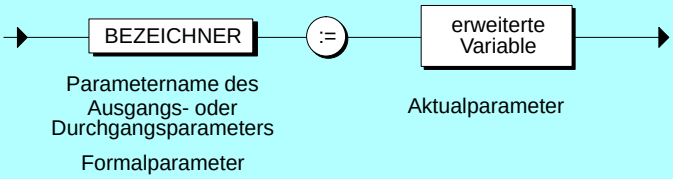
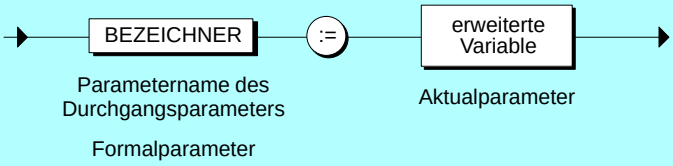
Regel	Syntaxdiagramm
FB-Aufruf	
Funktionsaufruf	
FB-Parameter	
FC-Parameter	

Tabelle C-7 Syntax der Aufrufe, Fortsetzung

Regel	Syntaxdiagramm
Eingangszuweisung	
Ausgangs-/Durch-gangszuweisung	
Durchgangszuweisung	

C.7 Kontrollanweisungen

Tabelle C-8 Syntax der Kontrollanweisungen

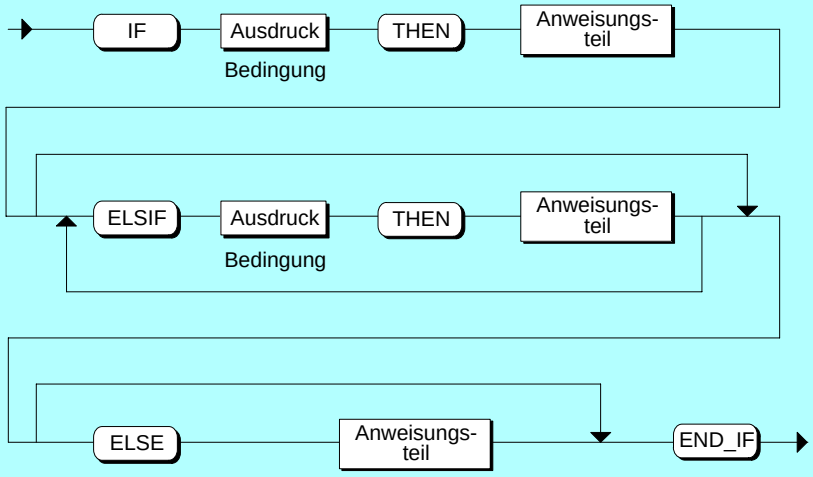
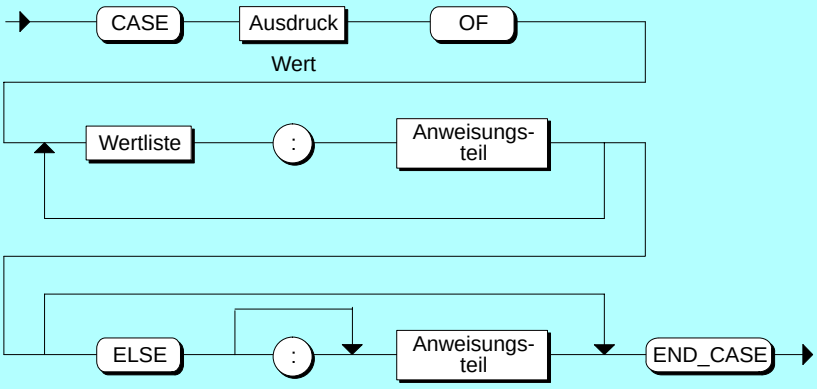
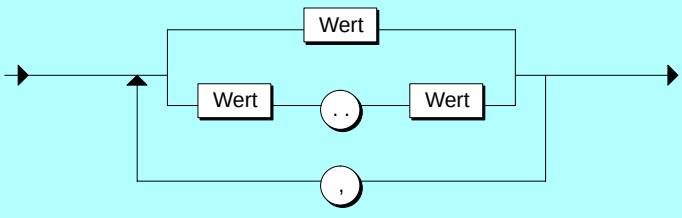
Regel	Syntaxdiagramm
IF-Anweisung Vergessen Sie nicht, das Schlüsselwort END_IF mit Semikolon abzuschließen !	
Case-Anweisung Vergessen Sie nicht, das Schlüsselwort END_CASE mit Semikolon abzuschließen !	
Wertliste	

