

Speicherprogrammierbare Steuerungen

Teil 3: Programmiersprachen
(IEC 1131-3 : 1993)
Deutsche Fassung EN 61131-3 : 1993

DIN**EN 61131-3**

Diese Norm enthält die deutsche Übersetzung der Internationalen Norm

IEC 1131-3

ICS 35.060; 35.240.50

Deskriptoren: Speicherprogrammierbare Steuerung, Programmiersprache, Darstellung

Programmable controllers – Part 3: Programming languages

Ersatz für DIN 19 239 : 1983-05

(IEC 1131-3 : 1993);

German version EN 61131-3 : 1993

Automates programmables – Partie 3: Langages de programmation

(CEI 1131-3 : 1993);

Version allemande EN 61131-3 : 1993

Die Europäische Norm EN 61131-3 : 1993 hat den Status einer Deutschen Norm.**Nationales Vorwort**

Zuständig für diese Europäische Norm ist in Deutschland das Unterkomitee 962.2 „SPS“ der Deutschen Elektrotechnischen Kommission im DIN und VDE (DKE).

Diese Norm enthält die Deutsche Fassung der Europäischen Norm EN 61131-3 : 1993. Gleichzeitig ist sie die deutsche Übersetzung der IEC 1131-3 : 1993, zu der der Anhang ZA hinzugefügt wurde.

Die Norm-Entwürfe waren veröffentlicht unter DIN IEC 65A(Sec)67/07.87 und DIN IEC 1131 Teil 3/09.92.

Die Reihe der Normen DIN EN 61131 „Speicherprogrammierbare Steuerungen“ wird aus folgenden Teilen bestehen, von denen dieser der dritte ist:

Teil 1: Allgemeine Informationen

Teil 2: Betriebsmittelanforderungen und Prüfungen

Teil 3: Programmiersprachen

Teil 4: Anwenderrichtlinien (bei IEC in Bearbeitung)

Teil 5: Kommunikation (bei IEC in Bearbeitung)

Zur Ergänzung der IEC 1131-3 werden in der Internationalen Elektrotechnischen Kommission (IEC SC 65B/WG7) zwei technische Berichte (Technical Reports) vorbereitet:

– Ein technischer Bericht vom Typ 2 (Type 2 Technical Report) enthält Vorschläge für Korrekturen und Erweiterungen der definierten Programmiersprachen. Das Ziel ist es hierbei, diese nach Erreichen eines Konsens für die nächste Ausgabe der IEC 1131-3 zu berücksichtigen.

– Ein technischer Bericht vom Typ 3 (Type 3 Technical Report) vermittelt Hintergrundinformationen zu verschiedenen Eigenschaften der in dieser Norm definierten Sprachen und enthält Richtlinien für die SPS-Anwendung, Sprachimplementierung sowie die Ausstattung der Programmierungsumgebung.

Es ist geplant, die verabschiedeten Berichte, ins Deutsche übersetzt, zu veröffentlichen.

Fortsetzung Seite 2
und 154 Seiten EN

Deutsche Elektrotechnische Kommission im DIN und VDE (DKE)

Der Zusammenhang zwischen den in dieser Norm zitierten internationalen Publikationen und den entsprechenden Deutschen Normen ist, soweit im Deutschen Normenwerk vorhanden, nachstehend wiedergegeben:

Internationale Normen	Deutsche Normen
IEC 559	DIN IEC 559
IEC 617-12	DIN 40 900 Teil 12
IEC 617-13	DIN 40 900 Teil 13
IEC 848	DIN 40 719 Teil 6
ISO/IEC 646	DIN 66 003
ISO 8601	DIN 1355 Teil 1

Zitierte Normen

– in der Deutschen Fassung:

Siehe Anhang ZA (normativ)

– in nationalen Zusätzen:

DIN IEC 559	Binäre Gleitpunkt-Arithmetik für Mikroprozessor-Systeme (IEC 559 : 1989); Deutsche Fassung HD 592 S1 : 1991
DIN 1355 Teil 1	Zeit; Kalender, Wochennumerierung, Tagesdatum, Uhrzeit
DIN 40 719 Teil 6	Schaltungsunterlagen; Regeln für Funktionspläne; IEC 848, modifiziert
DIN 40 900 Teil 12	Graphische Symbole für Schaltungsunterlagen; Binäre Elemente; IEC 617 : 1991, modifiziert
DIN 40 900 Teil 13	Graphische Symbole für Schaltungsunterlagen; Analoge Informationsverarbeitung, Schaltzeichen und Kennzeichen
DIN 66 003	Informationsverarbeitung; 7-Bit-Code

Frühere Ausgaben

DIN 19 239: 05.83

Änderungen

Gegenüber DIN 19 239/05.83 wurden folgende Änderungen vorgenommen:

- Inhaltlich vollständig überarbeitet –

Internationale Patentklassifikation

G 05 B 019/05
G 06 F 009/45

DK 681.5 : 62-55 : 681.3.04

Deskriptoren: Industrielle Prozesse, Prozeßsteuerung und -regelung, Programmierbare Steuerung, Programmiersprache

Deutsche Fassung

Speicherprogrammierbare Steuerungen

**Teil 3: Programmiersprachen
(IEC 1131-3 : 1993)**

Programmable controllers — Part 3: Programming languages — (IEC 1131-3 : 1993)

Automates programmables — Partie 3: Langages de programmation — (CEI 1131-3 : 1993)

Diese Europäische Norm wurde von CENELEC am 1993-07-06 angenommen.

Die CENELEC-Mitglieder sind gehalten, die CEN/CENELEC-Geschäftsordnung zu erfüllen, in der die Bedingungen festgelegt sind, unter denen dieser Europäischen Norm ohne jede Änderung der Status einer nationalen Norm zu geben ist.

Auf dem letzten Stand befindliche Listen dieser nationalen Normen mit ihren bibliographischen Angaben sind beim Zentralsekretariat oder bei jedem CENELEC-Mitglied auf Anfrage erhältlich.

Diese Europäische Norm besteht in drei offiziellen Fassungen (Deutsch, Englisch, Französisch). Eine Fassung in einer anderen Sprache, die von einem CENELEC-Mitglied in eigener Verantwortung durch Übersetzung in seine Landessprache gemacht und dem Zentralsekretariat mitgeteilt worden ist, hat den gleichen Status wie die offiziellen Fassungen.

CENELEC-Mitglieder sind die nationalen elektrotechnischen Komitees von Belgien, Dänemark, Deutschland, Finnland, Frankreich, Griechenland, Irland, Island, Italien, Luxemburg, Niederlande, Norwegen, Österreich, Portugal, Schweden, Schweiz, Spanien und dem Vereinigten Königreich.

CENELEC

EUROPÄISCHES KOMITEE FÜR ELEKTROTECHNISCHE NORMUNG
European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique

Zentralsekretariat: rue de Stassart 35, B-1050 Brüssel

Vorwort

Der Text des Schriftstücks 65B(CO)85, wie ausgearbeitet vom sub-committee 65B: Devices, vom IEC technical committee 65: Industrial-process measurement and control, wurde im Februar 1992 in die IEC-CENELEC-Parallel-Abstimmung gegeben. Das Referenzdokument wurde von CENELEC am 6. Juli 1993 als EN 61131-3 genehmigt.

Nachstehende Daten wurden festgelegt:

- spätestes Datum der Veröffentlichung einer identischen nationalen Norm (dop): 1994-07-01
- spätestes Datum für die Zurückziehung entgegenstehender nationaler Normen (dow): 1994-07-01

Anhänge, die als "normativ" bezeichnet sind, gehören zum Norm-Inhalt. Anhänge, die als "informativ" bezeichnet sind, enthalten nur Informationen.

In dieser Norm sind die Anhänge A, B, C, D, E und ZA normativ und die Anhänge F, G und H informativ.

Anerkennungsnotiz

Der Text der Internationalen Norm IEC 1131-1 : 1993 wurde von CENELEC als Europäische Norm ohne irgendeine Abänderung genehmigt.

Inhalt

	Seite		Seite
1 Allgemeines	4	2.6.6 Kompatibilität von AS-Elementen	74
1.1 Anwendungsbereich	4	2.6.7 Anforderungen an die Normerfüllung	74
1.2 Normative Verweisungen	4	2.7 Konfigurationselemente	74
1.3 Begriffe	4	2.7.1 Konfigurationen, Ressourcen und Zugriffspfade	75
1.4 Übersicht und allgemeine Anforderungen	8	2.7.2 Tasks	77
1.4.1 Software-Modell	9	3 Textsprachen	84
1.4.2 Kommunikationsmodell	10	3.1 Gemeinsame Elemente	84
1.4.3 Programmiermodell	11	3.2 Die Sprache AWL (Anweisungsliste)	85
1.5 Normerfüllung	13	3.2.1 Anweisungen	85
1.5.1 SPS-Systeme	13	3.2.2 Operatoren, Modifizierer und Operanden	85
1.5.2 Programme	14	3.2.3 Funktionen und Funktionsbausteine	85
2 Gemeinsame Elemente	14	3.3 Sprache ST (Strukturierter Text)	87
2.1 Gebrauch der gedruckten Zeichen	14	3.3.1 Ausdrücke	87
2.1.1 Zeichensatz	14	3.3.2 Anweisungen	87
2.1.2 Bezeichner	15	4 Graphische Sprachen	90
2.1.3 Schlüsselwörter	15	4.1 Gemeinsame Elemente	90
2.1.4 Gebrauch von Leerzeichen	15	4.1.1 Darstellung von Linien und Blöcken	91
2.1.5 Kommentare	15	4.1.2 Flußrichtung in Netzwerken	91
2.2 Externe Darstellung von Daten	16	4.1.3 Auswertung von Netzwerken	91
2.2.1 Numerische Literale	16	4.1.4 Elemente der Ausführungssteuerung	92
2.2.2 Zeichenfolge-Literale	16	4.2 Sprache KOP (Kontaktplan)	93
2.2.3 Zeitlitterale	17	4.2.1 Stromschienen	93
2.3 Datentypen	18	4.2.2 Verbindungselemente und Zustände	93
2.3.1 Elementare Datentypen	18	4.2.3 Kontakte	95
2.3.2 Allgemeine (generische) Datentypen	19	4.2.4 Spulen	95
2.3.3 Abgeleitete Datentypen	20	4.2.5 Funktionen und Funktionsbausteine	95
2.4 Variablen	22	4.2.6 Reihenfolge der Netzwerk-Auswertung	95
2.4.1 Darstellung	22	4.3 Sprache FBS (Funktionsbaustein-Sprache)	95
2.4.2 Initialisierung	23	4.3.1 Allgemeines	95
2.4.3 Deklaration	23	4.3.2 Kombination von Elementen	97
2.5 Programm-Organisationseinheiten	27	4.3.3 Reihenfolge der Netzwerk-Auswertung	97
2.5.1 Funktionen	27	Anhang A (normativ) Spezifikationsmethode für Textsprachen	98
2.5.2 Funktionsbausteine	40	A.1 Syntax	98
2.5.3 Programme	54	A.2 Semantische Regeln	99
2.6 Elemente der Ablaufsprache (AS)	54	Anhang B (normativ) Formale Festlegungen der Sprachelemente	99
2.6.1 Allgemeines	54	B.0 Programmiermodell	99
2.6.2 Schritte	54	B.1 Gemeinsame Elemente	99
2.6.3 Transitionen (Übergänge)	55		
2.6.4 Aktionen	56		
2.6.5 Ablaufregeln	63		



	Seite		Seite
B.2 Sprache AWL (Anweisungsliste)	105	28 Standardfunktionen für Vergleich	38
B.3 Sprache ST (Strukturierter Text)	105	29 Standardfunktionen für Zeichenfolgen	39
Anhang C (normativ) Begrenzungszeichen und Schlüsselwörter	107	30 Funktionen für Datentypen der Zeit	40
Anhang D (normativ) Implementierungsabhängige Parameter	110	31 Funktionen für Datentypen der Aufzählung	40
Anhang E (normativ) Fehlerursachen	112	32 Beispiele für den Gebrauch der E/A-Parameter von Funktionsbausteinen	41
Anhang F (informativ) Beispiele	113	33 Eigenschaften der Funktionsbaustein-Deklarationen	44
F.1 Funktion WIEGEN	113	34 Bistabile Standard-Funktionsbausteine	50
F.2 Funktionsbaustein KOMMANDO_ÜBERWACHUNG	114	35 Standard-Funktionsbausteine Flankenerkennung ..	51
F.3 Funktionsbaustein VORW_RÜCKW_ÜBERWACHUNG	117	36 Standard-Funktionsbausteine Zähler	52
F.4 Funktionsbaustein STACK_INT	122	37 Standard-Funktionsbaustein Zeitgeber	52
F.5 Funktionsbaustein MIX_2_ZIEGEL	127	38 Standard-Funktionsbausteine Zeitgeber – Zeitdiagramme	53
F.6 Analogsignal-Verarbeitung	130	39 Eigenschaften der Programm-Deklarationen	54
F.7 Programm KIES	139	40 Schritt-Eigenschaften	56
F.8 Programm AGV	146	41 Transitionen und Transitionsbedingungen	57
Anhang G (informativ) Alphabetisches Stichwortverzeichnis	149	42 Deklaration von Aktionen	59
Anhang H (informativ) Software-Erfüllungstest	154	43 Zugehörigkeit Schritt/Aktion	61
Anhang ZA (normativ) Andere in dieser Norm zitierte internationale Publikationen mit den Verweisungen auf die entsprechen- den europäischen Publikationen	154	44 Eigenschaften Aktionsblock	62
		45 Bestimmungszeichen für Aktionen	62
		46 Kettenablauf	67
		47 Kompatible Eigenschaften bei Ablaufketten	74
		48 Mindestanforderungen der Normerfüllung bei AS ..	74
		49 Eigenschaften der Deklaration von Konfiguration und Ressource	77
Tabellen		50 Task-Eigenschaften	79
1 Zeichensatz-Eigenschaften	15	51 Beispiele für Anweisungsfelder	85
2 Bezeichner-Eigenschaften	15	52 Operatoren der Anweisungsliste (AWL)	86
3 Kommentar-Eigenschaft	16	53 Eigenschaften des Funktionsbaustein-Aufrufs in der Sprache AWL	86
4 Numerische Literale	16	54 Eingangsoperatoren von Standard-Funktions- bausteinen für AWL-Sprache	87
5 Eigenschaft der Zeichenfolge-Literale	17	55 Operatoren der Sprache ST	88
6 2-Zeichen-Kombinationen in Zeichenfolgen	17	56 Anweisungen der Sprache ST	89
7 Zeitdauer-Literal-Eigenschaften	17	57 Darstellung von Linien und Blöcken	92
8 Literale für Datum und Tageszeit	18	58 Graphische Elemente zur Ausführungssteuerung	94
9 Beispiele für Datum und Tageszeit	18	59 Stromschienen	95
10 Elementare Datentypen	18	60 Verbindungselemente	95
11 Hierarchie der allgemeinen Datentypen	19	61 Kontakte	96
12 Eigenschaften der Datentyp-Deklarationen	20	62 Spulen	96
13 Voreingestellte Anfangswerte	21		
14 Eigenschaften der Deklaration der Anfangs- werte von Datentypen	21	Bilder	
15 Eigenschaften der Präfixe für Speicherort und Größe bei direkt dargestellten Variablen	22	1 Software-Modell	9
16 Schlüsselwörter für Variablen-Deklaration	24	2 Kommunikationsmodell	11
17 Eigenschaften der Typzuweisung für Variable ..	25	3 Kombination der SPS-Sprachelemente	12
18 Eigenschaften der Zuweisung von Anfangs- werten für Variable	26	4 Beispiele für den Gebrauch von Funktionen	27
19 Graphische Negation von booleschen Signalen	28	5 Gebrauch der Formalparameter-Namen	28
20 Der Gebrauch des "EN"-Eingangs und "ENO"-Ausgangs	29	6 Beispiele für Deklarationen einer Funktion	30
21 Typangabe und überladene Funktionen	31	7 Beispiele für explizite Typumwandlung bei überladenen Funktionen	31
22 Eigenschaften der Funktionen zur Typumwandlung	33	8 Beispiele von expliziter Typumwandlung bei Funktionen mit Typangabe	32
23 Standardfunktionen mit einer numerischen Variablen	34	9 Beispiel für Instanziierung eines Funktionsbausteins	41
24 Arithmetische Standardfunktionen	35	10 Beispiele für Deklarationen von Funktions- bausteinen	43
25 Standard-Bitschiebe-Funktionen	36	11 Graphischer Gebrauch eines Funktions- baustein-Namens als eine Eingangs-, Eingangs/ Ausgangs- und externe Variable	45
26 Bitweise boolesche Standardfunktionen	36		
27 Standardfunktionen für Auswahl	37		

	Seite		Seite
Bilder		18 AS-Fehler	72
12 Beispiele für den Gebrauch von Eingangs/ Ausgangsvariablen	48	19 Graphisches Beispiel für eine Konfiguration	75
13 Beispiel für den Gebrauch der Semaphore	49	20 Beispiele für Deklarations-Eigenschaften der CONFIGURATION und RESOURCE	78
14 Funktionsbaustein ACTION_CONTROL – Externe Schnittstelle	63	21 Synchronisation von Funktionsbausteinen bei expliziten und impliziten Task-Zuordnungen	82
15 Funktionsbausteinrumpf ACTION_CONTROL	64	22 Beispiel für die Anweisung EXIT	90
16 Beispiele für Aktionssteuerung	65	23 Beispiel eines Rückkopplungspfads	93
17 AS-Ablaufregeln	71	24 Beispiele für boolesches ODER	97

1 Allgemeines

1.1 Anwendungsbereich

Dieser Teil der IEC 1131 gilt für die Darstellung der Programmiersprachen für Speicherprogrammierbare Steuerungen, wie sie in Teil 1 der IEC 1131-1 definiert sind, auf Drucker und Bildschirm, dabei ist der ISO/IEC-646-Zeichensatz anzuwenden. Graphische und semigraphische Darstellungen der Sprachelemente, die in diesem Teil definiert sind, sind zwar zulässig, aber nicht in diesem Teil definiert.

Die Funktionen der Programmeingabe, des Tests, der Überwachung, des Betriebssystems usw. sind in Teil 1 der IEC 1131 festgelegt.

1.2 Normative Verweisungen

Die folgenden Normen enthalten Festlegungen, die durch Verweisung in diesem Text Bestandteil der vorliegenden Norm sind. Zum Zeitpunkt der Veröffentlichung dieser Norm gelten die angegebenen Ausgaben. Alle Normen unterliegen der Überarbeitung. Vertragspartner, deren Vereinbarungen auf diesem Teil der IEC 1131 basieren, werden gebeten, nach Möglichkeit die neuesten Ausgaben der nachfolgend aufgeführten Normen anzuwenden. Mitglieder von IEC und ISO führen Verzeichnisse der gültigen Internationalen Normen.

IEC 50	International Electrotechnical Vocabulary (IEV)
IEC 559 : 1989	Binary floating-point arithmetic for microprocessor systems
IEC 617-12 : 1991	Graphical symbols for diagrams – Part 12: Binary logic elements
IEC 617-13 : 1978	Graphical symbols for diagrams – Part 13: Analogue elements
IEC 848 : 1988	Preparation of function charts for control systems
ISO/AFNOR : 1989	Dictionary of computer science, ISBN 2-12-486911-6
ISO/IEC 646 : 1991	Information technology – ISO 7-bit coded character set for information processing interchange
ISO 8601 : 1988	Data elements and interchange formats – Information interchange – Representations of dates and times
ISO 7185 : 1990	Information technology – Programming languages – Pascal
ISO 7498 : 1984	Information processing systems – Open systems interconnection – Basic reference model

1.3 Begriffe

Für diesen Teil der IEC 1131 sind die folgenden Definitionen anzuwenden. Definitionen, die für alle Teile dieser Norm gelten, sind in Teil 1 angegeben.

ANMERKUNG 1: Begriffe, die in diesem Abschnitt definiert sind, *sind kursiv angegeben*, wenn sie innerhalb von Definitionen auftreten.

ANMERKUNG 2: Die Angabe "(ISO)", die einer Definition folgt, gibt an, daß die Definition aus dem ISO/AFNOR *Dictionary of Computer Science* entnommen ist.

ANMERKUNG 3: Das ISO/AFNOR *Dictionary of Computer Science* und das *International Electrotechnical Vocabulary* sollten für Begriffe zu Rate gezogen werden, die nicht in dieser Norm definiert sind.

1.3.1 Absolute Zeit (absolute time)

Die Kombination von Tageszeit und Datum.

1.3.2 Zugriffspfad (access path)

Die Verknüpfung eines symbolischen Namens mit einer Variablen zum Zweck der offenen Kommunikation.

1.3.3 Aktion (action)

Eine boolesche Variable oder eine Sammlung von Operationen, die zusammen mit einer zugehörigen Kontrollstruktur ausgeführt wird, wie in 2.6.4 definiert ist.

1.3.4 Aktionsblock (action block)

Graphisches Sprachelement, das eine boolesche Eingangsvariable benutzt, um den Wert einer booleschen Ausgangsvariablen oder die Freigabebedingung für eine *Aktion* zu bestimmen; dies erfolgt gemäß einer vorbestimmten Kontrollstruktur, wie in 2.6.4.5 definiert ist.

1.3.5 Aggregat (aggregate)

Strukturierte Sammlung von Datenobjekten, die einen *Datentyp* bilden (ISO).

1.3.6 Argument (argument)

Synonym für *Eingangs-* oder *Ausgangsparameter*.

1.3.7 Feld (array)

Aggregat, das aus Datenobjekten mit identischen Attributen besteht, von denen jedes eindeutig mit einem *Index* (subscript) referenziert werden kann (ISO).

1.3.8 Zuweisung (assignment)

Mechanismus, um einer Variablen oder einem *Aggregat* einen Wert zu geben.

1.3.9 Basisbezogene Zahl (based number)

Zahl, die in einer festgelegten Basis ungleich zehn dargestellt ist.

1.3.10 Bistabiler Funktionsbaustein (bistable function block)

Funktionsbaustein mit zwei stabilen Zuständen, gesteuert durch einen oder mehr Eingänge.

1.3.11 Bitfolge (bit string)

Ein Datenelement, bestehend aus einem oder mehreren Bits.

1.3.12 Rumpf (body)

Der Teil einer *Programm-Organisationseinheit*, der die Operationen festlegt, die mit den deklarierten Operanden einer *Programm-Organisationseinheit* auszuführen sind, wenn ihre Ausführung aufgerufen wird.

1.3.13 Aufruf (call)

Ein Sprachkonstrukt zum *Aufruf* der Ausführung einer *Funktion* oder eines *Funktionsbausteins*.

1.3.14 Zeichenfolge (character string)

Ein *Aggregat*, das aus einer geordneten Folge von Zeichen besteht.

1.3.15 Kommentar (comment)

Eine Sprachkonstruktion zur Einbindung von Text in ein Programm, ohne Auswirkung auf die Programmausführung zu haben (ISO).

1.3.16 Kompilieren (compile)

Übersetzen einer *Programm-Organisationseinheit* oder einer *Datentyp-Festlegung* in ihr Äquivalent in Maschinensprache oder einer Zwischenform.

1.3.17 Konfiguration (configuration)

Ein Sprachelement, das einem *SPS-System* entspricht, wie es in IEC 1131-1 definiert ist.

1.3.18 Zähler-Baustein (counter function block)

Ein *Funktionsbaustein*, der einen Wert entsprechend der Anzahl der erkannten Wechsel an einem oder mehreren festgelegten *Eingängen* akkumuliert.

1.3.19 Datentyp (data type)

Eine Menge von Werten zusammen mit den auf dieser Menge zulässigen Operationen (ISO).

1.3.20 Datum und Uhrzeit (date and time)

Das Datum im Jahr und die Tageszeit, gemäß ISO 8601 dargestellt.