Tabelle C-8 Syntax der Kontrollanweisungen, Fortsetzung

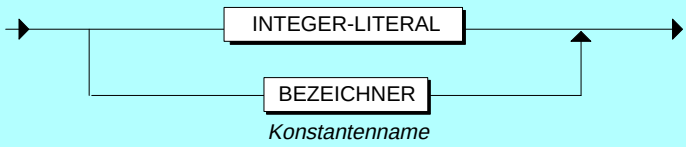
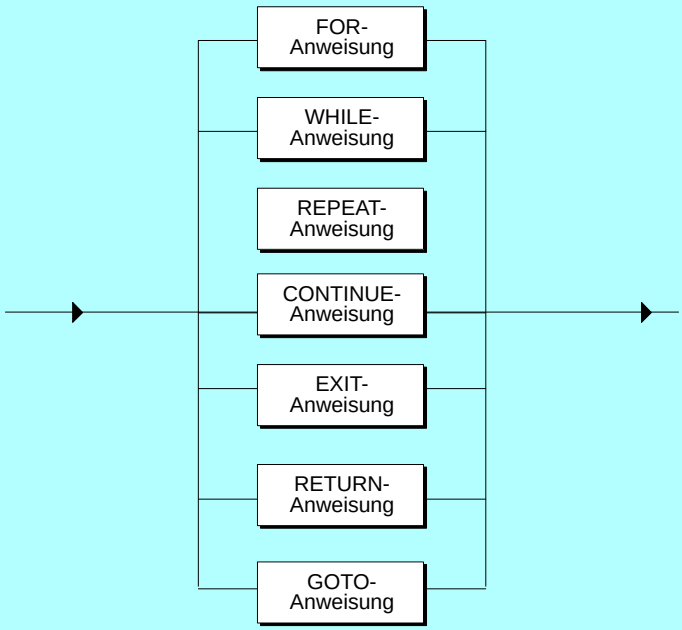
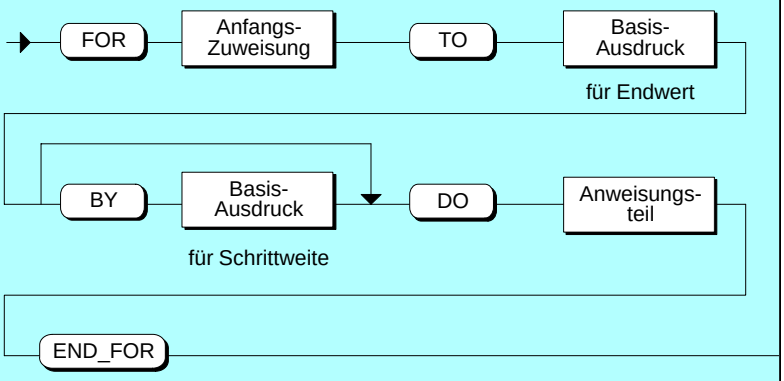
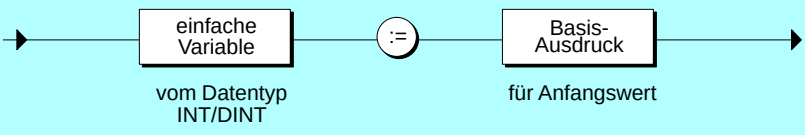
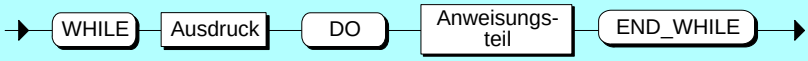
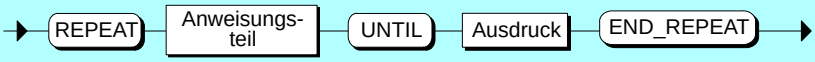
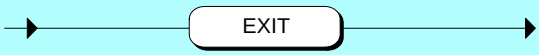
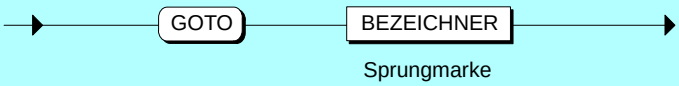
Regel	Syntaxdiagramm
Wert	
Wiederholungs-anweisungen und Sprunganweisungen	
FOR-Anweisung Vergessen Sie nicht, das Schlüsselwort END_FOR mit Semikolon abzuschließen !	
Anfangszuweisung	

Tabelle C-8 Syntax der Kontrollanweisungen, Fortsetzung

Regel	Syntaxdiagramm
WHILE-Anweisung Vergessen Sie nicht, das Schlüsselwort END_WHILE mit Semikolon abzuschließen !	
REPEAT-Anweisung Vergessen Sie nicht, das Schlüsselwort END_REPEAT mit Semikolon abzuschließen !	
CONTINUE-Anweisung	
RETURN-Anweisung	
EXIT-Anweisung	
Programmsprung	

Literaturverzeichnis

D

/12/ Broschüre: *Automatisierungssystem S7-300,*
Aufbau und Anwendung

/13/ Broschüre: *Automatisierungssystem S7-400,*
Aufbau und Anwendung

/14/ Broschüre: *Automatisierungssystem M7-300/400,*
Aufbau und Anwendung

/20/ Broschüre: *Automatisierungssystem S7-300/400,*
Programmierung

/25/ Broschüre: *Automatisierungssystem M7,*
Programmierung

/30/ Fibel: *Automatisierungssystem S7-300,*
Einfach aufbauen und programmieren

/70/ Handbuch: *Automatisierungssystem S7-300,*
Aufbauen, CPU-Daten

/71/ Referenzhandbuch: *Automatisierungssysteme S7-300, M7-300*
Baugruppendaten

/72/ Operationsliste: *Automatisierungssystem S7-300*

/100/ Installationshandbuch: *Automatisierungssystem S7-400, M7-400,*
Aufbauen

/101/ Referenzhandbuch: *Automatisierungssysteme S7-400, M7-400*
Baugruppendaten

/102/ Operationsliste: *Automatisierungssystem S7-400*

/230/ Umsteigerhandbuch: *Basissoftware für S7und M7,*
Von S5 nach S7

/231/ Benutzerhandbuch: *Basissoftware für S7und M7,*
STEP 7

/232/ Handbuch: *AWL für S7-300/400,*
Bausteine programmieren

/233/ Handbuch: *KOP für S7-300/400,*
Bausteine programmieren

/234/ Programmierhandbuch: *Systemsoftware für S7-300/400*
Programmentwurf

/235/ Referenzhandbuch: *Systemsoftware für S7-300/400*
System-und Standardfunktionen

/236/ Handbuch: *FUP für S7-300/400,*
Bausteine programmieren

/237/ *Gesamtindex*, STEP 7

/249/ Handbuch: *CFC für S7 und M7Band 2*

/251/ Handbuch: *GRAPH für S7-300/400*,
Ablaufsteuerungen programmieren

/252/ Handbuch: *HiGraph für S7-300/400*,
Zustandsgraphen programmieren

/253/ Handbuch: *C für S7-300/400*,
C-Programme erstellen

/254/ Handbuch: *CFC für S7 und M7Band 1*,

/290/ Benutzerhandbuch: *ProC/C++ für M7-300/400*,
C-Programme erstellen

/291/ Benutzerhandbuch: *ProC/C++ für M7-300/400*,
Debugger für C-Programme

/800/ *DOCPRO*
Schaltbücher normgerecht erstellen (nur auf CD)

/803/ Referenzhandbuch: *Systemsoftware für S7-300/400*
STEP 7 Standardfunktionen Teil 2 (nur auf CD)

Glossar

A

Adressierung, absolut

Bei der absoluten Adressierung wird die Adresse des zu bearbeitenden Operanden angegeben. Beispiel: Die Adresse A 4.0 bezeichnet das Bit 0 im Byte 4 des Prozeßabbilds der Ausgänge.