1.3.21 Deklaration (declaration)

Mechanismus zur Festlegung der Definition eines *Sprachelements*. Eine Deklaration umfaßt normalerweise die Verbindung eines Bezeichners mit dem Sprachelement und die Zuordnung von Attributen, wie *Datentypen* und Algorithmen.

1.3.22 Begrenzungszeichen (delimiter)

Zeichen oder eine Zeichenkombination, zur Trennung von *Sprachelementen*.

1.3.23 Direkte Darstellung (direct representation)

Mittel zur Darstellung einer Variablen in einem *SPS-Programm*, von der eine herstellerspezifische Entsprechung zu einem physikalischen oder *logischen Speicherort* direkt bestimmt werden kann.

1.3.24 Doppelwort (double word)

Datenelement, das 32 Bits enthält.

1.3.25 Auswertung (evaluation)

Vorgang der Ermittlung eines Wertes für einen Ausdruck oder eine *Funktion* oder für die *Ausgänge* eines Netzwerks oder *Funktionsbausteins* während der Programmbearbeitung.

1.3.26 Element zur Ausführungssteuerung (execution control element)

Ein *Sprachelement*, das den Fluß der Programmausführung steuert.

1.3.27 Fallende Flanke (falling edge)

Wechsel von 1 nach 0 bei einer booleschen Variablen.

1.3.28 Funktion (function)

Programm-Organisationseinheit, die bei der Ausführung genau ein Datenelement liefert (das ein Multi-Wert sein darf, z. B. ein *Feld* oder eine *Struktur*) und deren *Aufruf* in einer Textsprache als ein *Operand* in einem Ausdruck benutzt werden kann.

1.3.29 Funktionsbaustein-Instance (function block instance)

Instance eines *Funktionsbaustein-Typs*.

1.3.30 Funktionsbaustein-Typ (function block type)

SPS-*Sprachelement*, bestehend aus: (i) Der Definition einer Datenstruktur, geteilt in Eingangs-, Ausgangs- und interne Variablen und (ii) einer Menge von Operationen, die mit den Elementen der Datenstruktur durchgeführt werden, wenn eine *Instance* des Funktionsbaustein-Typs *aufgerufen* wird.

1.3.31 Funktionsbaustein-Sprache (function block diagram)

Ein oder mehrere Netzwerke aus graphisch dargestellten *Funktionen*, *Funktionsbausteinen*, Datenelementen, *Marken* und Verbindungselementen.

1.3.32 Allgemeiner Datentyp (generic data type)

Datentyp, der mehr als einen Typ von Daten darstellt, wie in 2.3.2 festgelegt ist.

1.3.33 Globaler Geltungsbereich (global scope)

Geltungsbereich einer Deklaration, die anwendbar ist auf alle Programm-Organisationseinheiten innerhalb einer *Ressource* oder *Konfiguration*.

1.3.34 Globale Variable (global variable)

Variable, deren *Geltungsbereich* *global* ist.

1.3.35 Hierarchische Adressierung (hierarchical addressing)

Direkte Darstellung eines Datenelements als Teil einer physikalischen oder logischen Hierarchie; z. B. der Anschlußpunkt einer Baugruppe (Modul), die in einem Rahmen steckt, der wiederum in einem Schrank eingebaut ist usw.

1.3.36 Bezeichner (identifier)

Kombination von Buchstaben, Zahlen und Unterstrich-Zeichen, wie in 2.1.2 festgelegt, die mit einem Buchstaben oder einem Unterstrich beginnt und die ein *Sprachelement* bezeichnet.

1.3.37 Anfangswert (initial value)

Der Wert, der einer Variablen beim Systemanlauf zugewiesen wird.

1.3.38 Eingangsparameter (input parameter)

Parameter, der benutzt wird, um einer *Programm-Organisationseinheit* ein Argument zu übergeben.

1.3.39 Instance (instance)

Individuelles, benanntes Exemplar einer Datenstruktur, verknüpft mit einem *Funktionsbaustein-Typ* oder *Programm-Typ*, das von einem *Aufruf* der zugehörigen Operationen bis zum nächsten erhalten bleibt.

1.3.40 Instance-Name (instance name)

Bezeichner, verknüpft mit einer festgelegten *Instance*.

1.3.41 Instanziierung (instantiation)

Die Erzeugung einer *Instance*.

1.3.42 Ganzzahliges Literal (integer literal)

Literal, das direkt einen Wert vom Typ SINT, INT, DINT, LINT, BOOL, BYTE, WORD, DWORD oder LWORD darstellt, wie in 2.3.1 definiert.

1.3.43 Aufruf (invocation)

Der Einleitungsvorgang zur Ausführung von Operationen, die in einer *Programm-Organisationseinheit* festgelegt sind.

1.3.44 Schlüsselwort (keyword)

Lexikalische Einheit, die ein *Sprachelement* charakterisiert, z. B. "IF".

1.3.45 Marke (label)

Eine Sprachkonstruktion, die eine Anweisung, ein Netzwerk oder eine Gruppe von Netzwerken bezeichnet und die einen *Bezeichner* einschließt.

1.3.46 Sprachelement (language element)

Jedes Element, das durch ein Symbol auf der linken Seite der Produktionsregeln in der formalen Festlegung identifiziert wird, die in Anhang B dieses Teils der IEC 1131 angegeben ist.

1.3.47 Literal (literal)

Lexikalische Einheit, die direkt einen Wert darstellt (ISO).

1.3.48 Lokaler Geltungsbereich (local scope)

Der *Geltungsbereich* einer *Deklaration* oder *Marke*, die nur für die *Programm-Organisationseinheit* gilt, in der die Deklaration oder Marke auftritt.

1.3.49 Logischer Speicherort (logical location)

Der Speicherort einer *hierarchisch adressierten* Variablen in einem Schema, das eine Relation zu der physikalischen Struktur der Eingänge, Ausgänge und des Speichers der SPS besitzen darf oder auch nicht.

1.3.50 Lange Realzahl (long real)

Realzahl, die in einem *Langwort* dargestellt ist.

1.3.51 Langwort (long word)

64-Bit-Datenelement.

1.3.52 Anwenderdatenspeicher (memory)

Eine funktionale Einheit, in die das Anwendungsprogramm Daten speichern kann und aus der es die gespeicherten Daten zurückgewinnen kann.

1.3.53 Bezeichnetes Element (named element)

Element einer *Struktur*, das durch den zugehörigen *Bezeichner* benannt ist.

1.3.54 Aus/Einschaltverzögerung (off/on-delay timer function block)

Funktionsbaustein, der die *fallende* (*steigende*) *Flanke* eines booleschen *Eingangs* um eine festgelegte Dauer verzögert.

1.3.55 Operand (operand)

Sprachelement, mit dem eine Operation durchgeführt wird.

1.3.56 Operator (operator)

Symbol, das die Aktion darstellt, mit der eine Operation durchgeführt wird.

1.3.57 Ausgangsparameter (output parameter)

Parameter, der benutzt wird, um das (die) Ergebnis(se) einer *Auswertung* einer *Programm-Organisationseinheit* zurückzugeben.

1.3.58 Überladen (overloaded)

Die Fähigkeit, bei einer Operation oder einer *Funktion* Operationen mit Daten verschiedenen Typs auszuführen, wie in 2.5.1.4 festgelegt.

1.3.59 Stromfluß (power flow)

Der symbolische Fluß des elektrischen Stroms in einem Kontaktplan, benutzt, um das Fortschreiten eines logischen Algorithmus zu beschreiben.

1.3.60 Programmieren (program)

Anwenderprogramme entwerfen, schreiben und testen.

1.3.61 Programm-Organisationseinheit (program organization unit)

Eine *Funktion*, ein *Funktionsbaustein* oder ein *Programm*.

ANMERKUNG: Dieser Begriff kann sich sowohl auf *Typen* als auch auf *Instanzen* beziehen.

1.3.62 Reelles Literal (real literal)

Literal, das Daten vom Typ REAL oder LREAL darstellt.

1.3.63 Ressource (resource)

Sprachelement, das einer "Signalverarbeitungsfunktion" und ihrer "Mensch-Maschine-Schnittstelle" und den "Funktionen der Sensor- und Aktor-Schnittstelle" entspricht, sofern diese vorhanden sind, wie in IEC 1131-1 definiert.

1.3.64 Gepufferte Daten (retentive data)

Daten, die in einer Weise gespeichert sind, daß ihr Wert nach einer Folge von Spannungsausfall/Wiederkehr unverändert erhalten bleibt.

1.3.65 Rücksprung (return)

Sprachkonstruktion innerhalb einer *Programm-Organisationseinheit*, die ein Ende einer Ausführungsfolge in dieser Einheit bezeichnet.

1.3.66 Steigende Flanke (rising edge)

Der Wechsel von 0 nach 1 einer booleschen Variablen.

1.3.67 Geltungsbereich (scope)

Der Teil eines *Sprachelements*, innerhalb dessen eine *Deklaration* oder eine *Marke* gilt.

1.3.68 Semantik (semantics)

Die Beziehungen zwischen den symbolischen Elementen einer Programmiersprache und ihren Bedeutungen, ihrer Auslegung und Anwendung.

1.3.69 Semigraphische Darstellung (semigraphic representation)

Darstellung der graphischen Information durch die Anwendung eines begrenzten Zeichensatzes.

1.3.70 Einzel-Datenelement (single data element)

Datenelement, das aus einem einzelnen Wert besteht.

1.3.71 Schritt (step)

Situation, in der das Verhalten einer *Programm-Organisationseinheit* in Bezug auf ihre *Ein- und Ausgänge* einer Menge von Regeln folgt, die durch die zugehörigen *Aktionen* des Schritts definiert sind.

1.3.72 Strukturierter Datentyp (structured data type)

Zusammengesetzter Datentyp (*Aggregat*), der unter Verwendung einer Deklaration STRUCT oder FUNCTION_BLOCK deklariert wurde.

1.3.73 Indizierung (subscripting)

Mechanismus zur Adressierung eines *Feldelementes* durch einen Feldverweis und einen oder mehrere Ausdrücke, die bei Auswertung die Position des Elementes bezeichnen.

1.3.74 Symbolische Darstellung (symbolic representation)

Die Anwendung von *Bezeichnern*, um Variable mit Namen zu versehen.

1.3.75 Task (task)

Element zur Ausführungssteuerung, das für eine periodische oder getriggerte Ausführung einer Gruppe von zugehörigen *Programm-Organisationseinheiten* sorgt.

1.3.76 Zeitliteral (time literal)

Literal, das Daten vom Typ TIME, DATE, TIME_OF_DAY oder DATE_AND_TIME darstellt.

1.3.77 Transition (transition)

Bedingung, bei der die Steuerung von einem oder mehreren Vorgängerschritten zu einem oder mehreren Nachfolgerschritten entlang einer gerichteten Verbindung übergeht.

1.3.78 Vorzeichenlose ganze Zahl (unsigned integer)

Ganzzahliges Literal, das kein führendes Plus- (+) oder Minus- (-) Zeichen enthält.

1.3.79 Verdrahtetes ODER (wired OR)

Konstruktion zur Ausführung der booleschen ODER-Funktion im Kontaktplan durch Verbinden der rechten Enden von horizontalen mit vertikalen Verbindungen.

1.4 Übersicht und allgemeine Anforderungen

Dieser Teil der IEC 1131 legt die Syntax und Semantik einer vereinheitlichten Reihe von Programmiersprachen für Speicherprogrammierbare Steuerungen (SPS) fest. Diese umfassen zwei Textsprachen, AWL (Anweisungsliste) und ST (Strukturierter Text), und zwei graphische Sprachen, KOP (Kontaktplan) und FBS (Funktionsbaustein-Sprache).

Die Elemente der Ablaufsprache (AS) sind zur Strukturierung der internen Organisation von SPS-Programmen und -Funktionsbausteinen definiert. Außerdem sind *Konfigurationselemente* definiert, die zur Installation von SPS-Programmen in die SPS-Systeme dienen.

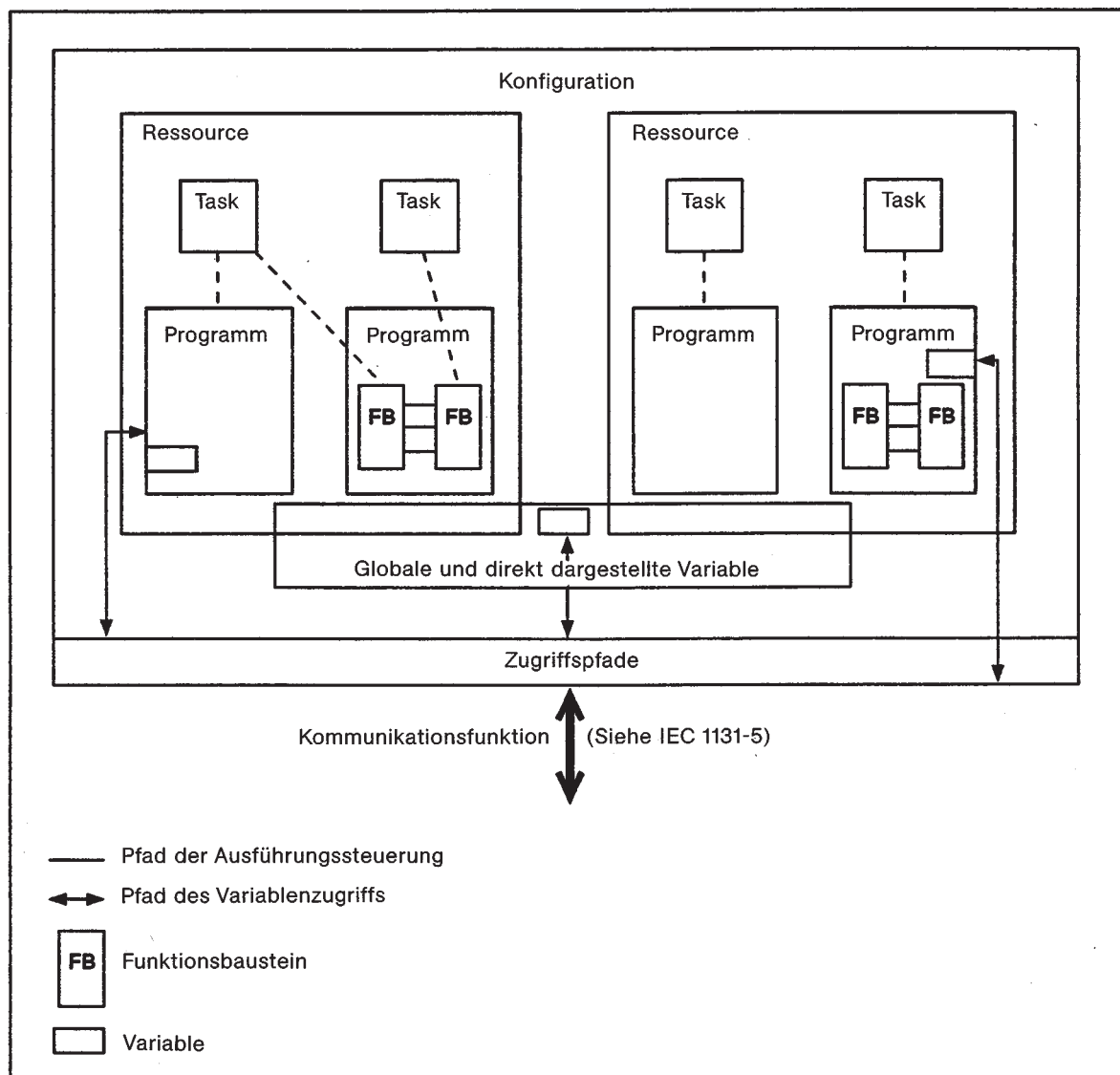
Zusätzlich sind Mittel definiert, die die Kommunikation zwischen Speicherprogrammierbaren Steuerungen und anderen Komponenten von automatisierten Systemen ermöglichen.

Die Elemente der Programmiersprachen in diesem Teil können in einer interaktiven Programmierumgebung angewendet werden. Die Festlegung derartiger Umgebungen gehört nicht zum Geltungsbereich dieses Teils; eine solche Umgebung muß jedoch eine Programmdokumentation in Text- oder Graphik-Formaten erzeugen können, wie sie in diesem Teil festgelegt sind.

Der Stoff in diesem Teil ist von "unten nach oben" ("bottom up") angelegt, d. h., die einfachen Sprachelemente werden zuerst dargestellt, um die Vorwärtsverweise im Text gering zu halten. Der Rest dieses Abschnitts bietet eine Übersicht über den Stoff, der in diesem Teil dargestellt ist, und enthält einige allgemeine Anforderungen.

1.4.1 Software-Modell

Die grundlegenden Hochsprachenelemente und ihre Beziehungen untereinander sind in Bild 1 veranschaulicht. Sie bestehen aus Elementen, die durch Anwendung der Sprachen *programmiert* werden, die in diesem Teil definiert sind; d. h. *Programme* und *Funktionsbausteine*, des weiteren *Konfigurationselemente* wie *Konfigurationen*, *Ressourcen*, *Tasks*, *globale Variablen* und *Zugriffspfade*, die zur Installation von SPS-Programmen in die SPS-Systeme dienen.



ANMERKUNG 1: Dieses Bild dient nur zur Veranschaulichung. Die graphische Darstellung ist nicht normativ.

ANMERKUNG 2: In einer Konfiguration mit einer einzigen Ressource muß die Ressource nicht explizit dargestellt werden.

Bild 1: Software-Modell

Eine *Konfiguration* ist das Sprachelement, das einem SPS-System entspricht, wie es in IEC 1131-1 definiert ist. Eine *Ressource* entspricht einer "Signalverarbeitungsfunktion" und ihrer "Mensch-Maschine-Schnittstelle" und den Funktionen der "Sensor- und Aktor-Schnittstelle", sofern vorhanden, wie in 1131-1 definiert. Eine *Konfiguration* enthält eine oder mehrere *Ressourcen*, von denen jede ein oder mehrere *Programme* enthält, die unter der Steuerung von null oder mehr *Tasks* ausgeführt werden. Ein *Programm* kann null oder mehr *Funktionsbausteine* oder andere Sprachelemente enthalten, wie in diesem Teil definiert ist. *Konfigurationen* und *Ressourcen* können mit Hilfe der Funktionen "Bedienerschnittstelle", "Programmierung, Testen und Beobachten" oder "Betriebssystem" gestartet oder gestoppt werden, wie in IEC 1131-1 definiert ist. Das Starten einer *Konfiguration* muß die Initialisierung ihrer *globalen Variablen* bewirken, gemäß den Regeln, die in 2.4.2 angegeben sind, gefolgt vom Starten aller *Ressourcen* in der *Konfiguration*. Das Starten einer *Ressource* muß die Initialisierung aller Variablen in der *Ressource* bewirken, gefolgt von der Freigabe aller *Tasks* in der *Ressource*. Das Stoppen einer *Ressource* muß das Sperren aller ihrer *Tasks* bewirken, während das Stoppen einer *Konfiguration* das Stoppen aller ihrer *Ressourcen* bewirken muß. Die Mechanismen zur Steuerung von *Tasks* sind in 2.7.2 definiert, während die Mechanismen zum Starten und Stoppen von *Konfigurationen* und *Ressourcen* über Kommunikationsfunktionen in IEC 1131-5 definiert sind.

Programme, Ressourcen, globale Variable, Zugriffspfade (und ihre zugehörigen Zugriffsberechtigungen) und Konfigurationen können durch die "Kommunikationsfunktion" geladen und gelöscht werden, die in IEC 1131-1 definiert ist. Das Laden und Löschen einer Konfiguration oder Ressource muß gleichbedeutend mit dem Laden und Löschen aller der Elemente sein, die sie enthält.

Zugriffspfade und ihre zugehörigen Zugriffsberechtigungen sind in 2.7.1 definiert.

Die Abbildung der Sprachelemente, die in diesem Unterabschnitt definiert sind, auf Kommunikationsobjekte ist in IEC 1131-5 definiert.

1.4.2 Kommunikationsmodell

Bild 2 veranschaulicht die Wege, über die die Werte von Variablen zwischen Software-Elementen ausgetauscht werden können.

Wie in Bild 2a gezeigt, können die Variablenwerte innerhalb eines Programms direkt ausgetauscht werden durch die Verbindung des Ausgangs des einen Programmelements mit dem Eingang des anderen. Diese Verbindung wird in den graphischen Sprachen explizit gezeigt und ist in den Textsprachen implizit angegeben.

Variablenwerte können zwischen Programmen derselben Konfiguration über globale Variablen ausgetauscht werden, wie es die Variable "x" in Bild 2b veranschaulicht. Diese Variable muß als GLOBAL in der Konfiguration und als EXTERNAL im Programm deklariert werden, wie in 2.4.3 festgelegt ist.

Wie in Bild 2c veranschaulicht, können die Werte von Variablen zwischen Teilen eines Programms, zwischen Programmen in derselben oder verschiedenen Konfigurationen oder zwischen einem SPS-Programm und einem Nicht-SPS-System ausgetauscht werden, indem die Kommunikations-Funktionsbausteine verwendet werden, die in IEC 1131-5 definiert und in 2.5.2.3.5 beschrieben sind. Außerdem können Speicherprogrammierbare Steuerungen oder Nicht-SPS-Systeme Daten übertragen, die durch die Zugriffspfade zur Verfügung gestellt werden, wie es in Bild 2d veranschaulicht ist, indem die Mechanismen benutzt werden, die in IEC 1131-5 definiert sind.