Adressierung, symbolisch

Bei der symbolischen Adressierung wird der zu bearbeitende Operand symbolisch angegeben (anstelle einer Adresse).

Aktualparameter

Aktualparameter ersetzen beim Aufruf eines Funktionsbausteins (FB) oder einer Funktion (FC) die Formalparameter.

Beispiel: Der Formalparameter ”Start” wird ersetzt durch den Aktualparameter ”E 3.6”

Anweisung

Eine Anweisung ist die kleinste selbständige Einheit eines in einer textuellen Sprache erstellten Anwenderprogramms. Sie stellt eine Arbeitsvorschrift für den Prozessor dar.

Anweisungsliste (AWL)

Die Anweisungsliste (AWL) ist eine maschinennahe, textuelle Programmiersprache.

Anwenderprogramm

Das Anwenderprogramm enthält alle Anweisungen und Deklarationen für die Signalverarbeitung, durch die eine Anlage oder ein Prozeß gesteuert werden können. Es ist einer programmierbaren Baugruppe (z. B. CPU, FM) zugeordnet und kann in kleinere Einheiten (Bausteine) strukturiert werden.

Attribut

Ein Attribut ist eine Eigenschaft, die z.B. an eine Bausteinbezeichnung oder einen Variablennamen angehängt werden kann. Bei SCL gibt es z. B. Attribute für folgende Angaben: Bausteinüberschrift, Ausgabestand, Bausteinschutz, Autor, Bausteinname, Bausteinfamilie.

Aufrufhierarchie

Alle Bausteine müssen erst aufgerufen werden, ehe sie bearbeitet werden können. Die Reihenfolge und Schachtelung dieser Aufrufe wird Aufrufhierarchie genannt.

Aufrufschnittstelle Die Aufrufschnittstelle wird definiert durch die Eingangs-, Ausgangs- und Durchgangsparameter (Formalparameter) eines Bausteins im Anwenderprogramm. Bei Aufruf des Bausteins werden diese Parameter durch die Aktualparameter ersetzt.

Ausdruck Ein Ausdruck dient in SCL zur Verarbeitung von Daten. Es wird unterschieden zwischen arithmetischen, logischen Ausdrücken und Vergleichsausdrücken.

Ausgangsparameter (A-Parameter) Mit den Ausgangsparametern eines Bausteins im Anwenderprogramm werden Ergebnisse an den aufrufenden Baustein übergeben.

B

Baustein Bausteine sind durch ihre Funktion, ihre Struktur oder ihren Verwendungszweck abgegrenzte Teile des Anwenderprogrammes. Es gibt bei STEP 7 Codebausteine (FB, FC, OB, SFC, SFB) Datenbausteine (DB, SDB) und anwenderdefinierte Datentypen (UDT).

Bausteinkommentar Zusatzinformationen zu einem Baustein (z.B. Erläuterungen zum automatisierten Prozeß), die nicht in den Arbeitsspeicher von SIMATIC S7-Automatisierungssystemen geladen werden.

Bausteinschutz Als Bausteinschutz bezeichnet man die Möglichkeit, einzelne Bausteine gegen Rückübersetzung zu schützen, wenn die Übersetzung der Bausteinquelle mit dem Schlüsselwort ”KNOW_HOW_PROTECTED” vorgenommen wurde.

Behälter Ordner auf der Benutzeroberfläche des SIMATIC Managers, der geöffnet werden kann und weitere Behälter und Objekte enthalten kann.

Bausteinart Die Bausteinarchitektur von STEP 7 kennt folgende Bausteinarten: Organisationsbausteine, Funktionsbausteine, Funktionen, Datenbausteine sowie Systemfunktionsbausteine, Systemfunktionen, Systemdatenbausteine und anwenderdefinierte Datentypen ⇒ Baustein.

Bausteinaufruf Starten eines Bausteins im STEP 7-Anwenderprogramm: Organisationsbausteine werden grundsätzlich vom Betriebssystem aufgerufen, alle anderen Bausteine werden vom STEP 7-Anwenderprogramm aufgerufen.

Bausteinklasse Bausteine unterteilt man, ihrem Inhalt entsprechend, in zwei Klassen:

Codebausteine und Datenbausteine.

Bezeichner Mit Bezeichnern werden Sprachobjekte von SCL angesprochen. Es gibt folgende Klassen: Standardbezeichner, vordefinierte Namen und Schlüsselwörter, Absolutbezeichner (bzw. Operandenkennzeichen), frei wählbare Namen, z. B. für Variablen und Sprungmarken, oder in einer Symboltabelle erzeugte symbolische Namen.

BCD-Darstellung Bei STEP 7 erfolgt die CPU-interne Angabe von Zeiten und Zählern nur im BCD-Format. BCD steht für ”Binär-Code für Dezimalzahlen”.

Blockstatus ⇒ Kontinuierlich beobachten.

C

CASE-Anweisung Diese Anweisung ist eine Verzweigungsanweisung. Sie dient abhängig vom Wert eines Auswahlausdrucks der 1 aus n Auswahl eines Programmteils.

Codebaustein Ein Codebausteinen ist bei SIMATIC S7 ein Baustein, der einen Teil des STEP 7-Anwenderprogramms enthält. Im Gegensatz dazu enthält ein Datenbaustein nur Daten. Es gibt folgende Codebausteine: Organisationsbausteine (OB), Funktionsbausteine (FB), Funktionen (FC), Systemfunktionsbausteine (SFB) und Systemfunktionen (SFC).