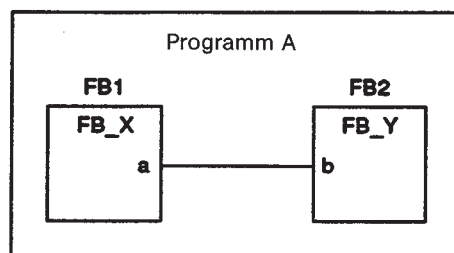


Bild 2a: Datenfluß-Verbindung innerhalb eines Programms

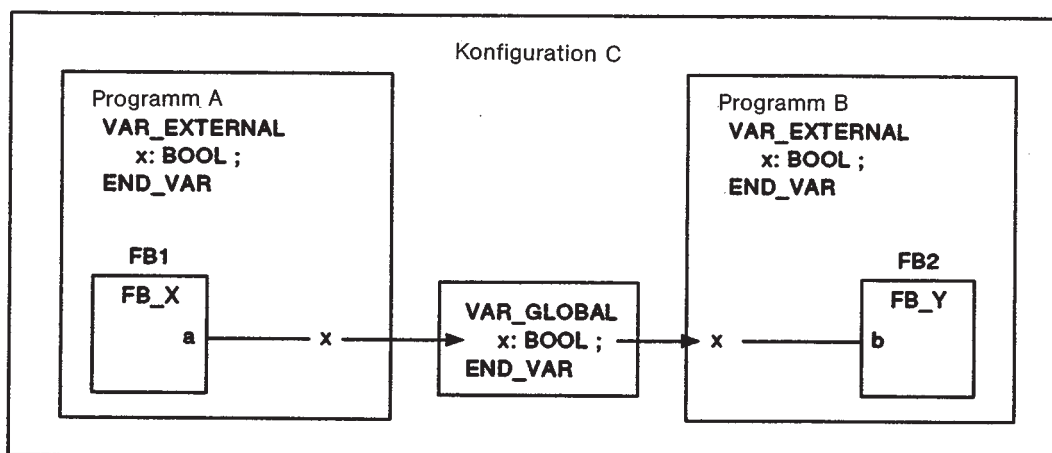


Bild 2b: Kommunikation über GLOBALE Variable

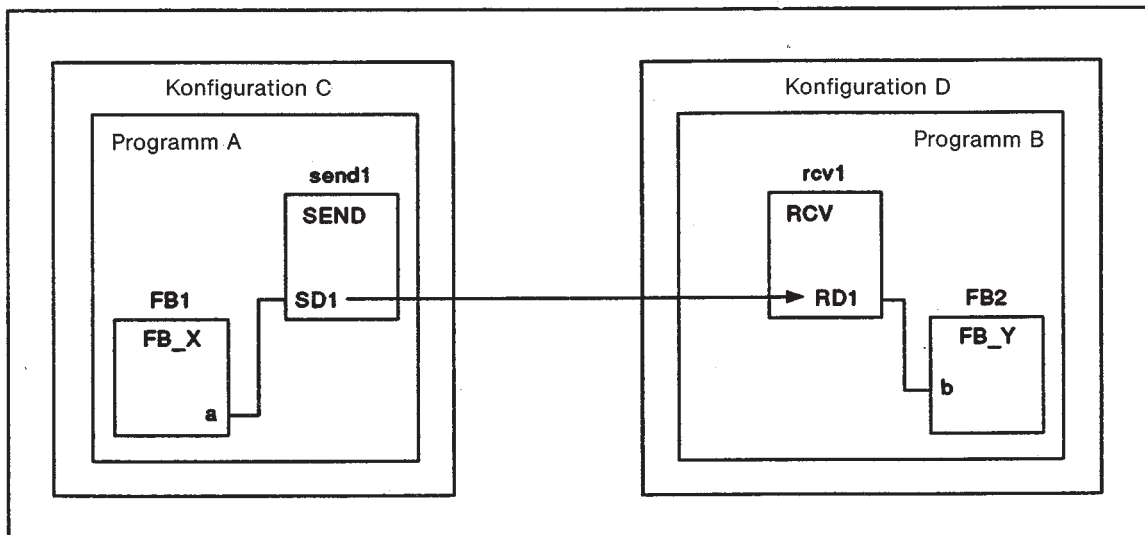


Bild 2c: Kommunikation über Funktionsbausteine

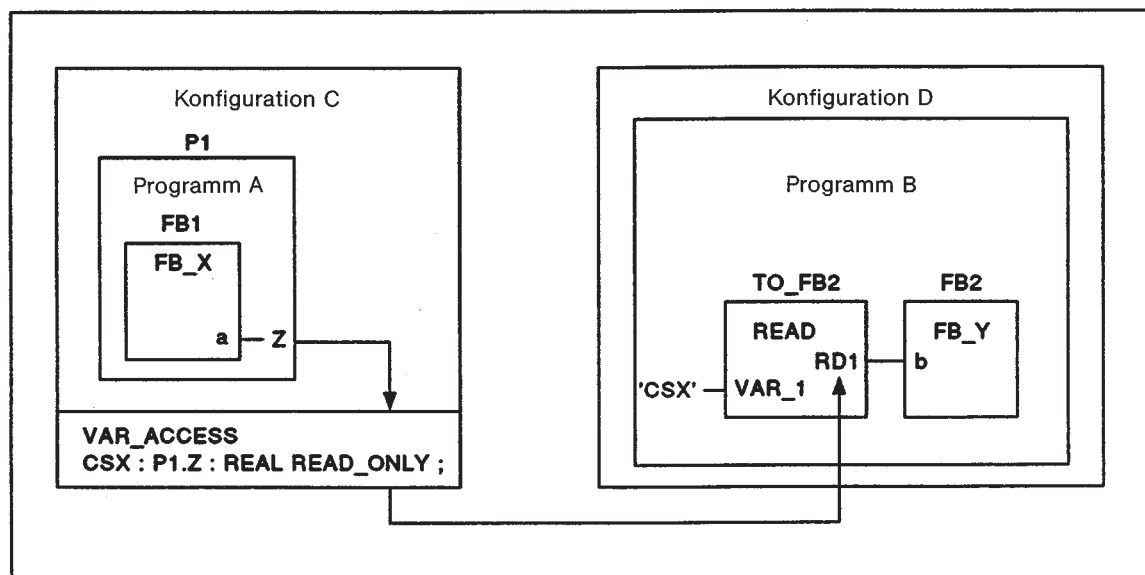


Bild 2d: Kommunikation über Zugriffspfade

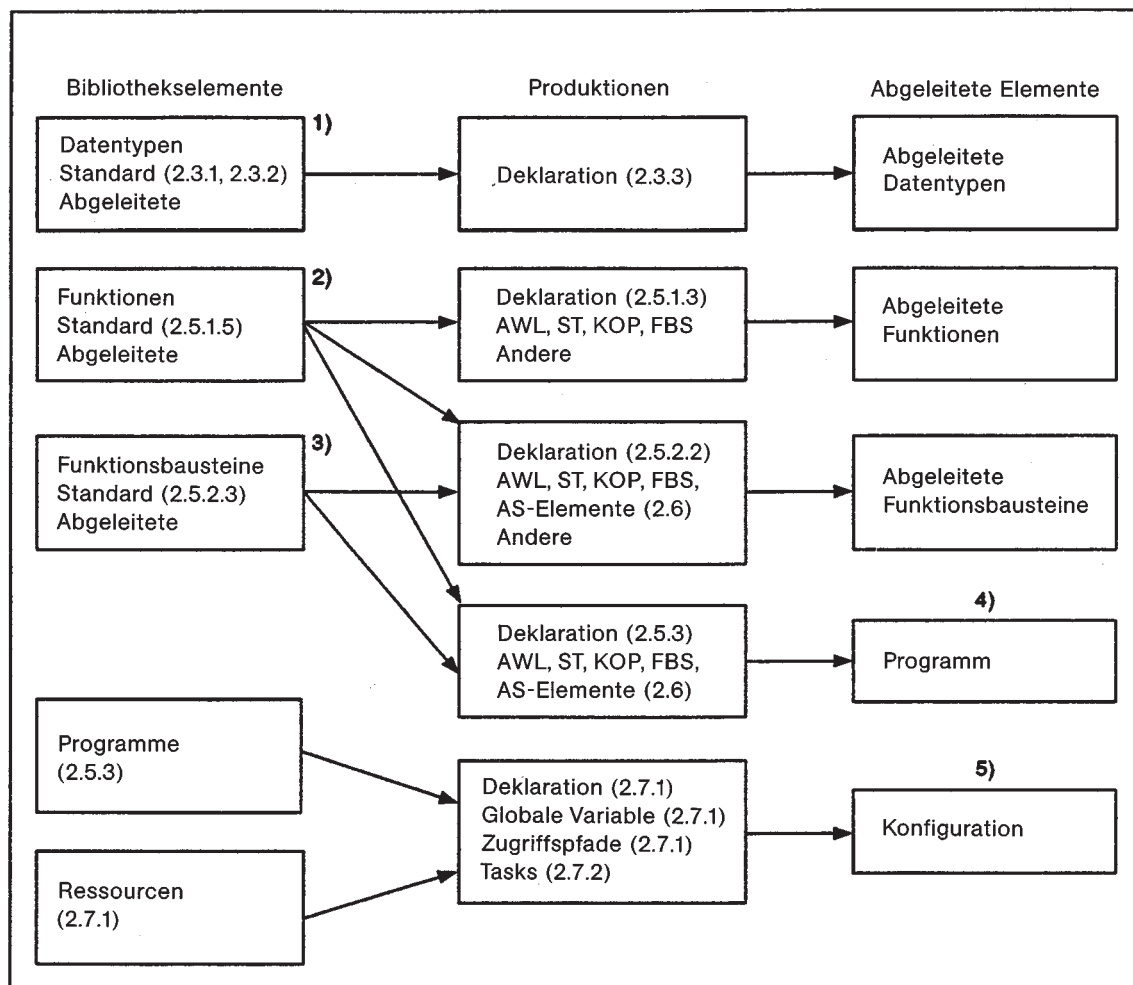
Bild 2: Kommunikationsmodell

- ANMERKUNG 1: Dieses Bild dient nur zur Veranschaulichung. Die graphische Darstellung ist nicht normativ.
- ANMERKUNG 2: In diesen Beispielen sollen die Konfigurationen C und D mit je einer einzelnen Ressource betrachtet werden.
- ANMERKUNG 3: Die Einzelheiten der Kommunikations-Funktionsbausteine sind in diesem Bild nicht gezeigt. Siehe 2.5.2.3.5 und IEC 1131-5.
- ANMERKUNG 4: Wie in 2.7 festgelegt, können die Zugriffspfade auf direkt dargestellten Variablen, globalen Variablen oder Programm-Eingangs-, Ausgangs- oder internen Variablen definiert werden.
- ANMERKUNG 5: IEC 1131-5 legt die Mittel fest, mit denen sowohl SPS- als auch Nicht-SPS-Systeme Zugriffspfade zum Lesen und Schreiben von Variablen benutzen können.

1.4.3 Programmiermodell

Die Elemente der SPS-Sprachen und die Unterabschnitte, in denen sie in diesem Teil erscheinen, sind folgendermaßen eingeteilt:

- Datentypen (2.3)
- Programm-Organisationseinheiten (2.5)
 - Funktionen (2.5.1)
 - Funktionsbausteine (2.5.2)



ANMERKUNG 1: Die geklammerten Zahlen 1) bis 5) verweisen auf die entsprechenden Absätze in Unterabschnitt 1.4.3.

ANMERKUNG 2: *Datentypen* werden in allen Produktionen benutzt. Die entsprechenden Verbindungen sind der Klarheit wegen in diesem Bild weggelassen.

Bild 3: Kombination der SPS-Sprachelemente
KOP – Kontaktplan (4.2)
FBS – Funktionsbaustein-Sprache (4.3)
AWL – Anweisungsliste (3.2)
ST – Strukturierter Text (3.3)
Andere: Andere Programmiersprachen (1.4.3)

Programme (2.5.3)
Ablaufsprachen-(AS-)Elemente (2.6)
Konfigurationselemente (2.7)
Globale Variable (2.7.1)
Ressourcen (2.7.1)
Tasks (2.7.2)
Zugriffspfade (2.7.1)

Wie in Bild 3 gezeigt, muß die Kombination dieser Elemente den folgenden Regeln folgen:

- 1) Abgeleitete *Datentypen* müssen deklariert werden, wie in 2.3.3 festgelegt ist, indem die Standard-Datentypen, die in 2.3.1 und 2.3.2 festgelegt sind, und alle vorher abgeleiteten Datentypen benutzt werden.
- 2) Abgeleitete *Funktionen* können deklariert werden, wie in 2.5.1.3 festgelegt ist, indem Standard- und abgeleitete Datentypen, Standardfunktionen, die in 2.5.1.5 definiert sind, und alle vorher abgeleiteten Funktionen benutzt werden. Diese Deklaration muß die Mechanismen benutzen, die für die Sprachen AWL, ST, KOP oder FBS definiert sind.

- 3) Abgeleitete *Funktionsbausteine* können deklariert werden, wie in 2.5.2.2 festgelegt ist, indem Standard- oder abgeleitete Datentypen und Funktionen, Standard-Funktionsbausteine, definiert in 2.5.2.3, und alle vorher abgeleiteten Funktionsbausteine benutzt werden. Diese Deklaration muß die Mechanismen benutzen, die für die Sprachen AWL, ST, KOP oder FBS definiert sind, und kann die Elemente der Ablaufsprache (AS) einschließen, die in 2.6 definiert ist.
- 4) Ein *Programm* muß deklariert werden, wie in 2.5.3 festgelegt ist, indem Standard- oder abgeleitete Datentypen, Funktionen und Funktionsbausteine benutzt werden. Diese Deklaration muß Mechanismen benutzen, die für die Sprachen AWL, ST, KOP oder FBS definiert sind, und kann die Elemente der Ablaufsprache (AS) einschließen, die in 2.6. definiert ist.
- 5) *Programme* können zu *Konfigurationen* kombiniert werden, indem die Elemente benutzt werden, die in 2.7 definiert sind; d. h. *globale Variable*, *Ressourcen*, *Tasks* und *Zugriffspfade*.

Der Hinweis auf "vorher abgeleitete" Datentypen, Funktionen und Funktionsbausteine in den obigen Regeln soll bedeuten, daß, wenn ein derartiges abgeleitetes Element einmal deklariert wurde, seine Definition z. B. in einer "Bibliothek" mit abgeleiteten Elementen zur Verfügung steht, um sie für weitere Ableitungen zu benutzen. Deshalb darf die Deklaration eines abgeleiteten Elementtyps nicht in der Deklaration eines anderen abgeleiteten Elementtyps enthalten sein.

Eine andere Programmiersprache als eine von denen, die in dieser Norm definiert sind, darf in der Deklaration von *Funktionen* oder *Funktionsbausteinen* benutzt werden. Die Mittel, mit denen ein Anwenderprogramm, das in einer der Sprachen geschrieben ist, die in dieser Norm definiert sind, die Ausführung einer derartigen abgeleiteten Funktion oder eines Funktionsbausteins aufruft und die Zugriffe auf die zugehörigen Daten ausführt, müssen so beschaffen sein, wie in dieser Norm definiert ist.

1.5 Normerfüllung

Dieser Unterabschnitt definiert die Anforderungen, die von SPS-Systemen und -Programmen erfüllt werden müssen, die den Anspruch auf Erfüllung dieses Teils der IEC 1131 erheben.

1.5.1 SPS-Systeme

Ein SPS-System, wie es in IEC 1131-1 definiert ist, das den Anspruch erhebt, vollständig oder teilweise die Anforderungen dieses Teils der IEC 1131 zu erfüllen, muß dies genau so tun, wie unten beschrieben ist.

Eine Aussage zur Normerfüllung muß in der Dokumentation enthalten sein, die mit dem System geliefert wird, oder muß vom System selbst erzeugt werden. Die Form der Aussage zur Normerfüllung muß lauten:

"Dieses System erfüllt die Anforderungen der IEC 1131-3 in folgenden Eigenschaften der Sprache",
gefolgt von einer Reihe von Tabellen in folgendem Format:

Tabellenüberschrift

Tabellen-Nr	Eigenschaftsnr.	Beschreibung der Eigenschaft
...	...	...

Die Nummern der Tabellen und Eigenschaften und die Beschreibungen sind aus den Tabellen zu entnehmen, die in den relevanten Unterabschnitten dieses Teils der IEC 1131 angegeben sind. Die Tabellenüberschriften sind aus der folgenden Tabelle zu entnehmen.

Tabellenüberschrift	Für Eigenschaften in:
Gemeinsame Elemente	Abschnitt 2
Gemeinsame Textelemente	Unterabschnitt 3.1
AWL-Sprachelemente	Unterabschnitte 3.2.1 bis 3.2.3
ST-Sprachelemente	Unterabschnitte 3.3.1 bis 3.3.2.4
Gemeinsame Graphikelemente	Unterabschnitte 4.1 bis 4.1.4
KOP-Sprachelemente	Unterabschnitte 4.2 bis 4.2.6
FBS-Sprachelemente	Unterabschnitte 4.3 bis 4.3.3

Ein SPS-System, das die Anforderungen dieses Teils bezüglich einer Sprache erfüllt, die in diesem Teil definiert ist,

- a) darf nicht das Einschließen von Ersatz- oder Zusatz-Sprachelementen erforderlich machen, um eine der Eigenschaften zu erreichen, die in diesem Teil festgelegt sind;
- b) muß mit einem Schriftstück versehen sein, das die Werte aller herstellerabhängigen Parameter angibt, wie in Anhang D aufgelistet ist;
- c) muß imstande sein, zu unterscheiden, ob ein Sprachelement des Anwenders eine Anforderung dieses Teils verletzt; muß erkennen können, wo ein solcher Verstoß nicht als Fehler nach Anhang E bezeichnet wird, und das Ergebnis dieser Unterscheidung dem Anwender melden. Falls das System nicht die gesamte Programm-Organisationseinheit prüft, muß der Anwender darauf hingewiesen werden, daß die Prüfung unvollständig ist, auch wenn keine Verletzung in dem untersuchten Teil der Programm-Organisationseinheit entdeckt wurde;
- d) muß jeden Verstoß des Anwenders, der als Fehler in Anhang E bezeichnet ist, in mindestens einer der folgenden Weisen behandeln:
 - 1) es muß eine Aussage in einem mitgelieferten Schriftstück gemacht werden, daß der Fehler nicht gemeldet wird;
 - 2) das System muß während der Aufbereitung des Programms für die Ausführung melden, daß ein Auftreten dieses Fehlers möglich ist;

3) das System muß den Fehler während der Aufbereitung des Programms für die Ausführung melden;

4) das System muß den Fehler während der Ausführung des Programms melden und geeignete Verfahren zur Behandlung der Fehler in system- oder anwenderdefinierter Form einleiten;

und falls irgendwelche Verstöße, die als Fehler bezeichnet sind, in einer Weise behandelt werden, wie oben in Abschnitt d)1) beschrieben ist, dann muß eine Anmerkung in einem eigenen Abschnitt des mitgelieferten Schriftstücks erscheinen, die auf jede derartige Behandlung hinweist;

e) muß mit einem Schriftstück versehen sein, das separat alle Eigenschaften beschreibt, die von dem System angenommen werden und in diesem Teil verboten oder nicht festgelegt sind. Derartige Eigenschaften sind als "Erweiterungen zur < sprache > Sprache, die in IEC 1131-3 definiert ist", zu beschreiben;

f) muß imstande sein, in gleicher Weise jeden Gebrauch einer derartigen Erweiterung so zu verarbeiten, wie es für Fehler festgelegt ist;

g) muß imstande sein, in einer gleichen Weise jeden Gebrauch einer der implementierungsabhängigen Eigenschaften, die in Anhang D festgelegt sind, so zu verarbeiten, wie es für Fehler festgelegt ist;

h) darf nicht in diesem Teil definierte Namen für Standard-Datentypen, Funktionen oder Funktionsbausteine für herstellerdefinierte Eigenschaften benutzen, deren Funktionalität sich von dem unterscheidet, was in diesem Teil beschrieben ist;

i) muß mit einem Schriftstück versehen sein, das in der Form, wie es in Anhang A festgelegt ist, die formale Syntax aller Elemente der Textsprachen definiert, die durch das System unterstützt werden.

Die Formulierung "muß imstande sein" wird in diesem Unterabschnitt benutzt, um die Implementierung eines Software-Schalters zu erlauben, mit dem der Anwender die Meldung von Fehlern steuern kann.

In Fällen, in denen die Kompilation oder Programm-Eingabe wegen irgendwelcher Tabellen-Begrenzungen usw. abgebrochen wird, wird eine vollständige Angabe der Art "es wurden keine Verstöße entdeckt, die Prüfung ist jedoch unvollständig" die Anforderungen dieses Unterabschnitts erfüllen.

1.5.2 Programme

Ein SPS-Programm, das die Anforderungen der IEC 1131-3 erfüllt,

a) darf nur Eigenschaften benutzen, die in diesem Teil zum Gebrauch der jeweiligen Sprache festgelegt sind;

b) darf keine Eigenschaften benutzen, die als Erweiterungen der Sprache gelten;

c) darf von keiner bestimmten Interpretation implementierungsabhängiger Eigenschaften abhängen.

Die Ergebnisse, die von einem normgerechten Programm erzeugt werden, müssen dieselben sein, wie von irgendeinem normgerechten System erzeugt, das die Merkmale unterstützt, die von dem Programm benutzt werden; es sei denn, diese Ergebnisse sind vom Zeitverhalten der Programmausführung, der Anwendung von implementierungsabhängigen Eigenschaften (wie in Anhang D aufgeführt) im Programm und in der Ausführung von Prozeduren zur Fehlerbehandlung beeinflusst.

2 Gemeinsame Elemente

Dieser Abschnitt definiert Text- und Graphikelemente, die für alle SPS-Programmiersprachen dieses Teils der IEC 1131 gemeinsam sind.

2.1 Gebrauch der gedruckten Zeichen

2.1.1 Zeichensatz

Die Textsprachen und die Textelemente der Graphiksprachen müssen nach der "Basis-Code-Tabelle" des ISO/IEC-646-Zeichensatzes dargestellt werden.

Die Codierung von Zeichen des nationalen oder erweiterten (8-bit-)Zeichensatzes muß mit ISO/IEC 646 übereinstimmen. Der *geforderte Zeichensatz*, gezeigt als Eigenschaft 1 in der Tabelle 1, besteht aus allen Zeichen in den Spalten 3 bis 7 der "Basic-Code-Tabelle", die als Tabelle 1 in ISO/IEC 646 angegeben ist; ausgenommen sind die Kleinbuchstaben und die Zeichenpositionen, die reserviert oder zusätzlich für die Anwendung in nationalen Zeichensätzen verfügbar sind.

Der Hersteller muß eine Alternative (a oder b) für jede der Eigenschaften (3a, b) bis (6a, b) der Tabelle 1 gemäß den folgenden Regeln unterstützen:

— Das "Pfund-Zeichen" (£) muß an Stelle des "Nummern-Zeichens" (#) benutzt werden, falls letzteres die Zeichenposition 2/3 einer nationalen Belegung des ISO/IEC-646-Zeichensatzes belegt.

— Das "Währungszeichen" muß an Stelle des "Dollar-Zeichens" (\$) benutzt werden, falls letzteres die Zeichenposition 2/4 einer nationalen Belegung des ISO/IEC-646-Zeichensatzes belegt.

— Wenn die Zeichenposition 7/12 im ISO/IEC-646-Zeichensatz von einem anderen Zeichen in einem nationalen Zeichensatz belegt ist, muß das "Ausrufungszeichen" (!) an der Position 2/1 zur Darstellung der vertikalen Linien benutzt werden.

— Für die Begrenzung von Indizierungen müssen die linken und rechten runden Klammern "(" an Stelle der linken und rechten eckigen Klammern "[" benutzt werden, wenn letztere die Zeichenpositionen einer nationalen Implementierung des ISO/IEC-646-Zeichensatzes belegen.

ANMERKUNG: Der Gebrauch von Zeichen aus nationalen Zeichensätzen ist eine typische Erweiterung dieser Norm.

Tabelle 1: Zeichensatz-Eigenschaften

Nr	Beschreibung
1	Geforderter Zeichensatz (siehe 2.1.1)
2	Kleinbuchstaben
3a 3b	Nummern-Zeichen (#) oder Pfund-Zeichen (£)
4a 4b	Dollar-Zeichen (\$) oder Währungszeichen
5a 5b	Senkrechter Strich () oder Ausrufungszeichen (!)
6a 6b	Indizierungs-Begrenzer: Linke und rechte eckige Klammern "[]" oder Linke und rechte runde Klammern "()"
ANMERKUNG: Wenn Kleinbuchstaben (Eigenschaft 2) unterstützt werden, darf die Groß/Klein-schreibung in den Sprachelementen nicht signifikant sein (ausgenommen innerhalb von Ter-minalsymbolen, wie in den Anhängen A und B definiert, Kommentaren, wie in 2.1.5 definiert, Zei-chenfolgen, wie in 2.2.2 definiert, und Variablen vom Typ STRING, wie in 2.3.1 definiert); z. B. müs-sen die Bezeichner "abcd", "ABCD" und "aBCd" gleich interpretiert werden.	

2.1.2 Bezeichner

Ein *Bezeichner* ist eine Folge von Buchstaben, Ziffern und Unterstrich-Zeichen, die mit einem Buchstaben oder Unterstrich-Zeichen beginnen muß.

Unterstriche müssen in Bezeichnern signifikant sein; z. B. "A_BCD" und "AB_CD" müssen als unterschiedliche Bezeichner interpretiert werden. Mehrere führende oder mehrfach eingebettete Unterstriche sind nicht zulässig.

Bezeichner dürfen keine Leerzeichen (SP) enthalten.

Mindestens sechs eindeutige Zeichen müssen in allen Systemen unterstützt werden, die den Gebrauch von Bezeichnern bieten; z. B. muß "ABCDE1" als unterschiedlich zu "ABCDE2" in allen diesen Systemen interpretiert werden.

Eigenschaften und Beispiele werden in Tabelle 2 gezeigt.

Tabelle 2: Bezeichner-Eigenschaften

Nr	Beschreibung der Eigenschaft	Beispiele
1	Großbuchstaben und Zahlen	IW215 IW215Z QX75 IDENT
2	Groß- und Kleinbuchstaben, Zahlen, eingebettete Unterstriche	Alle oben aufgeführten plus: LIM_SW_5 LimSw5 abcd ab_Cd
3	Groß- und Kleinbuchstaben, Zahlen, führende und eingebettete Unterstriche	Alle oben aufgeführten plus: _MAIN _12V7

2.1.3 Schlüsselwörter

Schlüsselwörter sind eindeutige Kombinationen von Zeichen, die als individuelle syntaktische Elemente angewendet werden, wie dies in Anhang B definiert ist. Alle Schlüsselwörter, die in diesem Teil benutzt werden, sind in Anhang C aufgelistet. Schlüsselwörter dürfen keine eingebetteten Leerzeichen enthalten. Die Schlüsselwörter, die im Anhang C aufgelistet sind, dürfen für keinen anderen Zweck genutzt werden, z. B. als Variablennamen oder Erweiterungen, wie in 1.5.1 definiert.

ANMERKUNG: Nationale Normungsorganisationen dürfen Übersetzungstabellen der Schlüsselwörter aus Anhang C herausgeben.

2.1.4 Gebrauch von Leerzeichen

Dem Anwender muß erlaubt sein, ein oder mehrere Leerzeichen (Codeposition 2/0 im ISO/IEC-646-Zeichensatz) überall im Text des SPS-Programms einzufügen, ausgenommen innerhalb von Schlüsselwörtern, Literalen, Bezeichnern oder Kombinationen von Begrenzern (z. B. für Kommentare, wie dies unten definiert ist).

2.1.5 Kommentare

Anwender-Kommentare müssen am Anfang und Ende mit einer besonderen Zeichenkombination "(" bzw. ")" begrenzt werden, wie in Tabelle 3 gezeigt ist. Außer in der Sprache AWL, die in Abschnitt 3.2 definiert ist, müssen Kommentare überall im Programm zulässig sein, wo Leerzeichen erlaubt sind; ausgenommen innerhalb Zeichenfolge-Literalen, die in 2.2.2 definiert sind. Kommentare dürfen keine syntaktische oder semantische Bedeutung für eine der in diesem Teil definierten Sprachen haben.

Geschachtelte Kommentare sind nicht zulässig; z. B. (*(*GESCHACHTELT*)*)

Tabelle 3: Kommentar-Eigenschaft

Nr	Beschreibung	Beispiel
1	Kommentare	<pre> (***** (* Ein eingerahmter Kommentar *) (***** </pre>

2.2 Externe Darstellung von Daten

Externe Darstellungen von Daten bei den verschiedenen SPS-Programmiersprachen müssen numerische Literale, Zeichenreihen und Zeitlitterale umfassen.

2.2.1 Numerische Literale

Es gibt zwei Arten von numerischen Literalen: die ganzzahligen (integer) Literale und die reellen (real) Literale. Ein numerisches Literal ist als eine Dezimalzahl oder eine basisbezogene Zahl definiert. Die maximale Anzahl von Ziffern für alle Arten von numerischen Literalen muß ausreichend sein, um den gesamten Bereich und die Genauigkeit der Werte aller Datentypen auszudrücken, die durch das Literal in einer gegebenen Implementierung dargestellt werden.

Einzelne Unterstrich-Zeichen (_), die zwischen den Ziffern eines numerischen Literals eingefügt sind, dürfen nicht signifikant sein. Eine andere Anwendung von Unterstrich-Zeichen in numerischen Literalen ist nicht zulässig.

Dezimale Literale müssen in herkömmlicher Dezimal-Schreibweise dargestellt werden. Reelle Literale müssen durch die Angabe eines Dezimalpunktes gekennzeichnet werden. Ein Exponent gibt die ganzzahlige Zehnerpotenz an, mit der die vorangehende Zahl zu multiplizieren ist, um den dargestellten Wert zu erhalten. Dezimale Literale und ihre Exponenten können ein vorangestelltes Vorzeichen (+ oder -) haben.

Ganzzahlige Literale können auch zur Basis 2, 8 oder 16 dargestellt werden. Diese Basis muß in dezimaler Angabe sein. Für die Basis 16 muß ein erweiterter Satz von Zeichen benutzt werden, der aus den Buchstaben A bis F besteht, die die eigentliche Bedeutung der Dezimalen 10 bis 15 haben. Basisbezogene Zahlen dürfen kein Vorzeichen (+ oder -) beinhalten.

Boolesche Daten müssen durch ganzzahlige Literale mit dem Wert Null (0) oder Eins (1) dargestellt werden oder durch die Schlüsselwörter FALSE bzw. TRUE.

Eigenschaften und Beispiele sind in Tabelle 4 gezeigt.

Tabelle 4: Numerische Literale

Nr	Beschreibung der Eigenschaft	Beispiele
1	Ganzzahlige Literale	-12 0 123_456 +986
2	Reelle Literale	-12.0 0.0 0.456 3.14159_26
3	Reelle Literale mit Exponenten	-1.34E-12 oder -1.34e-12 1.0E+6 oder 1.0e+6 1.234E6 oder 1.234e6
4	Basis-2-Literale	2 #1111_1111 (255 dezimal) 2 #1110_0000 (240 dezimal)
5	Basis-8-Literale	8 #377 (255 dezimal) 8 #340 (240 dezimal)
6	Basis-16-Literale	16 #FF oder 16 #ff (255 dezimal) 16 #E0 oder 16 #e0 (240 dezimal)
7	Boolesche Null und Eins	0 1
8	Boolesche FALSE und TRUE	FALSE TRUE
ANMERKUNG: Die Schlüsselwörter FALSE and TRUE korrespondieren mit den booleschen Werten 0 und 1.		

2.2.2 Zeichenfolge-Literale

Ein Zeichenfolge-Literal ist eine Sequenz von null oder mehr Zeichen, die durch einfache Anführungszeichen (') eingeschlossen ist. In Zeichenfolgen muß die Kombination der drei Zeichen Dollar-Zeichen (\$), gefolgt von zwei hexadezimalen Ziffern, als die hexadezimale Darstellung des 8-Bit-Zeichencodes interpretiert werden. Dies ist in Tabelle 5 gezeigt. Zusätzlich müssen die 2-Zeichen-Kombinationen, die mit dem Dollar-Zeichen beginnen und wenn sie in Zeichenfolgen auftreten, gemäß Tabelle 6 interpretiert werden.

Tabelle 5: Eigenschaft der Zeichenfolge-Literale

Nr	Beispiel	Erläuterung
1	''	Leere Zeichenfolge (Länge null)
	'A'	Zeichenfolge der Länge eins, mit dem einzelnen Zeichen A
	''	Zeichenfolge der Länge eins, mit "Leerzeichen"
	'\$'	Zeichenfolge der Länge eins, mit individuellem "Anführungszeichen"
	'\$R\$L' '\$OD\$OA'	Zeichenfolgen der Länge zwei, mit CR- und LF-Zeichen
	'\$\$1.00'	Zeichenfolge der Länge fünf, die als "\$1.00" gedruckt wird

Tabelle 6: 2-Zeichen-Kombinationen in Zeichenfolgen

Nr	Kombination	Bedeutung beim Ausdruck
2	\$\$	Dollar-Zeichen
3	'\$'	Einfaches Anführungszeichen
4	\$L oder \$l	Zeilenvorschub (Line feed)
5	\$N oder \$n	Neue Zeile (New line)
6	\$P oder \$p	Neue Seite (page)
7	\$R oder \$r	Wagenrücklauf (Carriage return)
8	\$T oder \$t	Tabulator

ANMERKUNG: Das Zeichen "neue Zeile" bietet ein implementierungsabhängiges Mittel, um das Ende einer Datenzeile zu definieren. Dies gilt sowohl bei physikalischen als auch bei Datei-Ein/Ausgaben. Beim Drucken bewirkt es das Beenden einer Datenzeile und das Fortsetzen in der nächsten Zeile.

2.2.3 Zeitlitterale

Die Notwendigkeit, externe Darstellungen für zwei unterschiedliche Typen von Zeitdaten zur Verfügung zu stellen, ist gegeben: Daten für die *Zeitdauer* zur Messung und Steuerung der verstrichenen Zeit eines Steuerungsereignisses und Daten für die *Tageszeit* (die auch Datumsinformationen beinhalten können) zur Synchronisation von Beginn oder Ende eines Steuerungsereignisses mit einer absoluten Zeitangabe.

Die Literale für Zeitdauer und Tageszeit müssen links durch die Schlüsselwörter, die in 2.2.3.1 und 2.2.3.2 definiert sind, begrenzt werden.

2.2.3.1 Zeitdauer

Die Daten für Zeitdauer müssen links durch das Schlüsselwort **T #**, **TIME #**, **t #** oder **time #** begrenzt werden. Die Darstellung von Daten zur Zeitdauer, wie Tage, Stunden, Minuten, Sekunden und Millisekunden oder alle Kombinationen hiervon, muß in der Weise erfolgen, wie es in Tabelle 7 gezeigt ist. Die niedrigstwertige Zeiteinheit kann als reelle Zahl ohne Exponent geschrieben werden.

Die Einheiten von Zeitdauer-Literalen können durch Unterstrich-Zeichen getrennt werden.

Der "Überlauf" der höchstwertigen Einheit eines Zeitdauer-Literals ist zulässig; z. B. ist die Angabe **T #25h_15m** erlaubt. Zeiteinheiten, wie z. B. Sekunden, Millisekunden usw., können in Groß- oder Kleinbuchstaben dargestellt werden.

Tabelle 7: Zeitdauer-Literal-Eigenschaften

Nr	Beschreibung	Beispiele
1a	Zeitdauer-Literale ohne Unterstriche:	T #14ms T #14.7s T #14.7m T #14.7h t #14.7d t #25h15m t #5d14h12m18s3.5ms
	kurzes Präfix	
1b	langes Präfix	TIME #14ms time #14.7s
2a	Zeitdauer-Literale mit Unterstrichen:	
2b	kurzes Präfix	t #25h_15m t #5d_14h_12m_18s_3.5ms
	langes Präfix	TIME #25h_15m time #5d_14h_12m_18s_3.5ms

2.2.3.2 Tageszeit und Datum

Die vorangestellten Schlüsselwörter für die Literale Tageszeit und Datum müssen Tabelle 8 entsprechen. Wie in Tabelle 9 veranschaulicht ist, muß die Darstellung der Information von Tageszeit und Datum der Festlegung von ISO 8601 entsprechen.

Tabelle 8: Literale für Datum und Tageszeit

Nr	Beschreibung der Eigenschaft	Schlüsselwort
1	Literale für Datum (langes Präfix)	DATE #
2	Literale für Datum (kurzes Präfix)	D #
3	Literale für Tageszeit (langes Präfix)	TIME_OF_DAY #
4	Literale für Tageszeit (kurzes Präfix)	TOD #
5	Literale für Datum und Zeit (langes Präfix)	DATE_AND_TIME #
6	Literale für Datum und Zeit (kurzes Präfix)	DT #

Tabelle 9: Beispiele für Datum und Tageszeit

In langer Präfix-Schreibweise	In kurzer Präfix-Schreibweise
DATE #1984-06-25 date #1984-06-25	D #1984-06-25 d #1984-06-25
TIME_OF_DAY #15:36:55.36 time_of_day #15:36:55.36	TOD #15:36:55.36 tod #15:36:55.36
DATE_AND_TIME #1984-06-25-15:36:55.36 date_and_time #1984-06-25-15:36:55.36	DT #1984-06-25-15:36:55.36 dt #1984-06-25-15:36:55.36

2.3 Datentypen

Eine Anzahl von elementaren (vordefinierten) Datentypen wird durch diese Norm anerkannt. Zusätzlich werden allgemeine (generische) Datentypen zur Anwendung bei der Definition von überladenen (typunabhängigen) Funktionen (siehe 2.5.1.4) definiert. Ein Verfahren für den Anwender oder den Hersteller zur Festlegung zusätzlicher Datentypen ist ebenfalls definiert.

2.3.1 Elementare Datentypen

Die elementaren Datentypen, das Schlüsselwort für jeden Datentyp, die Anzahl der Bits pro Datenelement und der Wertebereich für jeden elementaren Datentyp müssen der Tabelle 10 entsprechen.

Tabelle 10: Elementare Datentypen

Nr	Schlüsselwort	Datentyp	Bits	Bereich
1	BOOL	boolesche	1	Anmerkung 8
2	SINT	kurze ganze Zahl (short integer)	8	Anmerkung 2
3	INT	ganze Zahl (integer)	16	Anmerkung 2
4	DINT	doppelte ganze Zahl (double integer)	32	Anmerkung 2
5	LINT	lange ganze Zahl (long integer)	64	Anmerkung 2
6	USINT	vorzeichenlose kurze ganze Zahl (unsigned short integer)	8	Anmerkung 3
7	UINT	vorzeichenlose ganze Zahl	16	Anmerkung 3
8	UDINT	vorzeichenlose doppelte ganze Zahl	32	Anmerkung 3
9	ULINT	vorzeichenlose lange ganze Zahl	64	Anmerkung 3
10	REAL	reelle Zahl	32	Anmerkung 4
11	LREAL	lange reelle Zahl	64	Anmerkung 5

(fortgesetzt)

Tabelle 10 (abgeschlossen)

Nr	Schlüsselwort	Datentyp	Bits	Bereich
12	TIME	Zeitdauer	Anmerkung 1	Anmerkung 6
13	DATE	Datum (nur)	Anmerkung 1	Anmerkung 6
14	TIME_OF_DAY oder TOD	Uhrzeit (nur)	Anmerkung 1	Anmerkung 6
15	DATE_AND_TIME oder DT	Datum und Uhrzeit	Anmerkung 1	Anmerkung 6
16	STRING	variabel-lange Zeichenfolge	Anmerkung 1	Anmerkung 7
17	BYTE	Bit-Folge der Länge 8	8	Anmerkung 7
18	WORD	Bit-Folge der Länge 16	16	Anmerkung 7
19	DWORD	Bit-Folge der Länge 32	32	Anmerkung 7
20	LWORD	Bit-Folge der Länge 64	64	Anmerkung 7

ANMERKUNG 1: Die Länge dieser Datenelemente ist implementierungsabhängig.
ANMERKUNG 2: Der Wertebereich für Variablen dieses Datentyps reicht von $-(2^{**}(\text{Bits}-1))$ bis $(2^{**}(\text{Bits}-1)) - 1$.
ANMERKUNG 3: Der Wertebereich für Variablen dieses Datentyps reicht von 0 bis $(2^{**}\text{Bits}) - 1$.
ANMERKUNG 4: Der Wertebereich für Variablen dieses Datentyps muß so sein, wie es in IEC 559 für das Gleitpunkt-Format mit einfacher Länge definiert ist.
ANMERKUNG 5: Der Wertebereich für Variablen dieses Datentyps muß so sein, wie es in IEC 559 für das Gleitpunkt-Format mit doppelter Länge definiert ist.
ANMERKUNG 6: Der Wertebereich für Variablen dieses Datentyps ist implementierungsabhängig.
ANMERKUNG 7: Ein numerischer Wertebereich ist für diesen Datentyp nicht anwendbar.
ANMERKUNG 8: Die möglichen Werte für Variablen dieses Datentyps müssen 0 und 1 sein, entsprechend den Schlüsselwörtern FALSE und TRUE.

2.3.2 Allgemeine (generische) Datentypen

Zusätzlich zu den Datentypen in Tabelle 10 muß die Hierarchie der allgemeinen Datentypen aus Tabelle 11 so angewendet werden, wie es in 2.5.1.4 bei der Festlegung von überladenen Eingängen und Ausgängen von Standardfunktionen und Funktionsbausteinen definiert ist. Die allgemeinen Datentypen werden durch die Vorsilbe "ANY" identifiziert.

Tabelle 11: Hierarchie der allgemeinen Datentypen

ANY ANY_NUM ANY_REAL LREAL REAL ANY_INT LINT, DINT, INT, SINT ULINT, UDINT, UINT, USINT ANY_BIT LWORD, DWORD, WORD, BYTE, BOOL STRING ANY_DATE DATE_AND_TIME DATE TIME_OF_DAY TIME Abgeleitet (siehe Anmerkungen)
ANMERKUNG 1: Allgemeine Datentypen dürfen nicht in anwenderdeklarierten Programm-Organisationseinheiten, die in 2.5 definiert sind, benutzt werden. ANMERKUNG 2: Der allgemeine Typ eines abgeleiteten <i>Unterbereichstyps</i> (Eigenschaft 3 der Tabelle 12) muß ANY_INT sein. ANMERKUNG 3: Der allgemeine Typ eines <i>direkt abgeleiteten</i> Typs (Eigenschaft 1 der Tabelle 12) muß derselbe sein wie der allgemeine Typ des elementaren Typs, von dem er abgeleitet ist. ANMERKUNG 4: Der allgemeine Typ von allen anderen abgeleiteten Typen, die in Tabelle 12 definiert sind, muß ANY sein.

2.3.3 Abgeleitete Datentypen

2.3.3.1 Deklarationen (Vereinbarungen)

Abgeleitete (d. h. anwender- oder herstellerdefinierte) Datentypen können durch die Anwendung der textuellen Konstruktion TYPE...END_TYPE, die in Tabelle 12 gezeigt ist, deklariert werden. Diese abgeleiteten Datentypen können dann zusätzlich zu den elementaren Datentypen, die in 2.3.1 definiert sind, in Deklarationen von Variablen benutzt werden, wie in 2.4.3 definiert ist. Eine Deklaration für den Datentyp für *Aufzählung* legt fest, daß der Wert jedes Datenelements dieses Typs nur einen der Werte annehmen kann, die in der zugehörigen Liste der Bezeichner angegeben sind, wie in Tabelle 12 veranschaulicht ist.

Eine Deklaration für den *Unterbereich* legt fest, daß der Wert für jedes Datenelement dieses Typs nur Werte annehmen kann, die innerhalb und einschließlich der festgelegten Ober- und Untergrenze liegen, wie dies in Tabelle 12 angegeben ist.

Eine Deklaration für *Struktur* (STRUCT) legt fest, daß Datenelemente dieses Typs Unterelemente enthalten müssen, die festgelegte Typen haben und auf die mit festgelegten Namen zugegriffen werden kann. Zum Beispiel enthält ein Element des Typs ANALOG_CHANNEL_CONFIGURATION, wie in Tabelle 12 deklariert, ein Unterelement RANGE vom Typ ANALOG_SIGNAL_RANGE, ein Unterelement MIN_SCALE vom Typ ANALOG_DATA und ein Element MAX_SCALE vom Typ ANALOG_DATA.

Eine Deklaration für *Feld* (ARRAY) legt fest, daß eine ausreichende Menge von Datenspeichern für jedes Element dieses Typs zugewiesen werden muß, um alle die Daten zu speichern, auf die mit dem festgelegten Indexbereich gezeigt werden kann. So muß jedes Element vom Typ ANALOG_16_INPUT_CONFIGURATION, wie in Tabelle 12 gezeigt, ausreichend Speicherplatz für 16 Elemente CHANNEL vom Typ ANALOG_CHANNEL_CONFIGURATION (neben weiteren Elementen) enthalten. Mechanismen für den Zugriff auf Feldelemente sind in 2.4.1.2 definiert.

2.3.3.2 Initialisierung

Der voreingestellte Wert eines Datentyps für *Aufzählung* muß der erste Bezeichner in der zugehörigen Aufzählungsliste sein oder ein Wert, der durch den Zuweisungsoperator ":= " festgelegt ist. Zum Beispiel haben, wie in den Tabellen 12 und 14 gezeigt, die voreingestellten Anfangswerte der Elemente der Datentypen ANALOG_SIGNAL_TYPE und ANALOG_SIGNAL_RANGE die Werte SINGLE_ENDED bzw. UNIPOLAR_1_5V.

Für Datentypen mit *Unterbereichen* muß der voreingestellte Anfangswert die erste (untere) Grenze des Unterbereichs sein, falls dies nicht anders durch einen Zuweisungsoperator festgelegt ist. Wie in Tabelle 12 deklariert, ist z. B. der voreingestellte Anfangswert der Elemente vom Typ ANALOG_DATA gleich 4 095, während der voreingestellte Anfangswert für das Unterelement FILTER_PARAMETER der Elemente vom Typ ANALOG_16_INPUT_CONFIGURATION gleich Null ist. Im Gegensatz dazu ist der voreingestellte Anfangswert der Elemente vom Typ ANALOG_DATAZ, wie in Tabelle 14 deklariert, gleich Null.

Für weitere abgeleitete Datentypen müssen die voreingestellten Anfangswerte gleich denen der unterlagerten Datentypen sein, wie es in Tabelle 13 definiert ist; falls sie nicht anders durch die Anwendung des Zuweisungsoperators ":= " in der Typdeklaration festgelegt sind. Weitere Beispiele für die Anwendung des Zuweisungsoperators bei der Initialisierung sind in 2.4.2 angegeben.

Die voreingestellte maximale Länge von Elementen des Typs STRING muß ein implementierungsabhängiger Wert sein, falls sie nicht anders durch eine in Klammern angegebene Maximallänge in der zugehörigen Deklaration festgelegt ist. (Diese Länge darf die implementierungsabhängige Voreinstellung nicht überschreiten.) Wenn z. B. der Typ STR10 wie folgt deklariert ist:

TYPE STR10 : STRING (10) := 'ABCDEF' ; END_TYPE

dann ist die Maximallänge, der voreingestellte Anfangswert und die voreingestellte Anfangslänge von Datenelementen des Typs STR10 gleich 10 Zeichen, 'ABCDEF' bzw. 6 Zeichen.

Tabelle 12: Eigenschaften der Datentyp-Deklarationen

Nr	Eigenschaft/Beispiel in Textform
1	Direkte Ableitung von elementaren Typen, z. B.: TYPE R : REAL ; END_TYPE
2	Datentypen für Aufzählung, z. B.: TYPE ANALOG_SIGNAL_TYPE : (SINGLE_ENDED, DIFFERENTIAL) ; END_TYPE
3	Datentypen für Bereich, z. B.: TYPE ANALOG_DATA : INT (-4095..4095) ; END_TYPE
4	Datentypen für Feld, z. B.: TYPE ANALOG_16_INPUT_DATA : ARRAY [1..16] OF ANALOG_DATA ; END_TYPE
5	Datentypen für Strukturen, z. B.: TYPE ANALOG_CHANNEL_CONFIGURATION : STRUCT RANGE : ANALOG_SIGNAL_RANGE ; MIN_SCALE : ANALOG_DATA ; MAX_SCALE : ANALOG_DATA ; END_STRUCT ; ANALOG_16_INPUT_CONFIGURATION : STRUCT SIGNAL_TYPE : ANALOG_SIGNAL_TYPE ; FILTER_PARAMETER : SINT (0..99) ; CHANNEL : ARRAY [1..16] OF ANALOG_CHANNEL_CONFIGURATION ; END_STRUCT ; END_TYPE
ANMERKUNG: Beispiele für den Gebrauch dieser Typen in Deklarationen von Variablen siehe Unterabschnitte 2.3.3.3, 2.4.1.2 und Tabelle 17.	