CONTINUE-Anweisung Beendet eine Laufschleife und beginnt sie mit dem nächsten Wert der Laufvariable.

D

Daten, global Globale Daten sind Speicherbereiche der CPU, die von jeder Programmstelle aus ansprechbar sind (z.B. Merker).

Datenbaustein (DB) Datenbausteine (DB) sind Datenbereiche im Anwenderprogramm, die Anwenderdaten enthalten. Auf Datenbausteine kann von allen Codebausteinen aus zugegriffen werden. Datenbausteine, die einem bestimmten FB-Aufruf zugeordnet sind, heißen Instanz-Datenbausteine.

Daten, statisch Statische Daten sind Lokaldaten eines Funktionsbausteins, die im Instanz-Datenbaustein gespeichert werden und deshalb bis zur nächsten Bearbeitung des Funktionsbausteins erhalten bleiben.

Daten, temporär

Temporäre Daten gehören lokal zu einem Codebaustein und belegen **keinen** statischen Speicherbereich, da sie im Stack der CPU abgelegt werden. Ihr Wert bleibt nur während eines Bausteinablaufs erhalten.

Datentyp

Mit Hilfe eines Datentyps wird festgelegt, wie der Wert einer Variablen oder Konstanten im Anwenderprogramm verwendet werden soll. Bei SCL stehen dem Anwender drei Arten von Datentypen zur Verfügung:

- Elementare Datentypen (Datentyp, elementar)
- Zusammengesetzte Datentypen (Datentyp, zusammengesetzt)
- anwenderdefinierte Datentypen (UDT).

Datentyp, anwenderdefiniert

Anwenderdefinierte Datentypen (UDT) werden vom Anwender mit der Datentypdeklaration geschaffen. Sie haben einen eigenen Namen und sind mehrfach verwendbar. So kann ein anwenderdefinierter Datentyp zur Erzeugung mehrerer Datenbausteine mit der gleichen Struktur (z.B. Regler) genutzt werden.

Datentyp-deklaration

Mit der Datentypdeklaration kann der Anwender anwenderdefinierte Datentypen deklarieren.

Datentyp, elementar

Elementare Datentypen sind vordefinierte Datentypen gemäß IEC 1131-3. Beispiele: Datentyp ”`BOOL`” definiert eine binäre Variable (”`Bit`”); Datentyp ”`INT`” definiert eine 16-Bit-Festpunkt-Variable.

Datentyp, zusammengesetzt

Man unterscheidet zwischen Strukturen und Feldern. ”Strukturen” sind aus verschiedenen anderen Datentypen (z.B. elementaren Datentypen) zusammengesetzt. ”Felder” bestehen aus mehreren gleichartigen Elementen eines Datentyps. Auch die Datentypen `STRING` und `DATE_AND_TIME` sind zusammengesetzte Datentypen.

Deklarationstyp

Der Deklarationstyp gibt an, wie ein Parameter bzw. eine lokale Variable von einem Baustein verwendet werden soll. Es gibt Eingangsparameter, Ausgangsparameter und Durchgangsparameter sowie statische und temporäre Variablen.

Durchgangsparameter (D-Parameter)

Durchgangsparameter gibt es bei Funktionen und Funktionsbausteinen. Mit Durchgangsparametern werden Daten an den aufgerufenen Baustein übergeben, dort verarbeitet und die Ergebnisse vom aufgerufenen Baustein wieder in der gleichen Variablen abgelegt.

E

Eingabe, quellorientiert

Bei der quellorientierten Eingabe werden die Bausteine oder das gesamte Anwenderprogramm in einer Textdatei editiert. Eine Syntaxprüfung erfolgt erst bei der Übersetzung. Bei SCL wird eine quellorientierte Eingabe verwendet.

Eingangs-parameter (E-Parameter)

Eingangsparameter gibt es nur bei Funktionen und Funktionsbausteinen. Mit Hilfe der Eingangsparameter werden Daten zur Verarbeitung an den aufgerufenen Baustein übergeben.

Einzelschritt

Der Einzelschritt ist ein Testschritt innerhalb der Einzelschrittfunktion des Debuggers von SCL. In der Einzelschrittfunktion können Sie das Programm Anweisung für Anweisung ausführen und im Ergebnisfenster beobachten.

Enable (EN)

Bei STEP 7 hat jeder Baustein einen Eingang ”Enable” (EN), der beim Aufruf eines Bausteins gesetzt werden kann. Liegt an EN ein 1-Signal, dann wird der Baustein aufgerufen, bei 0-Signal wird er nicht aufgerufen.

Enable out (ENO)

Bei STEP 7 hat jeder Baustein einen Ausgang ”Enable Output” (ENO). Innerhalb des Bausteins kann der Anwender den Eingang ”Enable” mit einem internen Wert (UND) verknüpfen. Das Ergebnis wird automatisch dem Ausgang ENO zugewiesen. Mit ENO ist es möglich, bei verketteten Aufrufen von Bausteinen die Bearbeitung der Folgebausteine von der ordnungsgemäßen Bearbeitung des vorhergehenden Bausteins abhängig zu machen.

EXIT-Anweisung

Abbruch einer Laufschleife.

F

Feld

Ein Feld (ARRAY) ist ein zusammengesetzter Datentyp bestehend aus Datenelementen gleichen Typs. Diese Datenelemente können wiederum elementar oder zusammengesetzt sein.

FOR-Anweisung

Eine FOR-Anweisung dient zur Wiederholung einer Folge von Anweisungen solange die Laufvariable innerhalb eines angegebenen Wertebereichs liegt.