Tabelle 13: Voreingestellte Anfangswerte

Datentypen	Initialisierungswert
BOOL, SINT, INT, DINT, LINT	0
USINT, UINT, UDINT, ULINT	0
BYTE, WORD, DWORD, LWORD	0
REAL, LREAL	0.0
TIME	T #0S
DATE	D #0001-01-01
TIME_OF_DAY	TOD #00:00:00
DATE_AND_TIME	DT #0001-01-01-00:00:00
STRING	“(die leere Zeichenfolge)

Tabelle 14: Eigenschaften der Deklaration der Anfangswerte von Datentypen

Nr	Eigenschaft/Beispiel in Textform
1	Initialisierung von direkt abgeleiteten Typen, z. B.: TYPE FREQ : REAL := 50 ; END_TYPE
2	Initialisierung von Datentypen für Aufzählung, z. B. TYPE ANALOG_SIGNAL_RANGE : (BIPOLAR_10V, (* - 10 to + 10 VDC *) UNIPOLAR_10V, (* 0 to + 10 VDC *) UNIPOLAR_1_5V, (* + 1 to + 5 VDC *) UNIPOLAR_0_5V, (* 0 to + 5 VDC *) UNIPOLAR_4_20_MA, (* + 4 to + 20 mADC *) UNIPOLAR_0_20_MA (* 0 to + 20 mADC *)) := UNIPOLAR_1_5V ; END_TYPE
3	Initialisierung von Datentypen für Unterbereich, z. B.: TYPE ANALOG_DATAZ : INT (-4095..4095) := 0 ; END_TYPE
4	Initialisierung von Datentypen für Feld, z. B.: TYPE ANALOG_16_INPUT_DATAI : ARRAY [1..16] OF ANALOG_DATA := 8(-4095), 8(4095); END_TYPE
5	Initialisierung von Datentypen für Struktur, z. B.: TYPE ANALOG_CHANNEL_CONFIGURATIONI : STRUCT RANGE : ANALOG_SIGNAL_RANGE ; MIN_SCALE : ANALOG_DATA := -4095 ; MAX_SCALE : ANALOG_DATA := 4095 ; END_STRUCT ; END_TYPE
6	Initialisierung von Datentypen für abgeleitete Struktur, z. B.: TYPE ANALOG_CHANNEL_CONFIGZ : ANALOG_CHANNEL_CONFIGURATION (MIN_SCALE := 0, MAX_SCALE := 9999); END_TYPE

2.3.3.3 Gebrauch

Der Gebrauch von Variablen, die als abgeleitete Datentypen deklariert sind (wie in 2.4.3.1 definiert ist), muß folgenden Regeln folgen:

- 1) Eine Einzelelement-Variable, wie in 2.4.1.1 definiert, mit einem abgeleiteten Typ kann überall benutzt werden, wo eine Variable vom Typ ihrer "Eltern" benutzt werden kann; z. B. können Variable mit den Typen R und PI, wie in den Tabellen 12 und 14 gezeigt, überall benutzt werden, wo eine Variable vom Typ REAL benutzt werden kann; und Variable mit dem Typ ANALOG_DATA können überall benutzt werden, wo eine Variable vom Typ INT benutzt werden kann.

Diese Regel kann rekursiv angewendet werden. Sind z.B. die folgenden Deklarationen gegeben, dann kann die Variable R3 vom Typ R2 überall angewendet werden, wo eine Variable vom Typ REAL angewendet werden kann:

```
TYPE R1 : REAL := 1.0 ; END_TYPE
TYPE R2 : R1 ; END_TYPE
VAR R3 : R2 ; END_VAR
```

2) Ein Element einer Mehrelement-Variablen, wie in 2.4.1.2 definiert, kann überall benutzt werden, wo der "Elterntyp" benutzt werden kann. Sind z. B. die Deklarationen von ANALOG_16_INPUT_DATA aus der Tabelle 12 und die Deklaration

```
VAR INS : ANALOG_16_INPUT_DATA ; END_VAR
```

gegeben, dann können die Variablen INS [1] bis INS [16] überall benutzt werden, wo eine Variable vom Typ INT benutzt werden könnte.

Diese Regel kann auch rekursiv angewendet werden; sind z. B. die Deklarationen ANALOG_16_INPUT_CONFIGURATION, ANALOG_CHANNEL_CONFIGURATION und ANALOG_DATA aus Tabelle 12 und die Deklaration

```
VAR CONF : ANALOG_16_INPUT_CONFIGURATION ; END_VAR
```

gegeben, dann kann die Variable CONF.CHANNEL [2].MIN_SCALE überall benutzt werden, wo eine Variable vom Typ INT benutzt werden könnte.

2.4 Variablen

Im Gegensatz zu den externen Darstellungen von Daten, wie in 2.2 beschrieben, liefern *Variable* die Mittel zur Identifizierung von Datenobjekten, deren Inhalt sich ändern darf; d. h. Daten, die mit Eingängen, Ausgängen oder Speicherplatz der SPS verbunden sind. Eine Variable kann als eine der elementaren Typen, die in 2.3.1 definiert sind, deklariert werden, oder als eine der abgeleiteten Typen, wie in 2.3.3.1 definiert.

2.4.1 Darstellung

2.4.1.1 Einzelelement-Variablen

Eine *Einzelelement-Variable* ist als eine Variable definiert, die ein einzelnes Datenelement von einem der elementaren Typen, wie in 2.3.1 definiert, oder einen abgeleiteten Typ für Aufzählung oder Unterbereich, wie in 2.3.3.1 definiert, oder einen abgeleiteten Typ, dessen "Elternschaft", wie in 2.3.3.3 rekursiv definiert, zu einem elementaren, Aufzählungs- oder Unterbereichstyp zurückverfolgt werden kann, darstellt. Dieser Unterabschnitt definiert die Mittel zur Darstellung derartiger Variablen auf *symbolische* Art oder alternativ in einer *direkten* Weise, die die Verbindung des Datenelements zu den physikalischen oder logischen Speicherorten in der SPS, nämlich Eingang, Ausgang und Speicherstruktur, darstellt.

Die Bezeichner, wie sie in 2.1.2 definiert sind, müssen zur symbolischen Darstellung von Variablen angewendet werden.

Die direkte Darstellung einer Einzelelement-Variablen muß durch ein besonderes Symbol ausgedrückt werden. Dieses Symbol wird durch folgende Aneinanderreihung gebildet: ein Prozentzeichen "%" (Position 2/5 in der ISO/IEC-646-Codetabelle), ein *Präfix für den Speicherort* und ein *Präfix für die Größe* nach Tabelle 15 und eine oder mehrere vorzeichenlose ganze Zahlen, die durch Punkte (.) getrennt sind.

Beispiele für direkt dargestellte Variablen sind:

%QX75 und %Q75	Ausgangsbit 75
%IW215	Eingangswort-Speicherort 215
%QB7	Ausgangsbyte-Speicherort 7
%MD48	Merker-Doppelwort am Speicherort 48
%IW2.5.7.1	siehe Erläuterung unten

Der Hersteller muß den Zusammenhang zwischen der direkten Darstellung einer Variablen und dem physikalischen oder logischen Speicherort des angesprochenen Punkts im Speicher, Eingang oder Ausgang festlegen. Bei einer erweiterten direkten Darstellung mit zusätzlichen ganzzahligen Feldern, die durch Punkte getrennt sind, muß dies als *hierarchische* physi-

Tabelle 15: Eigenschaften der Präfixe für Speicherort und Größe bei direkt dargestellten Variablen

Nr	Präfix	Bedeutung
1	I	Speicherort Eingang
2	Q	Speicherort Ausgang
3	M	Speicherort Merker
4	X	(Einzel-)Bit-Größe
5	kein	(Einzel-)Bit-Größe
6	B	Byte-(8 bit) Größe
7	W	Wort-(16 bit) Größe
8	D	Doppelwort-(32 bit) Größe
9	L	Langwort-(64 bit) Größe
ANMERKUNG 1: Falls nicht anders deklariert, muß der Datentyp einer direkt adressierten Variablen einer "(Einzel-)Bit"-Größe vom Typ BOOL sein. ANMERKUNG 2: Nationale Normungsorganisationen können Tabellen mit Übersetzungen dieser Präfixe herausgeben.		

kalische oder logische Adresse interpretiert werden; dabei stellt das linke Feld die höchste Ebene der Hierarchie dar, die weiter rechts erscheinenden sind aufeinanderfolgende niedrigere Ebenen. Zum Beispiel darf die Variable %IW2.5.7.1 den ersten "Kanal" (Wort) des siebten "Moduls" im fünften "Baugruppenträger" des zweiten "E/A-Busses" eines SPS-Systems darstellen. Der Gebrauch der hierarchischen Adressierung, um einem Programm in einem SPS-System den Datenzugriff in einer weiteren SPS zu erlauben, muß als eine Spracherweiterung betrachtet werden.

Der Gebrauch von direkt dargestellten Variablen ist nur in *Programmen*, die in 2.5.3 definiert sind, und in *Konfigurationen* und *Ressourcen*, die in 2.7.1 definiert sind, zugelassen. Die maximale Anzahl der Ebenen einer hierarchischen Adressierung ist ein implementierungsabhängiger Parameter.

2.4.1.2 Multielement-Variable

Die Typen der Multielement-Variablen, die in dieser Norm definiert sind, sind die *Felder* (ARRAYS) und *Strukturen* (STRUCTURES).

Ein *Feld* ist eine Sammlung von Datenelementen des gleichen Datentyps, die durch einen oder mehrere *Feldindizes* angesprochen werden; diese Indizes sind in eckigen Klammern eingeschlossen und durch Kommas getrennt. Ein Index muß ein Ausdruck sein, der einen Wert entsprechend einem der Untertypen des allgemeinen Typs ANY_INT, wie in Tabelle 11 definiert, liefert.

Ein Beispiel für die Anwendung einer Feldvariablen in ST-Sprache, definiert in Unterabschnitt 3.3, ist:

```
OUTARY[%MB6,SYM] := INARY[0] + INARY[7] - INARY[%MB6] * %IW62 ;
```

Die Maximalanzahl von Indizes und der maximale Bereich der Index-Werte, die benutzt werden dürfen, um auf Feldvariable zuzugreifen, ist ein implementierungsabhängiger Parameter.

Eine *strukturierte Variable* ist eine Variable, die als ein Typ deklariert ist, der vorher bereits als eine Datenstruktur festgelegt wurde, d. h. als ein Datentyp, der aus einer Sammlung von bezeichneten Elementen besteht.

Ein Element einer strukturierten Variablen muß durch zwei oder mehr Bezeichner oder durch Feldzugriffe dargestellt werden, die durch Punkte (.) getrennt sind. Der erste Bezeichner stellt den Namen des strukturierten Elements dar. Die folgenden Bezeichner stellen die Folge von Komponentennamen zum Zugriff auf das bestimmte Datenelement innerhalb der Datenstruktur dar.

Wenn z. B. die Variable MODULE_5_CONFIG als Typ ANALOG_16_INPUT_CONFIGURATION, wie in Tabelle 12 gezeigt, deklariert worden ist, dann würden die folgenden Anweisungen in ST-Sprache, die in Unterabschnitt 3.3 definiert ist, bewirken, daß der Wert SINGLE_ENDED dem Element SIGNAL_TYPE aus der Variable MODULE_5_CONFIG zugewiesen wird. Außerdem würde der Wert BIPOLAR_10V dem Unterelement RANGE des fünften Elements CHANNEL von MODULE_5_CONFIG zugewiesen:

```
MODULE_5_CONFIG.SIGNAL_TYP := SINGLE_ENDED;
```

```
MODULE_5_CONFIG.CHANNEL[5].RANGE := BIPOLAR_10V;
```

Die Maximalanzahl der Ebenen der Adressierung von Strukturelementen ist ein implementierungsabhängiger Parameter.

2.4.2 Initialisierung

Wenn ein Konfigurationselement (*Ressource* oder *Konfiguration*) "gestartet" wird, wie in 1.4.1 definiert, können die Variablen, die mit dem Konfigurationselement und seinem *Programm* verbunden sind, einen der folgenden Anfangswerte annehmen:

- den Wert, den die Variable hatte, als das Konfigurationselement "gestoppt" wurde (ein *gepufferter* Wert);
- einen anwenderspezifisierten Anfangswert;
- den voreingestellten Anfangswert für den der Variablen zugeordneten Datentyp.

Der Anwender kann deklarieren, daß die Variable *gepuffert* sein soll, indem er das Bestimmungszeichen RETAIN aus Tabelle 16 benutzt, sofern diese Eigenschaft von der Implementierung unterstützt wird.

Der Anfangswert einer Variablen beim Starten seines zugehörigen Konfigurationselements muß nach folgenden Regeln ermittelt werden:

- 1) Falls die Startoperation ein "Warmstart" ist, der in IEC 1131-1 definiert ist, müssen die Anfangswerte von *gepufferten* Variablen ihre gepufferten Werte, wie oben definiert, annehmen.
- 2) Falls die Operation ein "Kaltstart" ist, wie in IEC 1131-1 definiert, müssen die Anfangswerte der gepufferten Variablen die anwenderspezifisierten Anfangswerte annehmen oder die Voreinstellung nach 2.3.3.2 für den zugehörigen Datentyp für jede Variable, für die kein Anfangswert durch den Anwender angegeben wurde.
- 3) Nichtgepufferte Variable müssen mit den anwenderspezifisierten Anfangswerten oder mit der Voreinstellung nach 2.3.3.2 für den zugehörigen Datentyp für jede Variable, für die kein Anfangswert durch den Anwender angegeben wurde, initialisiert werden.
- 4) Variable, die *Eingänge* eines SPS-Systems nach IEC 1131-1 darstellen, müssen in einer implementierungsabhängigen Weise initialisiert werden.

2.4.3 Deklaration

Jede Typdeklaration einer SPS-Programm-Organisationseinheit (d. h. jede Deklaration eines *Programms*, einer *Funktion* oder eines *Funktionsbausteins*, wie in Unterabschnitt 2.5 definiert) muß an seinem Anfang mindestens einen *Deklarationsteil* enthalten, der die Typen (und nötigenfalls den physikalischen oder logischen Speicherort) der Variablen festlegt, die in dieser Organisationseinheit benutzt werden. Dieser Deklarationsteil muß die Textform eines der Schlüsselwörter VAR, VAR_INPUT oder VAR_OUTPUT, wie in Tabelle 16 definiert, besitzen; dabei folgt im Falle von VAR nullmal oder einmal das Bestimmungszeichen RETAIN oder CONSTANT; im Falle von VAR_OUTPUT folgt nullmal oder einmal das Bestimmungszeichen RETAIN; darauf folgen jeweils eine oder mehrere Deklarationen, die durch Semikolons getrennt sind und mit dem Schlüsselwort VAR END

abgeschlossen sind. Wenn eine SPS die Deklaration von Anfangswerten von Variablen durch den Anwender unterstützt, dann muß diese Deklaration in der Art im Deklarationsteil (bzw. Teilen) geschehen, wie es in diesem Unterabschnitt definiert ist. Der *Geltungsbereich* (Bereich der Gültigkeit) der Deklarationen, die im Deklarationsteil enthalten sind, muß *lokal* zu der Programm-Organisationseinheit sein, in der der Deklarationsteil enthalten ist. D. h., die deklarierten Variablen dürfen nicht von anderen Organisationseinheiten aus erreichbar sein; ausgenommen ist die explizite Parameterübergabe über Variable, die vorher als *Eingänge* und *Ausgänge* dieser Einheiten deklariert wurden. Die einzige Ausnahme von dieser Regel ist der Fall von Variablen, die als *global* deklariert wurden, wie in 2.7.1 definiert ist. Solche Variablen sind von einer Programm-Organisationseinheit aus nur über eine VAR_EXTERNAL-Deklaration erreichbar. Der Typ einer Variablen, die in einem VAR_EXTERNAL-Block deklariert ist, muß mit dem Typ übereinstimmen, der im VAR_GLOBAL-Block des zugehörigen *Programms*, der *Konfiguration* oder der *Ressource* deklariert ist.

Tabelle 16: Schlüsselwörter für Variablen-Deklaration

Schlüsselwort	Gebrauch der Variablen
VAR	Innerhalb der Organisationseinheit
VAR_INPUT	Von außerhalb geliefert, nicht innerhalb der Organisationseinheit änderbar
VAR_OUTPUT	Von der Organisationseinheit nach außen geliefert
VAR_IN_OUT	Von außerhalb geliefert, kann innerhalb der Organisationseinheit geändert werden ANMERKUNG: Beispiele für den Gebrauch dieser Variablen sind in Abbildung 11b und 12 angegeben.
VAR_EXTERNAL	Von der Konfiguration geliefert via VAR_GLOBAL (2.7.1), kann innerhalb der Organisationseinheit geändert werden
VAR_GLOBAL	Deklaration von globalen Variablen (2.7.1)
VAR_ACCESS	Deklaration von einem Zugriffspfad (2.7.1)
RETAIN	Gepufferte Variablen (siehe vorangehender Text)
CONSTANT	Konstante (Variable kann nicht geändert werden)
AT	Zuweisung des Speicherorts (siehe 2.4.3.1)
ANMERKUNG: Der Gebrauch dieser Schlüsselwörter ist eine Eigenschaft der Programm-Organisationseinheit oder des Konfigurationselementes, in dem sie benutzt werden, siehe Unterabschnitte 2.5 und 2.7.	

2.4.3.1 Typzuweisung

Wie in Tabelle 17 gezeigt, muß die Konstruktion VAR...END_VAR angewendet werden zur Festlegung von Datentypen und Pufferungsverhalten von direkt dargestellten Variablen. Diese Konstruktion muß auch zur Festlegung von Datentypen, Pufferverhalten und (falls nötig, nur in *Programmen*) der physikalischen und logischen Speicherorte für symbolisch dargestellte Einzel- und Multiement-Variable benutzt werden. Der Gebrauch der Konstruktionen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT ist in Unterabschnitt 2.5 definiert.

Die Zuordnung einer physikalischen oder logischen Adresse zu einer symbolisch dargestellten Variablen muß durch Anwendung des Schlüsselworts AT durchgeführt werden. Wo eine derartige Zuordnung nicht gemacht wird, muß eine automatische Zuweisung dieser Variablen zu einem geeigneten Speicherort im Speicher der SPS erfolgen.

2.4.3.2 Anfangswert-Zuweisung

Die Konstruktion VAR...END_VAR, die in Tabelle 18 gezeigt ist, muß benutzt werden, um die Anfangswerte von direkt dargestellten Variablen darzustellen. Diese Konstruktion muß auch benutzt werden, um die Anfangswerte von symbolisch dargestellten Einfach- und Multiement-Variablen zuzuweisen. (Der Gebrauch der Konstruktionen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT ist in 2.5 definiert.)

Anfangswerte können nicht in VAR_EXTERNAL-Deklarationen angegeben werden.

Während der Initialisierung von Feldern, wenn das Feld mit einer Liste von Initialisierungsvariablen gefüllt wird, muß sich der Feldindex ganz rechts am schnellsten ändern.

Runde Klammern können als ein Wiederholungsfaktor in Initialisierungslisten für Felder benutzt werden; z. B. ist "2(1,2,3)" gleichbedeutend mit der Initialisierungsfolge "1,2,3,1,2,3".

Falls die Anzahl von Anfangswerten, die in der Initialisierungsliste angegeben ist, die Anzahl der Feldeinträge überschreitet, muß die Überschreitung der (rechten) Anfangswerte ignoriert werden. Falls die Anzahl der Anfangswerte geringer als die Anzahl der Feldeinträge ist, müssen die übriggebliebenen Feldeinträge mit den voreingestellten Werten des zugehörigen

Tabelle 17: Eigenschaften der Typzuweisung für Variable

Nr	Eigenschaft/Beispiele	
1	Deklaration von direkt dargestellten nichtgepufferten Variablen	
	VAR AT %IW6.2 : WORD ; AT %MW6 : INT ; END_VAR	16-Bit-Folge (Anmerkung 2) 16-Bit-Integer, Anfangswert = 0
2	Deklaration von direkt dargestellten gepufferten Variablen	
	VAR RETAIN AT %QW5 : WORD ; END_VAR	Beim Kaltstart wird %QW5 mit einer 16-Bit-Folge mit dem Wert 0 initialisiert.
3	Deklaration von Speicherorten bei symbolischen Variablen	
	VAR_GLOBAL LIM_SW_5 AT %IX27 : BOOL ; CONV_START AT %QX25 : BOOL ; TEMPERATURE AT %IW28 : INT ; END_VAR	Weist Eingangsbit 27 der booleschen Variablen LIM_SW_5 zu. (Anmerkung 2) Weist Ausgangsbit 25 der booleschen Variablen CONV_START zu. Weist Eingangswort 28 der ganzzahligen Variablen TEMPERATURE zu. (Anmerkung 2)
4	Zuweisung von Speicherort bei Feld	
	VAR INARRAY AT %IW6 : ARRAY [0..9] OF INT ; END_VAR	Deklariert ein Feld von 10 ganzen Zahlen, die einem zusammenhängenden Eingangsspeicherort ab %IW6 zugeteilt werden. (Anmerkung 2)
5	Automatische Speicherezuteilung für symbolische Variable	
	VAR CONDITION_RED : BOOL ; IBOUNCE : WORD ; MYDUB : DWORD ; AWORD, BWORD, CWORD : INT ; MYSTR : STRING(10) ; END_VAR	Teilt ein Bit der booleschen Variablen CONDITION_RED zu. Teilt ein Wort der Variablen IBOUNCE (16-Bit-Folge) zu. Teilt ein Doppelwort der Variablen MYDUB (32-Bit-Folge) zu. Teilt 3 separate Wörter den ganzzahligen Variablen AWORD, BWORD und CWORD zu. Teilt Speicher zu, der eine Zeichenfolge mit einer Maximallänge von 10 enthält. Nach der Initialisierung hat die Folge die Länge 0 und enthält die "leere Folge".
6	Deklaration für Feld	
	VAR THREE : ARRAY [1..5,1..10,1..8] OF INT ; END_VAR	Teilt für ein dreidimensionales Feld von ganzen Zahlen 400 Speicherwörter zu.
7	Deklaration für gepuffertes Feld	
	VAR RETAIN RTBT : ARRAY [1..2,1..3] OF INT ; END_VAR	Deklariert das gepufferte Feld RTBT mit den "Kaltstart"-Anfangswerten 0 für alle Elemente.
8	Deklaration für strukturierte Variablen	
	VAR MODULE_8_CONFIG : ANALOG_16_INPUT_CONFIGURATION ; END_VAR	Deklaration einer Variablen mit abgeleitetem Datentyp (siehe Tabelle 12)
ANMERKUNG 1: Die Eigenschaften 1 bis 4 können nur in den Deklarationen von PROGRAM und VAR_GLOBAL benutzt werden, wie in 2.5.3 bzw. 2.7.1 definiert. ANMERKUNG 2: Die Initialisierung von Systemeingängen ist implementierungsabhängig, siehe Unterabschnitt 2.4.2.		

Tabelle 18: Eigenschaften der Zuweisung von Anfangswerten für Variable

Nr	Eigenschaft/Beispiele	
1	Initialisierung von direkt dargestellten nichtgepufferten Variablen	
	VAR AT %QX5.1 : BOOL := 1 ; AT %MW6 : INT := 8 ; END_VAR	Boolescher Typ, Anfangswert = 1 Initialisiert ein Speicherwort mit der ganzen Zahl 8
2	Initialisierung von direkt dargestellten gepufferten Variablen	
	VAR RETAIN AT %QW5 : WORD := 16#FF00 ; END_VAR	Beim Kaltstart werden die 8 höchstwertigen Bits der 16-Bit-Folge des Ausgangsworts 5 mit 1 initialisiert und die 8 niedrigstwertigen Bits mit 0
3	Zuweisung von Speicherort und Anfangswert für symbolische Variablen	
	VAR VALVE_POS AT %QW28 : INT := 100 ; END_VAR	Weist das Ausgangswort 28 der ganzzahligen Variablen VALVE_POS mit einem Anfangswert 100 zu
4	Zuweisung von Speicherort und Initialisierung für Feld	
	VAR OUTARY AT %QW6 : ARRAY [0..9] OF INT := 10(1) ; END_VAR	Deklariert ein Feld von 10 ganzen Zahlen, die dem zusammenhängenden Ausgangsbereich, beginnend ab %QW6, zugeteilt werden, jeweils mit einem Anfangswert von 1
5	Initialisierung von symbolischen Variablen	
	VAR MYBIT : BOOL := 1 ; OKAY : STRING(10) := 'OK' ; END_VAR	Teilt ein Bit der booleschen Variablen MYBIT zu, mit einem Anfangswert von 1 Teilt Speicher zu, der eine Zeichenfolge mit einer Maximallänge von 10 enthält. Nach Initialisierung hat die Folge die Länge 2 und enthält die 2-Byte-Zeichenfolge 'OK' im ISO/IEC-646-Zeichensatz in einer für das Ausdrucken als Zeichenfolge geeigneten Anordnung.
6	Initialisierung für Feld	
	VAR BITS : ARRAY [0..7] OF BOOL := 1,1,0,0,0,1,0,0 ; TBT : ARRAY [1..2,1..3] OF INT := 1,2,3(4),6 ; END_VAR	Teilt 8 Speicherbits die Anfangswerte zu BITS[0] := 1, BITS[1] := 1,..., BITS[6] := 0, BITS[7] := 0 Teilt einem 2-mal-3-Feld TBT von ganzen Zahlen die Anfangswerte zu TBT[1,1] := 1, TBT[1,2] := 2, TBT[1,3] := 4, TBT[2,1] := 4, TBT[2,2] := 4, TBT[2,3] := 6
7	Deklaration und Initialisierung für gepuffertes Feld	
	VAR RETAIN RTBT : ARRAY [1..2,1..3] OF INT := 1,2,3(4) ; END_VAR	Deklariert das gepufferte Feld RTBT mit den "Kaltstart"-Anfangswerten RTBT[1,1] := 1, RTBT[1,2] := 2, RTBT[1,3] := 4, RTBT[2,1] := 4, RTBT[2,2] := 4, RTBT[2,3] := 0
8	Initialisierung von strukturierten Variablen	
	VAR MODULE_8_CONFIG : ANALOG_16_INPUT_CONFIGURATION (SIGNAL_TYPE := DIFFERENTIAL, CHANNEL[5].RANGE := BIPOLAR_10_V, CHANNEL[5].MIN_SCALE := 0, CHANNEL[5].MAX_SCALE := 500) ; END_VAR	Initialisierung einer Variablen mit abgeleitetem Datentyp (siehe Tabelle 12)
9	Initialisierung von Konstanten	
	VAR CONSTANT PI : REAL := 3.141592 ; END_VAR	
ANMERKUNG: Die Eigenschaften 1 bis 4 können nur in Deklarationen von PROGRAM und VAR_GLOBAL benutzt werden, wie in 2.5.3 bzw. 2.7.1 definiert ist.		

Datentyps aufgefüllt werden. In beiden Fällen muß der Anwender warnend auf diesen Vorgang während der Aufbereitung des Programms hingewiesen werden.

Wenn eine Variable als ein abgeleiteter strukturierter Datentyp nach 2.3.3.1 deklariert wird, können Anfangswerte für die Elemente der Variablen in einer eingeklammerten Liste mit nachfolgendem Datentyp-Bezeichner deklariert werden, wie in Tabelle 18 gezeigt ist. Elemente, für die keine Anfangswerte in der Liste der Anfangswerte angegeben sind, müssen die vor-eingestellten Anfangswerte besitzen, die für diese Elemente in der Datentyp-Deklaration vereinbart sind.

2.5 Programm-Organisationseinheiten

In diesem Teil von IEC 1131 werden folgende Programm-Organisationseinheiten definiert: die *Funktion*, der *Funktionsbaustein* und das *Programm*. Diese Programm-Organisationseinheiten können vom Hersteller geliefert werden oder vom Anwender mit den Mitteln programmiert werden, die in diesem Teil der Norm definiert sind.

Programm-Organisationseinheiten dürfen nicht *rekursiv* sein; d. h., der Aufruf einer Programm-Organisationseinheit darf nicht den Aufruf einer weiteren Programm-Organisationseinheit desselben Typs verursachen.

2.5.1 Funktionen

Für die Zwecke der SPS-Programmiersprachen ist eine *Funktion* als eine Programm-Organisationseinheit definiert, die bei Ausführung genau ein Datenelement liefert (das mehrere Werte haben darf, wie z. B. ein Feld oder eine Struktur) und deren Aufruf in Textsprachen als ein Operand in einem Ausdruck benutzt werden kann. So könnten z. B. die SIN- und COS-Funktionen, wie in Bild 4 gezeigt, angewendet werden.

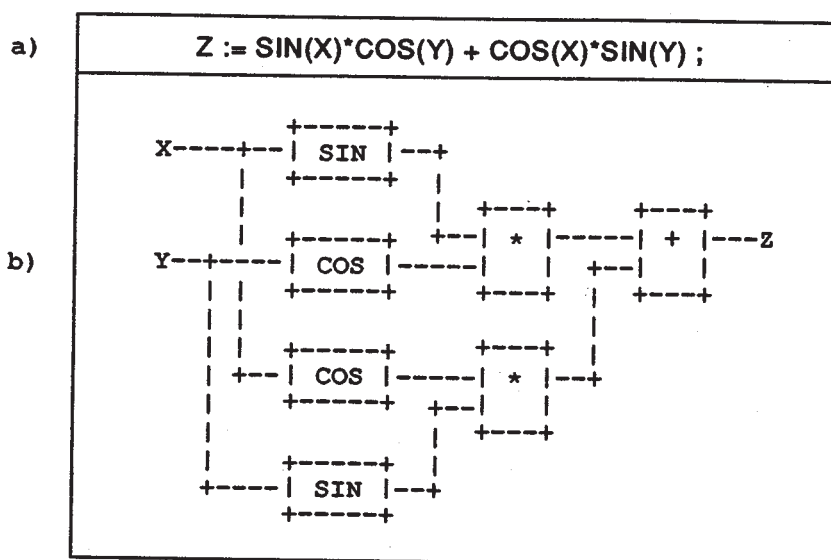


Bild 4: Beispiele für den Gebrauch von Funktionen

a) Strukturierte Text (ST); siehe Unterabschnitt 3.3

b) Funktionsbaustein-Sprache (FBS); siehe Unterabschnitt 4.3

Funktionen dürfen keine internen Zustandsinformationen beinhalten; d. h., der Aufruf einer Funktion mit denselben Argumenten (Eingangsparametern) muß denselben Wert (Ausgang) liefern.

Jeder Funktionstyp, der bereits deklariert wurde, kann in der Deklaration einer weiteren Programm-Organisationseinheit benutzt werden, wie in Bild 3 gezeigt.

2.5.1.1 Darstellung

Funktionen und ihr Aufruf können entweder graphisch oder in Textform dargestellt werden.

In den graphischen Sprachen, die in Abschnitt 4 dieses Teils definiert sind, müssen Funktionen als graphische Schaltzeichen (Block) gemäß den folgenden Regeln dargestellt sein:

- 1) Die Form des Schaltzeichens muß rechteckig oder quadratisch sein.
- 2) Die Größe und Lage des Schaltzeichens darf in Abhängigkeit von der Anzahl der Eingänge, Ausgänge und weiteren Informationen, die anzuzeigen sind, variieren.
- 3) Die Richtung der Bearbeitung muß von links nach rechts durch das Schaltzeichen verlaufen (Eingangsparameter auf der linken Seite und Ausgangsparameter auf der rechten Seite).
- 4) Der Name oder das Symbol der Funktion, die wie im folgenden festgelegt sind, müssen innerhalb des Schaltzeichens angegeben sein.
- 5) Es muß die Möglichkeit bestehen, daß die Namen der formalen Eingangsparameter auf der linken Innenseite des Schaltzeichens erscheinen, wenn das Schaltzeichen folgendes darstellt:

- eine der Standardfunktionen, die in Abschnitt 2.5.1.5 definiert sind, wenn die angegebene graphische Form die formalen Parameter enthält; oder
 - eine zusätzliche Funktion, die, wie in Abschnitt 2.5.1.3 festgelegt, deklariert ist.
- 6) Da der Name der Funktion für die Zuweisung des Ausgangswertes benutzt wird, wie in 2.5.1.3 festgelegt ist, darf kein formaler Ausgangsparameter-Name auf der rechten Innenseite des Schaltzeichens angegeben werden.
- 7) Die Verbindungen der Aktualparameter müssen durch Signalfluß-Linien angezeigt werden.
- 8) Die Negation von booleschen Signalen muß durch Angabe eines offenen Kreises angezeigt werden, und zwar direkt außerhalb der Verbindung der Eingangs- oder Ausgangsline mit dem Schaltzeichen. Dies muß im ISO/IEC-646-Zeichensatz durch den Großbuchstaben "O" dargestellt werden, wie es in Tabelle 19 gezeigt ist.
- 9) Der Ausgang einer graphisch dargestellten Funktion muß als eine einfache Linie auf der rechten Seite des Schaltzeichens dargestellt werden; das gilt auch, wenn der Ausgang ein Mehrfach-Element ist.

Tabelle 19: Graphische Negation von booleschen Signalen

Nr	Eigenschaft	Darstellung
1	Negierter Eingang	
2	Negierter Ausgang	
ANMERKUNG: Falls eine der beiden Eigenschaften für Funktionen unterstützt wird, muß sie auch für Funktionsbausteine, die in 2.5.2 definiert sind, unterstützt werden und umgekehrt.		

Wenn, wie in Bild 5 gezeigt, ein Formalparameter-Name in der Definition einer Standardfunktion nach 2.5.1.5 auftritt, muß der Formalparameter-Name auch beim Aufruf der Funktion in Textform benutzt werden. Im letzteren Fall können die Formalparameter-Namen und die zugehörigen Aktualwerte in beliebiger Reihenfolge angegeben werden.

Die Darstellung von Funktionen in Textsprachen muß, wie in Abschnitt 3 dieses Teils festgelegt, erfolgen.

Beispiel	Erläuterung
<pre> +-----+ ADD B--- ---A C--- D--- +-----+ </pre>	Graphischer Gebrauch der "ADD"-Funktion (siehe 2.5.1.5.2) (FBS-Sprache; siehe 4.3) (Keine Formalparameter-Namen)
A := ADD (B, C, D) ;	Textueller Gebrauch der "ADD"-Funktion (ST-Sprache; siehe 3.3)
<pre> +-----+ SHL B--- IN ---A C--- IN +-----+ </pre>	Graphischer Gebrauch der "SHL"-Funktion (siehe 2.5.1.5.3) (FBS-Sprache; siehe 4.3) (Formalparameter-Namen)
A := SHL (IN := B, N := C) ;	Textueller Gebrauch der "SHL"-Funktion (ST-Sprache; siehe 3.3)

Bild 5: Gebrauch der Formalparameter-Namen

2.5.1.2 Ausführungssteuerung

Wie in Tabelle 20 gezeigt, müssen ein zusätzlicher boolescher (Freigabe-)Eingang "EN" (Enable) und Ausgang "ENO" (Enable out) bei den Funktionen in der Sprache KOP, definiert in 4.2, benutzt werden. Ihr Gebrauch muß auch in der Sprache FBS möglich sein, die in diesem Teil definiert ist. Diese Variablen sind in jeder Funktion als verfügbar gemäß den folgenden impliziten Deklarationen zu betrachten:

VAR_INPUT EN : BOOL := 1 ; END_VAR

VAR_OUTPUT ENO : BOOL ; END_VAR

Wenn diese Variablen benutzt werden, muß die Ausführung der Operationen, die durch diese Funktion definiert sind, nach folgenden Regeln gesteuert werden:

- 1) Falls der Wert von EN gleich FALSE (0) ist, wenn die Funktion aufgerufen wird, dürfen die Operationen, die durch den Funktionsrumpf definiert sind, nicht ausgeführt werden, und der Wert von "ENO" muß durch das SPS-System auf FALSE (0) zurückgesetzt werden.
- 2) Andernfalls muß der Wert von ENO durch das SPS-System auf TRUE (1) gesetzt werden, und die Operationen, die durch den Funktionsrumpf definiert sind, müssen ausgeführt werden. Diese Operationen können die Zuweisung eines booleschen Wertes an ENO einschließen.
- 3) Falls einer der Fehler, die im Anhang E definiert sind, während der Ausführung einer der Standardfunktionen, die in 2.5.1.5 definiert sind, auftritt, muß der ENO-Ausgang dieser Funktion durch das SPS-System auf FALSE (0) rückgesetzt werden.

ANMERKUNG: Der Gebrauch der ENO-Ausgänge ist eine zulässige Ausnahme von der Regel, daß die Ausführung einer Funktion genau einen Ausgangswert liefert.

Tabelle 20: Der Gebrauch des "EN"-Eingangs und "ENO"-Ausgangs

Nr	Eigenschaft	Beispiel
1	Gebrauch von "EN" und "ENO" Erforderlich bei KOP (Kontaktplan-Sprache) (siehe 4.2)	<pre> +-----+ ADD_EN + ADD_OK +---+---+ EN ENO +---+ () ---+ A--- ---C B--- +-----+ </pre>
2	Gebrauch von "EN" und "ENO" Zusätzlich möglich bei FBS (Funktionsbaustein-Sprache) (siehe 4.3)	<pre> +-----+ + ADD_EN-- EN ENO ---ADD_OK A--- ---C B--- +-----+ </pre>
3	FBS ohne "EN" und "ENO"	<pre> +-----+ A--- + ---C B--- +-----+ </pre>

2.5.1.3 Deklaration

Eine Funktion muß in Textform oder in Graphik deklariert werden.

Wie in Bild 6 veranschaulicht, muß die Text-Deklaration einer Funktion aus folgenden Elementen bestehen:

- 1) dem Schlüsselwort **FUNCTION**, gefolgt von einem Bezeichner, der den Namen der zu deklarierenden Funktion festlegt, einem Doppelpunkt (:) und dem Datentyp des Wertes, der von der Funktion zurückgegeben wird;
- 2) einem Konstrukt **VAR_INPUT ... END_VAR**, wie in 2.4.2 definiert, das die Namen und Typen der Eingangsparameter der Funktion festlegt;
- 3) falls erforderlich, einem Konstrukt **VAR ... END_VAR**, das die Namen und Typen der internen Variablen der Funktion festlegt;
- 4) einem *Funktionsrumpf*, geschrieben in einer der in diesem Teil definierten Sprachen oder einer weiteren Programmiersprache, wie in 1.4.3 definiert, der die Operationen festlegt, die auf dem (den) Eingangsparameter(n) ausgeführt werden, um einen oder mehrere Werte einer Variablen zuzuweisen, die denselben Namen wie die Funktion hat und die den Wert darstellt, der von der Funktion zurückzugeben ist;
- 5) dem beendenden Schlüsselwort **END_FUNCTION**.

Wie in Bild 6 veranschaulicht, muß die graphische Deklaration einer Funktion aus folgenden Elementen bestehen:

- 1) den einklammernden Schlüsselwörtern FUNCTION und END_FUNCTION oder einem graphischen Äquivalent;
- 2) einer graphischen Festlegung des Funktionsnamens und den Namen und Typen der Eingänge und Ausgänge der Funktion;
- 3) einer Festlegung der Namen und Typen der internen Variablen, die in der Funktion benutzt werden; dazu kann z. B. das Konstrukt VAR ... END_VAR benutzt werden;
- 4) einem Funktionsrumpf, wie oben definiert.

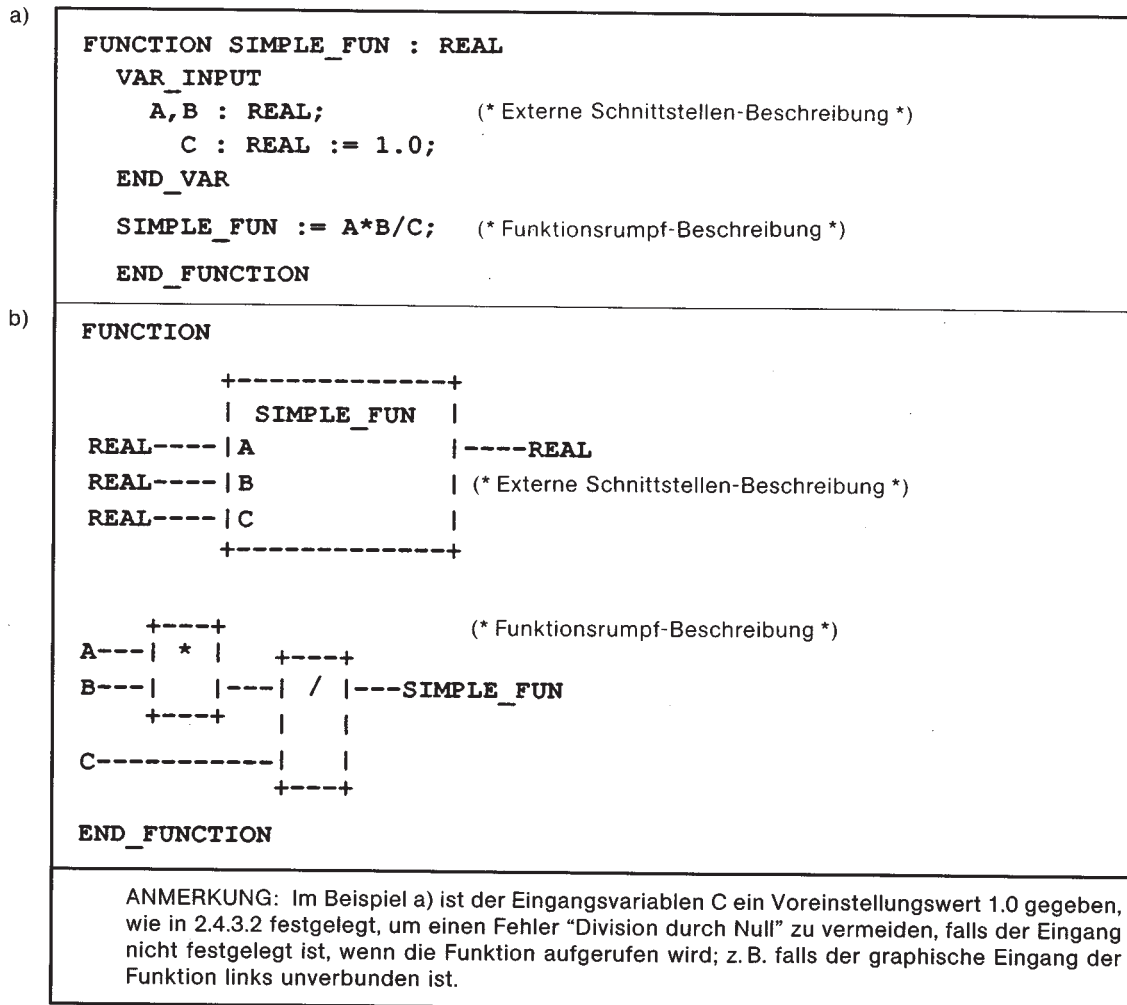


Bild 6: Beispiele für Deklarationen einer Funktion

a) Deklaration in Textform in der Sprache ST (siehe 3.3)

b) Graphische Deklaration in der Sprache FBS (siehe 4.3)

2.5.1.4 Typangabe, Überladen und Typumwandlung

Eine Funktion oder eine Operation nennt man *überladen*, wenn sie auf Eingangsdatenelemente verschiedenen Typs arbeiten kann, und zwar innerhalb eines allgemeinen Typs, wie in 2.3.2 definiert. Zum Beispiel kann eine überladene Additionsfunktion vom Typ ANY_NUM mit Daten vom Typ LREAL, REAL, DINT, INT und SINT arbeiten.

Wenn ein SPS-System eine überladene Operation oder Funktion unterstützt, muß diese Operation oder Funktion auf alle Datentypen eines gegebenen allgemeinen Typs anwendbar sein, die von diesem System unterstützt werden. Wenn z. B. ein SPS-System die überladene Funktion ADD und die Datentypen SINT, INT und REAL unterstützt, dann muß das System die Funktion ADD mit den Eingängen vom Datentyp SINT, INT und REAL unterstützen.

Wenn für eine Funktion, die normalerweise einen überladenen Operator darstellt, ein bestimmter Typ angegeben werden soll, d. h., wenn die Typen ihrer Eingänge und Ausgänge auf einen bestimmten Untertyp beschränkt sein sollen, dann muß dies durch Anhängen eines "Unterstrich"-Zeichens getan werden, auf das der gewünschte Typ folgt, wie in Tabelle 21 gezeigt ist.

Tabelle 21: Typangabe und überladene Funktionen

Nr	Eigenschaft	Beispiel
1	Überladene Funktionen (typunabhängig)	<pre> +-----+ ADD +-----+ ANY_NUM----- -----ANY_NUM ANY_NUM----- ----- . ----- ----- . ----- ----- ANY_NUM----- ----- +-----+ </pre>
2	Funktionen mit Typangabe	<pre> +-----+ ADD_INT +-----+ INT----- -----INT INT----- ----- . ----- ----- . ----- ----- INT----- ----- +-----+ </pre>
ANMERKUNG 1: Falls Eigenschaft 2 unterstützt wird, muß der Hersteller eine Tabelle zur Verfügung stellen, die angibt, welche Funktionen in der Implementierung überladen sind und welche mit Typ angegeben sind. ANMERKUNG 2: Anwenderdefinierte Funktionen können nicht überladen sein.		

Wenn alle formalen Eingangsparameter einer Standardfunktion, die in 2.5.1.5 definiert sind, vom selben allgemeinen Typ sind, dann müssen alle Aktualparameter vom selben Typ sein. Falls nötig, können die Typumwandlungsfunktionen, die in 2.5.1.5.1 definiert sind, benutzt werden, um diese Anforderung zu erfüllen. Der Ausgangswert der Funktion muß dann denselben Wert haben wie die Aktualeingänge. Als Ausnahme gelten die Angaben in Tabelle 22. Beispiele für die Anwendung dieser Regel sind in den Bildern 7 und 8 angegeben.

Typdeklaration (ST-Sprache – siehe 3.3)	Gebrauch (FBS-Sprache – siehe 4.3) (ST-Sprache – siehe 3.3)
<pre> VAR A : INT ; B : INT ; C : INT ; END_VAR </pre>	<pre> +---+ A--- + ---C B--- +---+ C := A+B; </pre>
<pre> VAR A : INT ; B : REAL ; C : REAL ; END_VAR </pre>	<pre> +-----+ +---+ A--- INT_TO_REAL --- + ---C +-----+ B----- +-----+ C := INT_TO_REAL (A)+B; </pre>
<pre> VAR A : INT ; B : INT ; C : REAL ; END_VAR </pre>	<pre> +---+ +-----+ A--- + --- INT_TO_REAL ---C B--- +-----+ +---+ C := INT_TO_REAL (A+B) ; </pre>
ANMERKUNG: Typumwandlung ist im ersten Beispiel nicht erforderlich.	

Bild 7: Beispiele für explizite Typumwandlung bei überladenen Funktionen

Typdeklaration (ST-Sprache – siehe 3.3)	Gebrauch (FBS-Sprache – siehe 4.3) (ST-Sprache – siehe 3.3)
VAR A : INT ; B : INT ; C : INT ; END_VAR	<pre> +-----+ A--- ADD_INT ---C B--- +-----+ C := ADD_INT (A+B) ; </pre>
VAR A : INT ; B : REAL ; C : REAL ; END_VAR	<pre> +-----+ +-----+ A--- INT_TO_REAL --- ADD_REAL ---C +-----+ B----- +-----+ C := ADD_REAL (INT_TO_REAL (A) , B) ; </pre>
VAR A : INT ; B : INT ; C : REAL END_VAR	<pre> +-----+ +-----+ A--- ADD_INT --- INT_TO_REAL ---C +-----+ B--- +-----+ C := INT_TO_REAL (ADD_INT (A, B)) ; </pre>
ANMERKUNG: Typumwandlung ist im ersten Beispiel nicht erforderlich.	

Bild 8: Beispiele von expliziter Typumwandlung bei Funktionen mit Typangabe

2.5.1.5 Standardfunktionen

In diesem Unterabschnitt sind die Definitionen der Funktionen angegeben, die allen SPS-Programmiersprachen gemeinsam sind. Wo graphische Darstellungen der Standardfunktionen in diesem Unterabschnitt gezeigt werden, dürfen gleichwertige Deklarationen in Textform geschrieben werden, wie in 2.5.1.3 festgelegt ist.