Formalparameter

Ein Formalparameter ist ein Platzhalter für den ”tatsächlichen” Parameter (Aktualparameter) bei parametrierbaren Codebausteinen. Bei FB und FC werden die Formalparameter vom Anwender deklariert, bei SFB und SFC sind sie bereits vorhanden. Beim Aufruf des Bausteins wird dem Formalparameter ein Aktualparameter zugeordnet, so daß der aufgerufene Baustein mit diesem aktuellen Wert arbeitet. Die Formalparameter zählen zu den Lokaldaten des Bausteins und unterteilen sich nach Eingangs-, Ausgangs, und Durchgangsparametern.

Funktion (FC)

Eine Funktion (FC) ist gemäß IEC 1131-3 ein Codebaustein **ohne** statische Daten. Eine Funktion bietet die Möglichkeit der Übergabe von Parametern im Anwenderprogramm. Dadurch eignen sich Funktionen zur Parametrierung von häufig wiederkehrenden komplexen Funktionen, z.B. Berechnungen.

Funktionsbaustein (FB)

Ein Funktionsbaustein (FB) ist gemäß IEC 1131-3 ein Codebaustein mit statischen Daten (Daten, statisch). Ein FB bietet die Möglichkeit der Übergabe von Parametern im Anwenderprogramm. Dadurch eignen sich Funktionsbausteine zur Programmierung von häufig wiederkehrenden komplexen Funktionen z.B. Regelungen, Betriebsartenanwahl. Da ein FB über ein Gedächtnis (Instanz-Datenbaustein) verfügt, kann auf seine Parameter (z. B. Ausgänge) zu jeder Zeit an jeder beliebigen Stelle im Anwenderprogramm zugegriffen werden.

G

Globale Daten

Globale Daten sind Daten, die von jedem Codebaustein (FC, FB, OB) aus ansprechbar sind. Im einzelnen sind das Merker M, Eingänge E, Ausgänge A, Zeiten, Zähler und Elemente von Datenbausteinen DB. Auf globale Daten kann entweder absolut oder symbolisch zugegriffen werden.

GOTO-Anweisung

Eine GOTO-Anweisung bewirkt den sofortigen Sprung zu einer angegebenen Marke.

H

HALT

Der Betriebszustand HALT wird aus dem Betriebszustand RUN durch Anforderung vom Programmiergerät erreicht. In diesem Betriebszustand sind spezielle Testfunktionen möglich.

Haltepunkt

Mit dieser Funktion kann die Zentralbaugruppe (CPU) an definierten Programmstellen in den Betriebszustand HALT versetzt werden. Beim Erreichen eines Haltepunktes können die Testfunktionen wie z.B. schrittweise Befehlsbearbeitung oder Variablen beobachten/steuern durchgeführt werden.

I

Instanz

Mit ”Instanz” wird der Aufruf eines Funktionbausteins bezeichnet. Dabei ist ihm ein Instanz-Datenbaustein oder eine Lokale Instanz zugeordnet. Wird ein Funktionsbaustein im STEP 7-Anwenderprogramm n-mal mit jeweils unterschiedlichen Parametern und Instanz-Datenbausteinnamen aufgerufen, so existieren n Instanzen

```
FB13.DB1 (P1:=...), FB13.DB3(P3:=...),
FB13.DB2 (P2:=...), .....FB13.DBn (Pn:=...).
```

Instanz, lokal

Eine lokale Instanz wird im Teil der statischen Variablen eines Funktionsbausteins definiert. Anstelle eines gesamten Instanz-Datenbausteins wird nur ein lokaler Teil als Datenbereich für den Funktionsbaustein verwendet, der mit dem lokalen Instanznamen aufgerufen wird.

Integer (INT)

Integer (INT) ist einer der elementaren Datentypen. Die Darstellung erfolgt als 16-bit Ganzzahl.

Instanz-Datenbaustein (Instanz-DB)

Ein Instanz-Datenbaustein speichert die Formalparameter und statischen Lokaldaten von Funktionsbausteinen. Ein Instanz-Datenbaustein kann einem FB-Aufruf oder einer Aufrufhierarchie von Funktionsbausteinen zugeordnet sein. Er wird bei SCL automatisch generiert.

K

Konstante (symbolisch)

Konstanten mit symbolischem Namen sind Platzhalter für konstante Werte bei Codebausteinen. Symbolische Konstanten werden verwendet, um die Lesbarkeit eines Programms zu erhöhen.

Konstante (Literal)

Konstanten, deren Wert und Typ durch die formale Schreibweise bestimmt werden. Es werden numerische Literale, Zeichenliterale und Literale für Zeitangaben unterschieden.

Konvertieren, explizit

Explizit konvertieren bedeutet, eine Konvertierungsfunktion in das Quellprogramm einfügen. Bei der Verknüpfung von zwei Operanden ungleichen Datentyps muß der Anwender eine explizite Konvertierung durchführen: Beim Wechsel in eine andere Typklasse, z. B. von einem Bitdatentyp in einen Numerischen Datentyp, und – wenn der Zieldatentyp weniger mächtig ist als der Quelldatentyp – auch beim Wechsel innerhalb einer Typklasse.

Konvertieren, implizit

Implizit konvertieren bedeutet, daß eine Konvertierungsfunktion automatisch durch den Compiler eingefügt wird. Bei der Verknüpfung von zwei Operanden ungleichen Datentyps erfolgt eine implizite Konvertierung: Wenn kein Wechsel in eine andere Typklasse erfolgt und wenn der Zieldatentyp nicht weniger mächtig ist als der Quelldatentyp.

Kontinuierlich beobachten

Testmodus von SCL. Beim kontinuierlichen Beobachten eines Programms können Sie eine Gruppe von Anweisungen testen. Diese Gruppe von Anweisungen nennt man auch Beobachtungsbereich.

L

Laden in Zielsystem

Laden von ladbaren Objekten (z.B. Codebausteine) vom Programmiergerät in den Ladespeicher einer programmierbaren Baugruppe. Dies kann sowohl über ein direkt angeschlossenes Programmiergerät oder z.B. über den PROFIBUS geschehen.