Eine Standardfunktion, die in diesem Unterabschnitt als *erweiterbar* festgelegt ist, darf eine variable Anzahl von Eingängen haben und muß betrachtet werden, als wende sie die angezeigte Operation der Reihe nach auf jeden Eingang an; z. B. muß die erweiterbare Addition an ihrem Ausgang die Summe aller Eingänge liefern. Die maximale Anzahl der Eingänge einer erweiterbaren Funktion ist ein implementierungsabhängiger Parameter.

2.5.1.5.1 Funktionen zur Typumwandlung

Wie in Tabelle 22 gezeigt, müssen die Typumwandlungsfunktionen die Form *_TO_** haben; wobei "*" der Typ der Eingangsvariablen IN ist und "**" der Typ der Ausgangsvariablen OUT ist; z. B. INT_TO_REAL.

2.5.1.5.2 Numerische Funktionen

Die standardmäßige graphische Darstellung, die Funktionsnamen, die Typen der Eingangs- und Ausgangsvariablen und die Funktionsbeschreibungen der Funktionen mit einer einzelnen numerischen Variablen müssen nach Tabelle 23 angegeben werden. Diese Funktionen müssen mit den definierten allgemeinen Typen überladen sein, und sie können mit Typangabe nach 2.5.1.4 definiert sein. Für diese Funktionen müssen die Typen von Eingang und Ausgang gleich sein.

Die standardmäßige graphische Darstellung, die Funktionsnamen und -symbole und die Beschreibungen der arithmetischen Funktionen mit zwei oder mehr Variablen müssen nach Tabelle 24 angegeben werden. Diese Funktionen müssen mit allen numerischen Typen überladen sein und können mit bestimmten Typen angegeben werden, wie in 2.5.1.4 definiert ist.

2.5.1.5.3 Bit-Folge-Funktionen

Die standardmäßige graphische Darstellung, die Funktionsnamen und Beschreibungen von Schiebefunktionen für eine einzelne Bit-Folge-Variable müssen, wie in Tabelle 25 definiert, angegeben werden. Diese Funktionen müssen mit allen Bit-Folge-Typen überladen sein, und sie können mit bestimmten Typen angegeben werden, wie in 2.5.1.4 definiert ist.

Die standardmäßige graphische Darstellung, die Funktionsnamen und -symbole und die Beschreibungen der bitweisen booleschen Funktionen müssen, wie in Tabelle 26 definiert, angegeben werden. Diese Funktionen müssen erweiterbar sein, ausgenommen ist NOT (nicht), und sie müssen mit allen Bit-Folge-Typen überladen sein. Sie können mit bestimmtem Typ angegeben werden, wie in 2.5.1.4 definiert ist.

Tabelle 22: Eigenschaften der Funktionen zur Typumwandlung

Nr	Graphische Form	Beispiel	Anmerkung
1	<pre> +-----+ *--- *_TO_* ---** +-----+ </pre> (*) Eingangsdatentyp, z. B. INT (**) Ausgangsdatentyp, z. B. REAL (*_TO_*) Funktionsname, z. B. INT_TO_REAL	A := INT_TO_REAL(B) ;	1 2 5
2	<pre> +-----+ ANY_REAL--- TRUNC ---ANY_INT +-----+ </pre>	A := TRUNC(B) ;	3
3	<pre> +-----+ ANY_BIT-- BCD_TO_* ---ANY_INT +-----+ </pre>	A := BCD_TO_INT(B) ;	4
4	<pre> +-----+ ANY_INT-- *_TO_BCD ---ANY_BIT +-----+ </pre>	A := INT_TO_BCD(B) ;	4

ANMERKUNG 1: Eine Aussage bezüglich Übereinstimmung mit Eigenschaft 1 dieser Tabelle muß eine Liste der spezifischen unterstützten Typumwandlungen und eine Aussage der Wirkungen, die bei der Bearbeitung der jeweiligen Umwandlung auftreten, einschließen.

ANMERKUNG 2: Die Umwandlung vom Typ REAL oder LREAL nach SINT, INT, DINT oder LINT muß auf die nächste ganze Zahl runden, gemäß der Festlegungen nach IEC 559, z. B.:

REAL_TO_INT (1.6) ist gleich 2
REAL_TO_INT (-1.6) ist gleich -2
REAL_TO_INT (1.5) ist gleich 2
REAL_TO_INT (-1.5) ist gleich -2
REAL_TO_INT (1.4) ist gleich 1
REAL_TO_INT (-1.4) ist gleich -1
REAL_TO_INT (2.5) ist gleich 2
REAL_TO_INT (-2.5) ist gleich -2

ANMERKUNG 3: Die Funktion TRUNC muß zum Abschneiden Richtung Null von REAL oder LREAL benutzt werden, dabei wird einer der ganzzahligen Typen geliefert, z. B.:

TRUNC (1.6) ist gleich 1
TRUNC (-1.6) ist gleich -1
TRUNC (1.4) ist gleich 1
TRUNC (-1.4) ist gleich -1

ANMERKUNG 4: Die Umwandlungsfunktionen *_TO_BCD und BCD_TO_* sind für Umwandlungen zwischen Variablen vom Typ BYTE, WORD, DWORD und LWORD und Variablen vom Typ SINT, INT und DINT (durch "*" dargestellt), wenn die entsprechenden Bit-Folge-Variablen Daten beinhalten, die im BCD-Format kodiert sind. Z. B. wird der Wert von INT_TO_BCD (25) gleich 2#0010_0101 sein, und der Wert von BCD_TO_INT (2#0011_0110_1001) wird gleich 369 sein.

ANMERKUNG 5: Wenn ein Eingang oder Ausgang einer Typumwandlungsfunktion vom Typ STRING ist, müssen die Daten der Zeichenfolge mit der äußeren Darstellung der entsprechenden Daten, wie in 2.2 festgelegt, mit dem ISO/IEC-646-Zeichensatz übereinstimmen.

ANMERKUNG 6: Anwendungsbeispiele sind in der Sprache ST angegeben, die in 3.3 definiert ist.

Tabelle 23: Standardfunktionen mit einer numerischen Variablen

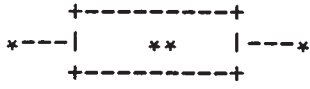
Graphische Form			Beispiel zur Benutzung
 (*) – Eingang/Ausgang (E/A)-Typ (**) – Funktionsname			A := SIN(B) ; (ST-Sprache – siehe 3.3)
Nr	Name	E/A-Typ	Beschreibung
Allgemeine Funktionen			
1	ABS	ANY_NUM	Absolutwert
2	SQRT	ANY_REAL	Quadratwurzel
Logarithmus-Funktionen			
3	LN	ANY_REAL	Natürlicher Logarithmus
4	LOG	ANY_REAL	Logarithmus zur Basis 10
5	EXP	ANY_REAL	Exponentialfunktion (e-Funktion)
Trigonometrische Funktionen			
6	SIN	ANY_REAL	Sinus, mit Eingang im Bogenmaß
7	COS	ANY_REAL	Cosinus mit Eingang im Bogenmaß
8	TAN	ANY_REAL	Tangens mit Eingang im Bogenmaß
9	ASIN	ANY_REAL	Arcsin, Hauptwert
10	ACOS	ANY_REAL	Arccos, Hauptwert
11	ATAN	ANY_REAL	Arctan, Hauptwert

Tabelle 24: Arithmetische Standardfunktionen

Graphische Form			Beispiel zur Benutzung
<pre> +-----+ *** ANY_NUM--- ---ANY_NUM ANY_NUM--- . --- . --- ANY_NUM--- +-----+ </pre> (***) – Name oder Symbol			A := ADD(B,C,D) ; oder A := B+C+D ;
Nr	Name	Symbol (Anmerkung 1)	Beschreibung (Anmerkungen 2 und 8)
Erweiterbare arithmetische Funktionen			
12	ADD	+	OUT := IN1 + IN2 + ... + INn
13	MUL	*	OUT := IN1 * IN2 * ... * INn
Nichterweiterbare arithmetische Funktionen			
14	SUB	-	OUT := IN1 - IN2
15	DIV	/	OUT := IN1 / IN2 (Anmerkung 5)
16	MOD		OUT := IN1 modulo IN2 (Anmerkung 3)
17	EXPT	**	Exponentiation: OUT := IN1 ^{IN2} (Anmerkung 4)
18	MOVE	:=	OUT := IN (Anmerkung 9)
ANMERKUNG 1: Diese Symbole sind als Operatoren in Textsprachen anwendbar, wie in den Tabellen 52 und 55 gezeigt. ANMERKUNG 2: Die Angaben IN1, IN2, ..., INn beziehen sich auf die Eingänge in der Reihenfolge von oben nach unten; OUT bezieht sich auf den Ausgang. ANMERKUNG 3: IN1 und IN2 müssen für diese Funktion den allgemeinen Typ ANY_INT haben. Das Ergebnis der Bearbeitung dieser Funktion muß gleichwertig mit der Ausführung der folgenden Anweisungen in der Sprache ST, wie in 3.3 definiert, sein: IF (IN2 = 0) THEN OUT := 0 ; ELSE OUT := IN1 - (IN1/IN2) * IN2 ; END_IF ANMERKUNG 4: Bei dieser Funktion muß IN1 den Typ ANY_REAL und IN2 den Typ ANY_NUM haben. Der Ausgang muß denselben Typ wie IN1 haben. ANMERKUNG 5: Das Ergebnis der Division von ganzen Zahlen muß eine ganze Zahl vom selben Typ sein, mit Abschneiden Richtung Null. Z. B. 7/3 = 2 und (-7) / 3 = -2. ANMERKUNG 6: Wenn die Darstellung einer Funktion mit Namensangabe unterstützt wird, muß dies durch das Suffix "n" in der Aussage zur Normerfüllung angegeben werden. Z. B. repräsentiert "12n" die Schreibweise "ADD". ANMERKUNG 7: Wenn die Darstellung einer Funktion mit Symbol unterstützt wird, muß dies durch das Suffix "s" in der Aussage zur Normerfüllung angegeben werden. Z. B. repräsentiert "12s" die Schreibweise "+". ANMERKUNG 8: Anwendungsbeispiele und Beschreibungen sind in der Sprache ST angegeben, die in 3.3 definiert ist. ANMERKUNG 9: Die Funktion MOVE hat genau einen Eingang (IN) vom Type ANY und einen Ausgang (OUT) vom Typ ANY.			

Tabelle 25: Standard-Bitschiebe-Funktionen

Graphische Form		Beispiel zur Benutzung
<pre> +-----+ *** ANY_BIT--- IN ---ANY_BIT ANY_BIT--- N +-----+ (***) – Funktionsname </pre>		A := SHL(IN := B, N := 5) ; (ST-Sprache – siehe 3.3)
Nr	Name	Beschreibung
1	SHL	OUT := IN links schieben um N Bits, rechts mit Null füllen
2	SHR	OUT := IN rechts schieben um N Bits, links mit Null füllen
3	ROR	OUT := IN rechts rotieren um N Bits, im Kreis
4	ROL	OUT := IN links rotieren um N Bits, im Kreis
ANMERKUNG: Die Angabe "OUT" bezieht sich auf den Funktionsausgang.		

2.5.1.5.4 Funktionen für Auswahl und Vergleich

Die Funktionen für Auswahl und Vergleich müssen mit allen Datentypen überladen sein. Die standardmäßigen graphischen Darstellungen, Funktionsnamen und Beschreibungen von Auswahlfunktionen müssen gemäß Tabelle 27 erfolgen.

Die standardmäßige graphische Darstellung, die Funktionsnamen und -symbole und die Beschreibungen der Vergleichsfunktionen müssen, wie in Tabelle 28 definiert, erfolgen. Alle Vergleichsfunktionen (ausgenommen NE) müssen erweiterbar sein.

Vergleiche von Bit-Folgen müssen bitweise vom höchstwertigen zum niedrigstwertigen Bit erfolgen, und kürzere Bit-Folgen müssen betrachtet werden, als seien sie links mit Nullen aufgefüllt, wenn sie mit längeren Bit-Folgen verglichen werden; d. h., der Vergleich von Bit-Variablen muß das gleiche Ergebnis haben wie der Vergleich von vorzeichenlosen ganzzahligen Variablen.

Tabelle 26: Bitweise boolesche Standardfunktionen

Graphische Form		Beispiel zur Benutzung	
+-----+ ANY_BIT--- *** ---ANY_BIT ANY_BIT--- . --- . --- ANY_BIT--- +-----+ (***) – Name oder Symbol		A := AND(B,C,D) ; oder A := B & C & D ;	
Nr	Name	Symbol	Beschreibung
5	AND	& (Anmerkung 1)	OUT := IN1 & IN2 & ... & INn
6	OR	>=1 (Anmerkung 2)	OUT := IN1 OR IN2 OR ... OR INn
7	XOR	=2k+1 (Anmerkung 2)	OUT := IN1 XOR IN2 XOR ... XOR INn
8	NOT		OUT := NOT IN1 (Anmerkung 4)
ANMERKUNG 1: Dieses Symbol ist für den Gebrauch als Operator in Textsprachen geeignet, wie in den Tabellen 52 und 55 gezeigt. ANMERKUNG 2: Dieses Symbol ist nicht für den Gebrauch als Operator in Textsprachen geeignet. ANMERKUNG 3: Die Angaben IN1, IN2, ..., INn beziehen sich auf die Eingänge in der Reihenfolge von oben nach unten; OUT bezieht sich auf den Ausgang. ANMERKUNG 4: Die graphische Negation von Signalen vom Typ BOOL kann auch, wie in Tabelle 19 gezeigt, erreicht werden. ANMERKUNG 5: Wenn die Darstellung einer Funktion mit Namensangabe unterstützt wird, muß dies durch das Suffix "n" in der Aussage zur Normerfüllung angegeben werden. Z. B. stellt "5n" die Angabe "AND" dar. ANMERKUNG 6: Wenn die Darstellung einer Funktion mit Symbolangabe unterstützt wird, muß dies durch das Suffix "s" in der Aussage zur Normerfüllung angegeben werden. Z. B. stellt "5s" die Angabe "&" dar. ANMERKUNG 7: Anwendungsbeispiele und Beschreibungen sind in der ST-Sprache angegeben, die in 3.3 definiert ist.			

Tabelle 27: Standardfunktionen für Auswahl

Nr	Graphische Form	Erläuterung/Beispiel
1	<pre> +-----+ SEL BOOL----- G -----ANY ANY----- IN0 ANY----- IN1 +-----+ </pre>	Binäre Auswahl OUT := IN0 falls G = 0 OUT := IN1 falls G = 1 Beispiel A := SEL(G := 0, IN0 := X, IN1 := 5) ;
2a	<pre> +-----+ MAX (Anmerkung 1) --- -----ANY --- (Anmerkung 1) --- +-----+ </pre>	Erweiterbare Maximum-Funktion OUT := MAX (IN1, IN2, ..., INn) Beispiel: A := MAX(B, C, D) ;
2b	<pre> +-----+ MIN (Anmerkung 1) --- -----ANY --- (Anmerkung 1) --- +-----+ </pre>	Erweiterbare Minimum-Funktion OUT := MIN (IN1, IN2, ..., INn) Beispiel: A := MIN(B, C, D) ;
3	<pre> +-----+ LIMIT (Anmerkung 1) -- MN -----ANY (Anmerkung 1) -- IN (Anmerkung 1) -- MX +-----+ </pre>	Begrenzer OUT := MIN(MAX(IN, MN),MX) ; Beispiel: A := LIMIT(IN := B, MN := 0, MX := 5) ;
4	<pre> +-----+ MUX ANY_INT--- K -----ANY ANY--- : --- ANY--- +-----+ </pre>	Erweiterbarer Multiplexer: Wählt einen von "N" Eingängen aus, abhängig vom Eingang K Beispiel: A := MUX(K := 0, IN0 := B, IN1 := C, IN2 := D) ; hat dieselbe Wirkung wie A := B ;
ANMERKUNG 1: Die Eingänge können vom Typ ANY_BIT, ANY_NUM, STRING, ANY_DATE oder TIME sein. Die Regeln der Typumwandlung, die in 2.5.1.4 angegeben sind, müssen für diese Eingänge benutzt werden. ANMERKUNG 2: Die Angaben IN1, IN2, ..., INn beziehen sich in der Reihenfolge von oben nach unten auf die Eingänge; OUT bezieht sich auf den Ausgang. ANMERKUNG 3: Anwendungsbeispiele und Beschreibungen sind in der Sprache ST angegeben, die in 3.3 definiert ist.		

Tabelle 28: Standardfunktionen für Vergleich

Graphische Form			Beispiele
<pre> +-----+ (Anmerkung 1) -- *** ---BOOL (Anmerkung 1) -- +-----+ </pre> (***) – Name oder Symbol			<pre> A := GT(B, C, D) ; oder A := (B>C) & (C>D) ; </pre>
Nr	Name	Symbol	Beschreibung
5	GT	>	Fallende Folge, OUT := (IN1>IN2) & (IN2>IN3) & ... & (INn-1 > INn)
6	GE	>=	Monotone Folge, OUT := (IN1>=IN2) & (IN2>=IN3) & ... & (INn-1 >= INn)
7	EQ	=	Gleichheit, OUT := (IN1=IN2) & (IN2=IN3) & ... & (INn-1 = INn)
8	LE	<=	Monotone Folge, OUT := (IN1<=IN2) & (IN2<=IN3) & ... & (INn-1 <= INn)
9	LT	<	Steigende Folge, OUT := (IN1<IN2) & (IN2<IN3) & ... & (INn-1 < INn)
10	NE	<>	Ungleichheit (nicht erweiterbar) OUT := (IN1 <> IN2)
ANMERKUNG 1: Die Eingänge können vom Typ ANY_BIT, ANY_NUM, STRING, ANY_DATE oder TIME sein. Die Regeln der Typumwandlung, die in 2.5.1.4 angegeben sind, müssen für diese Eingänge benutzt werden. ANMERKUNG 2: Die Angaben IN1, IN2, ..., INn beziehen sich in der Reihenfolge von oben nach unten auf die Eingänge; OUT bezieht sich auf den Ausgang. ANMERKUNG 3: Alle Symbole, die in dieser Tabelle gezeigt werden, sind als Operator für den Gebrauch in Textsprachen geeignet, wie in Tabelle 52 und 55 gezeigt. ANMERKUNG 4: Wenn die Darstellung einer Funktion mit Namensangabe unterstützt wird, muß dies durch den Suffix "n" in der Aussage zur Normerfüllung angegeben werden. Z.B. stellt "5n" die Angabe "GT" dar. ANMERKUNG 5: Wenn die Darstellung einer Funktion als Symbol unterstützt wird, muß dies durch das Suffix "s" in der Aussage zur Normerfüllung angegeben werden. Z.B. stellt "5s" die Angabe ">" dar. ANMERKUNG 6: Der Gebrauch der Beispiele und die Beschreibungen sind in der ST-Sprache angegeben, die in 3.3 definiert ist.			

2.5.1.5.5 Funktionen für Zeichenfolgen

Alle Funktionen, die in 2.5.1.5.4 definiert sind, müssen auf Zeichenfolgen anwendbar sein. Zum Zweck des Vergleiches von zwei Folgen ungleicher Länge muß die kürzere Folge betrachtet werden, als wäre sie rechts auf die Länge der längeren Folge um Zeichen mit dem Wert Null erweitert. Der Vergleich muß von links nach rechts erfolgen, basierend auf den numerischen Werten der Zeichen-Codes der ISO/IEC-646-Codetabelle. Die Zeichenfolge "Z" muß z. B. größer sein als die Zeichenfolge "AZ", und "AZ" muß größer sein als "ABC".

Die standardmäßigen graphischen Darstellungen, Funktionsnamen und Beschreibungen von Funktionen für Zeichenfolgen müssen wie in Tabelle 29 erfolgen. Zum Zweck dieser Operationen müssen die Zeichenpositionen innerhalb der Folge als numeriert mit 1, 2, ..., L betrachtet werden, beginnend mit der linken Zeichenposition, wobei L die Länge der Folge darstellt.

2.5.1.5.6 Funktionen für Datentypen der Zeit

Zusätzlich zu den Funktionen für Vergleich und Auswahl, die in Abschnitt 2.5.1.5.4 definiert sind, müssen diejenigen Kombinationen von Eingang und Ausgang der Datentypen der Zeit, die in Tabelle 30 gezeigt sind, bei den zugehörigen Funktionen erlaubt sein.

2.5.1.5.7 Funktionen für Datentypen der Aufzählung

Die Funktionen für Auswahl und Vergleich, die in Tabelle 31 aufgezählt sind, können mit Eingaben verwendet werden, die einen Datentyp Aufzählung haben, wie in Abschnitt 2.3.3.1 definiert ist.

Tabelle 29: Standardfunktionen für Zeichenfolgen

Nr	Graphische Form	Erläuterung/Beispiel
1	<pre> +-----+ LEN ---INT +-----+ </pre>	Länge einer Folge Beispiel: A := LEN('ASTRING') ; ist gleich A := 7 ;
2	<pre> +-----+ LEFT +-----+ IN ---STRING +-----+ L +-----+ </pre>	L linksstehende Zeichen von IN Beispiel: A := LEFT(IN := 'ASTR', L := 3) ; ist gleich A := 'AST' ;
3	<pre> +-----+ RIGHT +-----+ IN ---STRING +-----+ L +-----+ </pre>	L rechtsstehende Zeichen von IN Beispiel: A := RIGHT(IN := 'ASTR', L := 3) ; ist gleich A := 'STR' ;
4	<pre> +-----+ MID +-----+ IN ---STRING +-----+ L +-----+ P +-----+ </pre>	L Zeichen von IN, beginnend mit dem P-ten Beispiel: A := MID(IN := 'ASTR', L := 2, P := 2) ; ist gleich A := 'ST' ;
5	<pre> +-----+ CONCAT +-----+ IN1 ---STRING +-----+ IN2 +-----+ </pre>	erweiterbare Aneinanderreihung Beispiel: A := CONCAT('AB', 'CD', 'E') ; ist gleich A := 'ABCDE' ;
6	<pre> +-----+ INSERT +-----+ IN1 ---STRING +-----+ IN2 +-----+ P +-----+ </pre>	Fügt IN2 in IN1 nach der P-ten Zeichenposition ein Beispiel: A := INSERT(IN1 := 'ABC', IN2 := 'XY', P := 2) ; ist gleich A := 'ABXYC' ;
7	<pre> +-----+ DELETE +-----+ IN ---STRING +-----+ L +-----+ P +-----+ </pre>	Löscht L Zeichen von IN, beginnend mit der P-ten Zeichenposition Beispiel: A := DELETE(IN := 'ABXYC', L := 2, P := 3) ; ist gleich A := 'ABC' ;
8	<pre> +-----+ REPLACE +-----+ IN1 ---STRING +-----+ IN2 +-----+ L +-----+ P +-----+ </pre>	Ersetzt L Zeichen von IN1 durch IN2, beginnend ab der P-ten Zeichenposition Beispiel: A := REPLACE(IN1 := 'ABCDE', IN2 := 'X', L := 2, P := 3) ; ist gleich A := 'ABXE' ;
9	<pre> +-----+ FIND +-----+ IN1 ---INT +-----+ IN2 +-----+ </pre>	Ermittelt die Zeichenposition des Anfangs für das erste Auftreten von IN2 in IN1. Falls kein Auftreten von IN2 gefunden wird, ist OUT := 0. Beispiel: A := FIND(IN1 := 'ABCB', IN2 := 'BC') ; ist gleich A := 2 ;
ANMERKUNG: Die Beispiele in dieser Tabelle sind in der Sprache Strukturierter Text (ST) angegeben, die in 3.3 definiert ist.		

Tabelle 30: Funktionen für Datentypen der Zeit

Numerische und Aneinanderreihungsfunktionen					
Nr	Name	Symbol	IN1	IN2	OUT
1 2 3	ADD	+	TIME TIME_OF_DAY DATE_AND_TIME	TIME TIME TIME	TIME TIME_OF_DAY DATE_AND_TIME
4 5 6 7 8 9	SUB	-	TIME DATE TIME_OF_DAY TIME_OF_DAY DATE_AND_TIME DATE_AND_TIME	TIME DATE TIME TIME_OF_DAY TIME DATE_AND_TIME	TIME TIME TIME_OF_DAY TIME DATE_AND_TIME TIME
10 11	MUL DIV	* /	TIME TIME	ANY_NUM ANY_NUM	TIME TIME
12	CONCAT		DATE	TIME_OF_DAY	DATE_AND_TIME
Funktionen für Typumwandlung					
13 14	DATE_AND_TIME_TO_TIME_OF_DAY DATE_AND_TIME_TO_DATE				
ANMERKUNG: Die Funktionen zur Typumwandlung müssen die Wirkung des "Herausgreifens" von entsprechenden Daten haben. Z. B. müssen die Anweisungen in ST-Sprache X := DT#1986-04-28-08:40:00 ; Y := DATE_AND_TIME_TO_TIME_OF_DAY(X) ; W := DATE_AND_TIME_TO_DATE(X) dasselbe Ergebnis haben wie die Anweisungen X := DT#1986-04-28-08:40:00 ; W := DATE#1986-04-28 Y := TIME_OF_DAY#08:40:00					

Tabelle 31: Funktionen für Datentypen der Aufzählung

Nr	Name	Symbol	Nummer der Eigenschaft in 2.5.1.5.4
1	SEL		1
2	MUX		4
3	EQ	=	7
4	NE	<>	10

2.5.2 Funktionsbausteine

Für die Anwendungen in den SPS-Sprachen ist ein *Funktionsbaustein* eine Programm-Organisationseinheit, die bei der Ausführung einen oder mehrere Werte liefert. Es können mehrere bezeichnete *Instanzen* ("Fälle", "Kopien") eines Funktionsbausteins erzeugt werden. Jede Instance muß einen zugehörigen Bezeichner (den Instance-Namen) und eine Datenstruktur besitzen, die ihre Ausgangs- und internen Variablen beinhaltet und abhängig von der Implementierung die Werte oder Verweise auf ihre Eingangsparameter. Alle Werte der Ausgangsvariablen und die notwendigen internen Variablen dieser Datenstruktur müssen von einer Bearbeitung des Funktionsbausteins zur nächsten erhalten bleiben; deshalb muß der Aufruf desselben Funktionsbausteins mit denselben Argumenten (Eingangsparametern) nicht immer dieselben Ausgangswerte liefern.

Nur die Eingangs- und Ausgangsparameter dürfen von außerhalb einer Instance eines Funktionsbausteins erreichbar sein, d. h., die internen Variablen des Funktionsbausteins müssen für den Anwender des Funktionsbausteins verborgen sein.

Die Ausführung der Operationen eines Funktionsbausteins muß aufgerufen werden, wie dies in Abschnitt 3 für die Textsprachen definiert ist, nach den Regeln der Netzwerkauswertung, die in Abschnitt 4 für die graphischen Sprachen angegeben sind, oder unter der Kontrolle von Elementen zur Ablaufsprache (AS), wie in 2.6 definiert.

Jeder Funktionsbaustein, der bereits deklariert wurde, kann in der Deklaration eines weiteren Funktionsbausteins oder Programms, wie in Bild 3 gezeigt, benutzt werden.

Der Anwendungsbereich einer Instance eines Funktionsbausteins muß lokal zu der Programm-Organisationseinheit sein, in der sie instanziiert wird, sofern sie nicht als global in einem VAR_GLOBAL-Abschnitt deklariert ist, wie in 2.7.1 definiert ist. Wie in 2.5.2.2 veranschaulicht, kann der Instance-Name einer Funktionsbaustein-Instance als der Eingang für eine Funktion oder einen Funktionsbaustein benutzt werden, wenn diese Instance in einer VAR_INPUT-Deklaration als Eingangsvariable deklariert ist, oder er kann in einer Deklaration VAR_IN_OUT als eine Eingangs/Ausgangsvariable eines Funktionsbausteins benutzt werden, wie es in 2.4.3 definiert ist.

2.5.2.1 Darstellung

Wie in Bild 9 veranschaulicht, kann eine Instance eines Funktionsbausteins in *Textsprache* erzeugt werden, indem man ein Datenelement durch Anwendung eines bereits deklarierten Funktionsbaustein-Typs in der VAR...END_VAR-Konstruktion deklariert. Dies geschieht in der gleichen Weise wie der Gebrauch eines strukturierten Datentyps, wie in 2.4.3 definiert ist. Wie des weiteren in Bild 9 veranschaulicht ist, kann eine Instance eines Funktionsbausteins *graphisch* erzeugt werden, indem eine graphische Darstellung des Funktionsbausteins benutzt wird, mit dem Namen des Funktionsbaustein-Typs innerhalb des Blocks und dem Namen der Instance oberhalb des Blocks. Dabei folgt man den Regeln für die Darstellung von Funktionen, die in 2.5.1.1 angegeben sind, mit den folgenden zusätzlichen Bedingungen:

- 1) Die Größe und die Anordnung des Blocks darf in Abhängigkeit der Anzahl der Eingänge, Ausgänge und weiterer anzuzeigender Informationen variieren.
- 2) Die Namen der formalen Eingangs- und Ausgangsparameter müssen links- bzw. rechtsseitig innerhalb des Blocks angezeigt werden.

Wie in Bild 9 gezeigt, können die Eingangs- und Ausgangsvariablen einer Instance eines Funktionsbausteins als Elemente von strukturierten Datentypen dargestellt werden, wie in 2.3.6.1 definiert ist.

Wenn eine der beiden Eigenschaften zur graphischen Negation, die in Tabelle 19 gezeigt sind, für Funktionsbausteine unterstützt wird, dann muß sie auch für Funktionen unterstützt werden, die in 2.5.1 definiert sind und umgekehrt.

Die Instancen von Funktionsbausteinen können als gepuffert deklariert werden; siehe Eigenschaft 3 in Tabelle 33.

Graphische (FBS-Sprache)	Textform (ST-Sprache)
<pre> FF75 +-----+ SR %IX1--- S1 Q1 ---%QX3 %IX2--- R +-----+ </pre>	<pre> VAR FF75 : SR ; END_VAR (* Deklaration *) FF75(S1 := %IX1, R := %IX2) ; (* Aufruf *) %QX3 := FF75.Q1 ; (* Ausgangszuweisung *) </pre>

Bild 9: Beispiel für Instanziierung eines Funktionsbausteins

Die Zuweisung eines Wertes an eine Ausgangsvariable eines Funktionsbausteins ist unzulässig, auer von innerhalb des Funktionsbausteins. Die Zuweisung eines Wertes an einen Eingang eines Funktionsbausteins ist nur als Teil des Aufrufs des Funktionsbausteins gestattet. Die zulässigen Anwendungen von Eingängen und Ausgängen von Funktionsbausteinen sind in Tabelle 32 zusammengefat; dabei wird der Funktionsbaustein FF75 des Typs SR, der in Bild 9 gezeigt ist, benutzt. Die Beispiele sind in der ST-Sprache gezeigt.

Tabelle 32: Beispiele für den Gebrauch der E/A-Parameter von Funktionsbausteinen

Gebrauch	Innerhalb des Funktionsbausteins	Auerhalb des Funktionsbausteins
Eingang lesen	IF S1 THEN ...	Nicht zulässig (Anmerkungen 1 und 2)
Eingang schreiben	Nicht zulässig (Anmerkungen 1 und 3)	FF75(S1 := %IX1, R := %IX2) ;
Ausgang lesen	Q1 := Q1 AND NOT R ;	%QX3 := FF75.Q1 ;
Ausgang schreiben	Q1 := 1 ;	Nicht zulässig (Anmerkung 1)
ANMERKUNG 1: Die Anwendungen, die als "nicht zulässig" in dieser Tabelle angegeben sind, könnten zu herstellerabhängigen, unvorhersehbaren Seiteneffekten führen. ANMERKUNG 2: Das Lesen eines Eingangs eines Funktionsbausteins kann mittels der "Kommunikationsfunktion", "Bediener-Schnittstelle" oder den "Funktionen für Programmierung, Testen und Überwachen" durchgeführt werden, die in Teil 1 dieser Norm definiert sind. ANMERKUNG 3: Wie in Abschnitt 2.5.2.2 veranschaulicht, ist ein Ändern einer Variablen, die in einem VAR_IN_OUT-Block deklariert ist, innerhalb des Funktionsbausteins erlaubt.		

2.5.2.2 Deklaration

Wie in Bild 10 veranschaulicht, muß ein Funktionsbaustein in Text oder Graphik in derselben Weise deklariert werden, wie dies für Funktionen in 2.5.1.3 definiert ist, mit den Unterschieden, die unten beschrieben und in Tabelle 33 zusammengefaßt sind:

- 1) Die begrenzenden Schlüsselwörter für die Deklaration von Funktionsbausteinen müssen FUNCTION_BLOCK...END_FUNCTION_BLOCK sein.
- 2) Ein Funktionsbaustein kann mehr als einen Ausgangsparameter haben, der in Textform mit dem Konstrukt VAR_OUTPUT...END_VAR deklariert wird, das in 2.4.3 definiert ist, oder graphisch, wie in Bild 10 veranschaulicht ist.
- 3) Das Bestimmungszeichen RETAIN, das in 2.4.3 definiert ist, kann für interne und Ausgangsvariablen eines Funktionsbausteins benutzt werden, wie in den Eigenschaften 1, 2 und 3 in Tabelle 33 gezeigt ist.
- 4) Die Werte von Variablen, die dem Funktionsbaustein mittels eines Konstrukts VAR_IN_OUT oder VAR_EXTERNAL übergeben werden, können innerhalb des Funktionsbausteins verändert werden, wie in Eigenschaft 4 der Tabelle 33 gezeigt ist.
- 5) Auf die Ausgangswerte einer Instance eines Funktionsbausteins, dessen Name in den Funktionsbaustein mittels eines Konstrukts VAR_INPUT, VAR_IN_OUT oder VAR_EXTERNAL übergeben wird, kann zugegriffen werden; sie können aber nicht innerhalb des Funktionsbausteins verändert werden; dies ist in den Eigenschaften 5, 6 und 7 der Tabelle 33 gezeigt.
- 6) Ein Funktionsbaustein, dessen Instance-Name in den Funktionsbaustein mittels einer Konstruktion VAR_IN_OUT oder VAR_EXTERNAL übergeben wird, kann innerhalb des Funktionsbausteins aufgerufen werden; dies ist in den Eigenschaften 6 und 7 der Tabelle 33 gezeigt.
- 7) In Deklarationen in Textform können die Bestimmungszeichen R_EDGE und F_EDGE benutzt werden, um eine Flankenerkennung bei booleschen Eingängen anzuzeigen. Dies muß die implizite Deklaration eines Funktionsbausteins vom Typ R_TRIG bzw. F_TRIG bewirken, wie in 2.5.2.3.2 definiert ist, um die geforderte Flankenerkennung auszuführen. Als ein Beispiel dieser Konstruktion siehe die Eigenschaften 8a und 8b der Tabelle 33 und die zugehörige Anmerkung.
- 8) Die Konstruktion, die in Tabelle 33 mit den Eigenschaften 9a und 9b veranschaulicht ist, muß in graphischen Deklarationen zur Erkennung von steigenden und fallenden Flanken benutzt werden. Wenn der ISO/IEC-646-Zeichensatz benutzt wird, muß das Zeichen "größer als" (>) oder "kleiner als" (<) in der Begrenzungslinie des Funktionsbausteins benutzt werden. Wenn graphische oder semigraphische Darstellungen benutzt werden, muß die Schreibweise der IEC 617 Teil 12 für dynamische Eingänge benutzt werden.
- 9) Die Konstrukte zur Initialisierung von Variablen, die in 2.4.3.2 definiert sind, können zur Deklaration der Voreinstellungswerte von Funktionsbaustein-Eingängen und der Anfangswerte ihrer internen und Ausgangsvariablen benutzt werden.

Wie in Bild 12 veranschaulicht, können nur Variablen und Namen von Instancen der Funktionsbausteine in einen Funktionsbaustein mittels des Konstrukts VAR_IN_OUT übergeben werden; d. h., Ausgänge von Funktionen und Funktionsbausteinen können nicht mittels dieser Konstruktion übergeben werden. Dies soll die versehentlichen Änderungen solcher Ausgänge verhindern. Das "Kaskadieren" von VAR_IN_OUT-Konstruktionen, wie in Bild 12c) gezeigt, ist jedoch zugelassen.

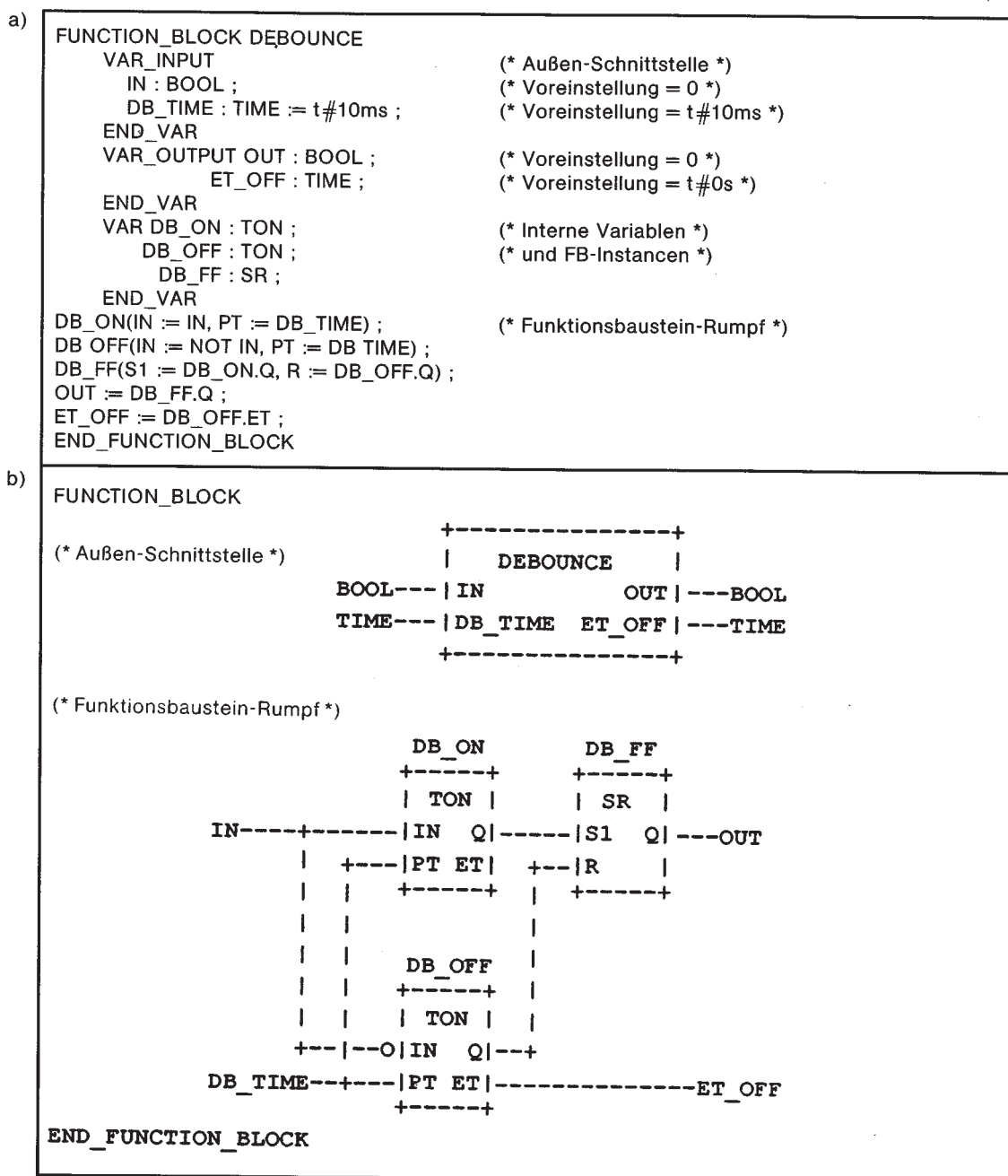


Bild 10: Beispiele für Deklarationen von Funktionsbausteinen

- a) Deklaration in Textform in ST-Sprache (siehe 3.3)
b) Graphische Deklaration in FBS-Sprache (siehe 4.3)

Tabelle 33: Eigenschaften der Funktionsbaustein-Deklarationen

Nr	Beschreibung	Beispiel
1	RETAIN-Bestimmungszeichen für interne Variablen	VAR RETAIN X : REAL ; END_VAR
2	RETAIN-Bestimmungszeichen für Ausgangsvariablen	VAR_OUTPUT RETAIN X : REAL ; END_VAR
3	RETAIN-Bestimmungszeichen für interne Funktionsbausteine	VAR RETAIN TMR1: TON ; END_VAR
4a	Eingangs/Ausgangsdeklaration (Textform)	VAR_INPUT X : INT ; END_VAR VAR_IN_OUT A : INT ; END_VAR A := A+X ;
4b	Eingangs/Ausgangsdeklaration (graphisch)	Siehe Bild 12
5a	Funktionsbaustein-Instance-Name als Eingang (Textform)	VAR_INPUT I_TMR : TON ; END_VAR EXPIRED := I_TMR.Q ; (* Anmerkung 1 *)
5b	Funktionsbaustein-Instance-Name als Eingang (graphisch)	Siehe Bild 11a)
6a	Funktionsbaustein-Instance-Name als Eingang/Ausgang (Textform)	VAR_IN_OUT IO_TMR : TOF ; END_VAR IO_TMR(IN := A_VAR, PT := T#10S) ; EXPIRED := I_TMR.Q ; (* Anmerkung 1 *)
6b	Funktionsbaustein-Instance-Name als Eingang/Ausgang (graphisch)	Siehe Bild 11b)
7a	Funktionsbaustein-Instance-Name als externe Variable (Textform)	VAR_EXTERNAL EX_TMR : TOF ; END_VAR EX_TMR(IN := A_VAR, PT := T#10S) ; EXPIRED := EX_TMR.Q ; (* Anmerkung 1 *)
7b	Funktionsbaustein-Instance-Name als externe Variable (graphisch)	Siehe Bild 11c)
8a	Deklaration von Eingängen (Textform) mit steigender und	FUNCTION_BLOCK AND_EDGE (* Anmerkung 2 *) VAR_INPUT X : BOOL R_EDGE ; Y : BOOL F_EDGE ; END_VAR VAR_OUTPUT Z : BOOL ; END_VAR Z := X AND Y ; (* Beispiel in ST-Sprache *) END_FUNCTION_BLOCK (* siehe 3.3 *)
8b	fallender Flanke in Textform	
9a	Graphische Deklaration von Eingängen mit steigender Flanke und	FUNCTION_BLOCK (* Anmerkung 2 *) +-----+ AND_EDGE +-----+ BOOL--->X +-----+ Z ---BOOL +-----+ +-----+ BOOL---<Y +-----+ +-----+ +-----+ +---+ X--- & ---Z +---+ +---+ Y--- +---+ (* FB-Rumpf *) (* FBS-Sprache *) (* siehe 4.3 *) END_FUNCTION_BLOCK
9b	fallender Flanke	

ANMERKUNG 1: Es wird in diesen Beispielen angenommen, daß die Variablen EXPIRED und A_VAR mit dem Typ BOOL deklariert wurden.

ANMERKUNG 2: Die Deklaration des Funktionsbausteins AND_EDGE in den obigen Beispielen entspricht dem folgenden:

```

FUNCTION_BLOCK AND_EDGE
  VAR_INPUT XCLK : BOOL ; YCLK : BOOL ; END_VAR
  VAR X_TRIG : R_TRIG ; Y_TRIG : F_TRIG ; END_VAR
  X_TRIG(CLK := XCLK) ; X := X_TRIG.Q ;
  Y_TRIG(CLK := YCLK) ; Y := Y_TRIG.Q ;
  Z := X AND Y ;
END_FUNCTION_BLOCK

```

Siehe 2.5.2.3.2 für die Definition der Funktionsbausteine zur Flankenerkennung R_TRIG und F_TRIG.

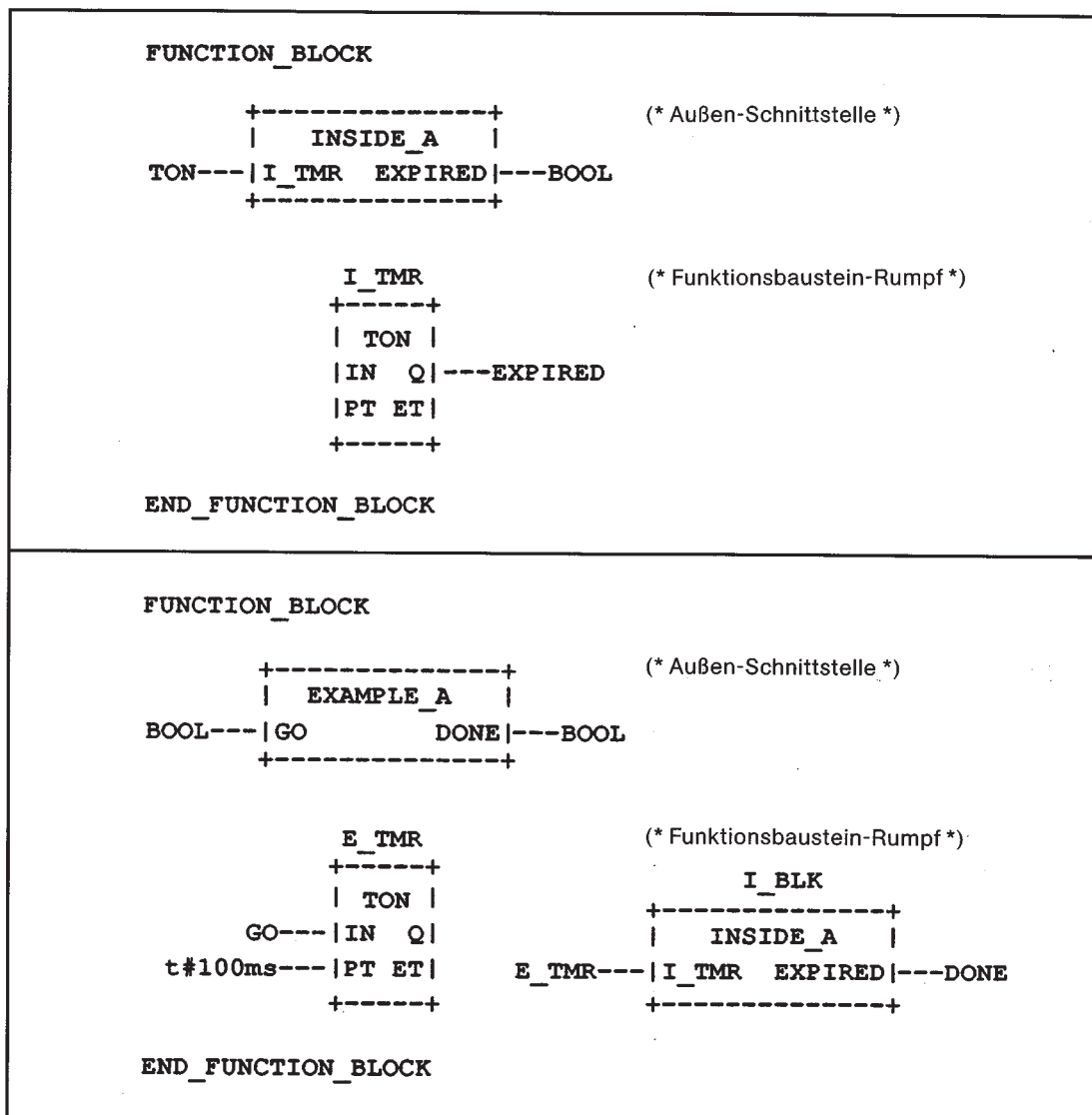


Bild 11a: Graphischer Gebrauch eines Funktionsbaustein-Namens als eine Eingangsvariable
(Tabelle 33, Eigenschaft 5b)