Laden in PG

Laden von ladbaren Objekten (z.B. Codebausteine) aus dem Ladespeicher einer programmierbaren Baugruppe in das Programmiergerät. Dies kann sowohl über ein direkt angeschlossenes Programmiergerät oder z.B. über den PROFIBUS geschehen.

Lexikalische Regel

Die untere Regelstufe der formalen SCL-Sprachbeschreibung besteht aus den lexikalischen Regeln. Bei ihrer Anwendung besteht keine Formatfreiheit, d.h. die Ergänzung von Leerzeichen und Steuerzeichen(z. B.), ist nicht erlaubt.

Lokaldaten

Lokaldaten sind die einem Codebaustein zugeordneten Daten, die in seinem Deklarationsteil bzw. Vereinbarungsteil deklariert werden. Sie umfassen (bausteinabhängig): Formalparameter, statische Daten, temporäre Daten.

M

Merker (M)

Speicherbereich im Systemspeicher einer SIMATIC S7-CPU. Auf ihn kann schreibend und lesend zugegriffen werden (bit-, byte-, wort- und doppelwortweise). Der Merkerbereich kann vom Anwender zum Speichern von Zwischenergebnissen verwendet werden.

Mnemonik

Die Mnemonik ist eine abgekürzte Darstellung der Operanden und der Programmieroperationen im Programm (z. B. steht ”E” für Eingang). STEP 7 unterstützt die IEC-Darstellung (die auf der englischen Sprache basiert) und die SIMATIC-Darstellung (die auf der deutschen Darstellung der Operationen und den Konventionen für SIMATIC-Adressierung beruht).

Multiinstanz

Bei der Verwendung von Multiinstanzen enthält der Instanz-Datenbaustein die Daten für mehrere Funktionsbausteine einer Aufrufhierarchie.

N

Nonterminal

Ein Nonterminal ist ein zusammengesetztes Element, das durch eine weitere lexikalische oder syntaktische Regel beschrieben wird.

Nutzdaten

Nutzdaten werden zwischen einer Zentralbaugruppe und Signalbaugruppe, Funktionsbaugruppe und Kommunikationsbaugruppen über das Prozeßabbild oder über Direktzugriffe ausgetauscht. Nutzdaten können sein: Digitale und analoge Ein-/Ausgangssignale von Signalbaugruppen, Steuer- und Statusinformationen von Funktionsbaugruppen.

O

Offline

Offline bezeichnet den Betriebszustand, bei dem das Programmiergerät keine Verbindung mit dem Automatisierungssystem hat (physikalisch, logisch).

OK-Variable

Die OK-Variable dient dazu, die korrekte oder inkorrekte Ausführung einer Bausteinbefehlsfolge zu vermerken. Sie ist global vom Typ BOOL.

Online

Online bezeichnet den Betriebszustand, bei dem das Programmiergerät mit dem Automatisierungssystem verbunden ist (physikalisch, logisch).

Online-Hilfe

STEP 7 bietet Ihnen die Möglichkeit, sich während des Arbeitens mit der Programmiersoftware kontextabhängige Hilfetexte am Bildschirm anzeigen zu lassen.

Operand

Ein Operand ist ein Teil einer Anweisung und sagt aus, womit der Prozessor etwas tun soll. Er kann sowohl absolut als auch symbolisch adressiert werden.

Operanden-kennzeichen

Ein Operandenkennzeichen ist der Teil des Operanden einer Operation, in dem Informationen enthalten sind, wie z. B. der Speicherbereich, in dem die Operation einen Wert (Datenobjekt) findet, mit dem sie eine Verknüpfung ausführt oder die Größe eines Werts (Datenobjekt), mit dem sie eine Verknüpfung ausführt. In der Anweisung ”Wert := EB10” ist ”EB” das Operandenkennzeichen (”E” steht für den Eingangsbereich des Speichers, ”B” steht für ein Byte in diesem Bereich).

Operation

Eine Operation ist Teil einer Anweisung und sagt aus, was der Prozessor tun soll.

Organisations-baustein (OB)

Organisationsbausteine bilden die Schnittstelle zwischen dem Betriebssystem der CPU und dem Anwenderprogramm. In den Organisationsbausteinen wird die Reihenfolge der Bearbeitung des Anwenderprogramms festgelegt.

P

Parameter

Bei SCL: Variable eines Codebausteins (Aktualparameter, Formalparameter).

Parametertyp

Ein Parametertyp ist ein spezieller Datentyp für Zeiten, Zähler und Bausteine. Er kann bei Eingangsparametern von Funktionsbausteinen **und** Funktionen, bei Durchgangsparametern nur von Funktionsbausteinen verwendet werden, um Zeiten, Zähler und Bausteine an den aufgerufenen Baustein zu übergeben.

Programmierung, strukturiert

Zur Lösung komplexer Automatisierungsaufgaben wird das Anwenderprogramm in einzelne abgeschlossene Programmteile (Bausteine) unterteilt. Die Gliederung des Anwenderprogramms erfolgt funktional oder entsprechend der technologischen Anlagenstruktur.

Programmierung, symbolisch

Die Programmiersprache SCL ermöglicht das Verwenden von symbolischen Zeichenfolgen anstelle von Operanden: z. B. der Operand A1.1 kann ersetzt werden durch ”Ventil_17”. Die Symboltabelle bei STEP 7 stellt die Verbindung zwischen Operand und der zugeordneten symbolischen Zeichenfolge her.

Projekt

Ein Behälter für alle Objekte einer Automatisierungslösung unabhängig von der Anzahl der Stationen, Baugruppen und deren Vernetzung.

Prozeßabbild

Die Signalzustände der digitalen Ein- und Ausgabebaugruppen werden in der CPU in einem Prozeßabbild hinterlegt. Man unterscheidet das Prozeßabbild der Eingänge (PAE) und das der Ausgänge (PAA).

Prozeßabbild der Ausgänge (PAA)

Das Prozeßabbild der Ausgänge wird am Ende des Anwenderprogramms vom Betriebssystem auf die Ausgangsbaugruppen übertragen.

Prozeßabbild der Eingänge (PAE)

Das Prozeßabbild der Eingänge wird vor der Bearbeitung des Anwenderprogramms vom Betriebssystem von den Eingangsbaugruppen gelesen.

Q

Quelle

Eine Quelle (Textdatei) enthält Quellcode (ASCII-Text), der mit beliebigen Texteditoren erstellbar ist. Eine Quelle wird mit einem Compiler (AWL, SCL) in ein lauffähiges Anwenderprogramm übersetzt. Eine Quelle wird im Behälter ”Quellen” unter dem S7-Programm abgelegt.

Quellorientierte Eingabe

Die quellorientierte Eingabe eines STEP 7-Programms ist bei Programmierung in SCL möglich. Die Eingabe eines Programms ist mit jedem beliebigen Texteditor möglich. Der eigentliche Programmcode wird erst beim Übersetzungslauf erzeugt. Dann werden auch eventuelle Fehler erkannt. Diese Eingabeart eignet sich für die symbolische Erstellung von Standardprogrammen.

R

Realzahl

Eine Realzahl, auch Gleitpunktzahl genannt, ist eine positive oder negative Zahl, die einen Dezimalwert wie z.B. 0.339 oder –11.1 enthält.

REPEAT-Anweisung

Eine REPEAT-Anweisung dient zur Wiederholung einer Folge von Anweisungen bis zu einer Abbruchbedingung.

RETURN-Anweisung

Diese Anweisung bewirkt das Verlassen des aktuellen Bausteins.

Rückübersetzung

Durch die Rückübersetzung nach AWL ist es möglich, den in der CPU geladenen Baustein mit einem beliebigen PG/PC laden und anzeigen zu können. Dabei können bestimmte Teile des Bausteins, z. B. Symbolik und Kommentare, fehlen.

RUN

Im Betriebszustand RUN wird das Anwenderprogramm bearbeitet, das Prozeßabbild wird zyklisch aktualisiert. Alle digitalen Ausgänge sind freigegeben.

RUN-P

Der Betriebszustand RUN-P entspricht dem Betriebszustand RUN, mit dem Unterschied, daß bei dem Betriebszustand RUN-P sämtliche Programmiergerätefunktionen ohne Einschränkungen erlaubt sind.

S

S7-Anwender-programm

Das S7-Anwenderprogramm befindet sich im Behälter ”Bausteine”. Es enthält Bausteine, die auf eine programmierbare S7-Baugruppe (z.B. CPU) geladen werden und dort lauffähig sind, um eine Anlage oder einen Prozeß zu steuern.

SCL

PASCAL-ähnliche Hochsprache nach der Norm DIN EN-61131-3 (int. IEC 1131-3) zur Programmierung von komplexen Aufgaben in einer SPS, z. B. Algorithmen, Datenverarbeitungsaufgaben. Abkürzung für ”Structured Control Language”.

SCL-Compiler

Der SCL-Compiler ist ein Batch-Compiler, mit dem das zuvor editierte Programm (SCL-Quelle) in den MC7-Maschinencode übersetzt wird. Die dadurch erzeugten Bausteine werden im S7-Programm im Behälter ”Bausteine” abgelegt.

SCL-Debugger

Der SCL-Debugger ist ein Hochsprachendebugger, mit dem logische Programmierfehler im mit SCL erstellten Anwenderprogrammen gefunden werden können.

SCL-Editor

Der SCL-Editor ist ein auf SCL zugeschnittener Editor, mit dem die SCL-Quelle erstellt werden kann.

SCL-Quelle

Die SCL-Quelle ist die Datei, in der das Programm in SCL erstellt wird. Die Quelldatei wird anschließend mit dem SCL-Compiler übersetzt.

Schlüsselwort

Schlüsselwörter werden bei SCL verwendet, um den Beginn eines Bausteins zu kennzeichnen, um Sektionen im Deklarations- bzw. Vereinbarungsteil zu markieren und um Anweisungen zu kennzeichen. Außerdem werden sie für Kommentare und Attribute benutzt.

Single Step

⇒Einzelschritt

Speicherbereich Eine Zentralbaugruppe hat bei SIMATIC S7 drei Speicherbereiche: Den Ladebereich, den Arbeitsbereich und den Systembereich.

Statuswort Das Statuswort ist Bestandteil der Register der Zentralbaugruppe. Im Statuswort befinden sich Statusinformationen und Fehlerinformationen, die im Zusammenhang mit der Bearbeitung von STEP 7-Befehlen auftreten. Die Statusbits können vom Anwender gelesen und beschrieben werden; die Fehlerbits können nur gelesen werden.

Struktur (STRUCT) Eine Struktur ist ein zusammengesetzter Datentyp bestehend aus Datenelementen unterschiedlichen Typs. Diese Datenelemente können elementar oder zusammengesetzt sein.

Symbol Ein Symbol ist ein vom Anwender unter Berüksichtigung bestimmter Syntaxvorschriften definierter Name. Dieser Name kann nach der Festlegung, wofür er stehen soll (z.B. Variable, Datentyp, Baustein) bei der Programmierung und beim Bedienen und Beobachten verwendet werden.
Beispiel: Operand: E 5.0, Datentyp: Bool, Symbol: Taster_Notaus.

Symboltabelle Tabelle zur Zuordnung von Symbolen (=Name) zu Adressen für globale Daten und Bausteine. Beispiele: Notaus (Symbol) – E 1.7 (Adresse) oder Regler (Symbol) – SFB 24 (Baustein).

Syntaktische Regel Die obere Regelstufe der formalen SCL-Sprachbeschreibung besteht aus den syntaktischen Regeln. Bei ihrer Anwendung besteht Formatfreiheit, d.h. z. B. Leerzeichen und Steuerzeichen dürfen ergänzt werden.

Systemfunktion (SFC) Eine Systemfunktion (SFC) ist eine im Betriebssystem der CPU integrierte Funktion, die bei Bedarf im STEP 7-Anwenderprogramm aufgerufen werden kann.

Systemfunktionsbaustein (SFB) Ein Systemfunktionsbaustein (SFB) ist ein im Betriebssystem der CPU integrierter Funktionsbaustein, der bei Bedarf im STEP 7-Anwenderprogramm aufgerufen werden kann.

Systemdatenbaustein (SDB) System-Datenbausteine sind Datenbereiche in der Zentralbaugruppe, die Systemeinstellungen und Baugruppenparameter enthalten. Die Systemdatenbausteine werden mit der STEP 7 Basissoftware erzeugt und geändert.

Systemspeicher (Systembereich) Der Systemspeicher ist auf der Zentralbaugruppe integriert und als RAM-Speicher ausgeführt. Im Systemspeicher sind die Operandenbereiche (z. B. Zeiten, Zähler, Merker) sowie vom Betriebssystem intern benötigte Datenbereiche (z. B. Puffer für Kommunikation) abgelegt.

T

Terminal

Ein Terminal ist ein Grundelement einer lexikalischen oder syntaktischen Regel, das nicht durch eine weitere Regel, sondern verbal erklärt wird. Ein Terminal kann z.B. ein Schlüsselwort oder nur ein einzelnes Zeichen sein.

U

Übersetzen

Erzeugen eines lauffähigen Anwenderprogramms aus einer Quelle.

Übersetzung, quellorientiert

Bei der quellorientierten Eingabe wird erst beim Übersetzen auf eventuelle Eingabefehler geprüft. Ein ablauffähiger Code wird erst erzeugt, wenn keine Fehler mehr vorhanden sind.

UDT

⇒ Datentyp anwenderdefiniert.

V

Variable

Eine Variable definiert ein Datum mit variablem Inhalt, das im STEP 7-Anwenderprogramm verwendet werden kann. Eine Variable besteht aus einem Operanden (z. B. M 3.1) und einem Datentyp (z. B. Bool) und kann mit einem Symbol (z. B. BAND_EIN) gekennzeichnet werden. Die Variable wird im Vereinbarungsteil deklariert.

Variablen-deklaration

Die Variablendeklaration umfaßt die Angabe eines symbolischen Namens, eines Datentyps und evtl. eines Vorbelegungswerts und eines Kommentars.

Variablentabelle

In der Variablentabelle werden die Variablen zusammengestellt, die beobachtet und gesteuert werden sollen inkl. der zugehörigen Format-angaben.

Vereinbarungsteil

Hier werden die Lokaldaten eines Codebausteins deklariert.

Z

Zähler

Zähler sind Bestandteile des Systemspeichers der CPU. Der Inhalt dieser Zähler wird asynchron zum Anwenderprogramm vom Betriebssystem aktualisiert. Mit STEP 7-Anweisungen wird die genaue Funktion der Zählerzelle (z. B. Aufwärtszähler) festgelegt und ihre Bearbeitung (Start) angestoßen.

Zeiten

Zeiten sind Bestandteile des Systemspeichers der CPU. Der Inhalt dieser Zeiten wird asynchron zum Anwenderprogramm vom Betriebssystem aktualisiert. Mit STEP 7-Anweisungen wird die genaue Funktion der Zeitzelle (z. B. Einschaltverzögerung) festgelegt und ihre Bearbeitung (Start) angestoßen.

Zyklus-überwachungszeit

Überschreitet die Bearbeitungszeit des Anwenderprogramms die eingestellte Zyklusüberwachungszeit, so erzeugt das Betriebssystem eine Fehlermeldung und die CPU geht in den STOP-Zustand.

Zykluszeit

Die Zykluszeit ist die Zeit, die die CPU für die einmalige Bearbeitung des Anwenderprogramms benötigt.

Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

An
Siemens AG
AUT E 146
Östliche Rheinbrückenstr. 50
76181 Karlsruhe

Absender:

Ihr Name: _

Ihre Funktion: _

Ihre Firma: _

Straße: _

Ort: _

Telefon: _

Bitte kreuzen Sie Ihren zutreffenden Industriezweig an:

- ❒ Automobilindustrie
- ❒ Chemische Industrie
- ❒ Elektroindustrie
- ❒ Nahrungsmittel
- ❒ Leittechnik
- ❒ Maschinenbau
- ❒ Petrochemie
- ❒ Pharmazeutische Industrie
- ❒ Kunststoffverarbeitung
- ❒ Papierindustrie
- ❒ Textilindustrie
- ❒ Transportwesen
- ❒ Andere _ _ _ _ _ _ _ _ _ _ _ _ _

Vorschläge und Anmerkungen zur Anwenderdokumentation

Ihre Anmerkungen und Vorschläge helfen uns, die Qualität und Benutzbarkeit unserer Dokumentation zu verbessern. Bitte füllen Sie diesen Fragebogen bei der nächsten Gelegenheit aus und senden Sie ihn an Siemens zurück.

Geben Sie bitte bei den folgenden Fragen Ihre persönliche Bewertung mit Werten von 1 = gut bis 5 = schlecht an.

1. Entspricht der Inhalt Ihren Anforderungen? ☐
2. Sind die benötigten Informationen leicht zu finden? ☐
3. Sind die Texte leicht verständlich? ☐
4. Entspricht der Grad der technischen Einzelheiten Ihren Anforderungen? ☐
5. Wie bewerten Sie die Qualität der Abbildungen und Tabellen? ☐

Falls Sie auf konkrete Probleme gestoßen sind, erläutern Sie diese bitte in den folgenden Zeilen:

_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _
_ _