

Speicherprogrammierbare Steuerungen
Teil 3: Programmiersprachen
(IEC 61131-3:2003) Deutsche Fassung EN 61131-3:2003

DIN
EN 61131-3

ICS 25.040.40; 35.060; 35.240.50

Ersatz für
DIN EN 61131-3:1994-08
Siehe Beginn der Gültigkeit

Programmable controllers – Part 3: Programming languages
(IEC 61131-3:2003); German version EN 61131-3:2003

Automates programmables – Partie 3: Langages de programmation
(CEI 61131-3:2003); Version allemande EN 61131-3:2003

Die Europäische Norm EN 61131-3:2003 hat den Status einer Deutschen Norm.

Beginn der Gültigkeit

Die EN 61131-3 wurde am 2002-12-01 angenommen.

Daneben darf DIN EN 61131-3:1994-08 noch bis 2005-12-01 angewendet werden.

Nationales Vorwort

Für die vorliegende Norm ist das nationale Arbeitsgremium K 962 „SPS“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE zuständig.

Norm-Inhalt war veröffentlicht als E DIN IEC 65B/373/CD:1999-03.

Die enthaltene IEC-Publikation wurde vom SC 65B „Devices“ erarbeitet.

Das IEC-Komitee hat entschieden, dass der Inhalt dieser Publikation bis zum Jahr 2007 unverändert bleiben soll. Zu diesem Zeitpunkt wird entsprechend der Entscheidung des Komitees die Publikation

- bestätigt,
- zurückgezogen,
- durch eine Folgeausgabe ersetzt oder
- geändert.

Fortsetzung Seite 2
und 223 Seiten EN

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Änderungen

Gegenüber DIN EN 61131-3:1994-08 wurden folgende Änderungen vorgenommen:

- Doppel-Byte-Strings für die Darstellung von Zeichen in allen Fremdsprachen, auch z. B. Japanisch,
- direkte Adressierung mit noch nicht vollständigen Pfadnamen mittels „Wild-Card“,
- Deklarationen von temporären Variablen,
- Attribute RETAIN und NON_RETAIN für die Kennzeichnung von gepufferten und nicht-gepufferten Variablen,
- formale und nicht-formale Aufrufe bei Funktionen (d. h. mit und ohne Parameter-Namen auch für Funktionsargumente, wie bei Funktionsbausteinen),
- BCD-Wandlungsfunktionen, Funktionen für die Zeit-Datentypen,
- erweiterte Initialisierungsmöglichkeiten für Funktionsbausteine, Initialisierung von Konfiguration,
- außerdem gibt es einige Erweiterungen für die Anweisungsliste und bei Sprachelementen der Ablaufsprache,
- die erweiterten Spracheigenschaften führten neben den Änderungen der Semantik zu entsprechenden Änderungen in der formalen Syntax im Anhang B der Norm.

Frühere Ausgaben

DIN EN 61131-3:1994-08

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf eine Norm ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils neueste gültige Ausgabe der in Bezug genommenen Norm.

Für den Fall einer datierten Verweisung im normativen Text bezieht sich die Verweisung immer auf die in Bezug genommene Ausgabe der Norm.

Der Zusammenhang der zitierten Normen mit den entsprechenden Deutschen Normen ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation. Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

IEC hat 1997 die Benummerung der IEC-Publikationen geändert. Zu den bisher verwendeten Normnummern wird jeweils 60000 addiert. So ist zum Beispiel aus IEC 68 nun IEC 60068 geworden.

Seit der Veröffentlichung der 1. Ausgabe der IEC 61131-3 im Jahre 1993 hat sich das Umfeld der Norm stark verändert. Während dieser Zeit wurde die Norm in großem Umfang weltweit angewendet, und dabei wurden viele Erfahrungen mit der praktischen Anwendung der Norm gemacht. Eine Anzahl von Ungereimtheiten, Widersprüchen und ungelösten Fragen wurde dabei gefunden. Außerdem zeigten sich Anforderungen, die unnötig schwer zu realisieren sind. Die Nutzer der Norm, z. B. SPS-Hersteller und -Anwender, sowie Software-Häuser, die SPS-Programmiertool entwickeln, lieferten viele Änderungs- und Verbesserungsvorschläge.

Um die von IEC 61131-3 Anwendern geleisteten Investitionen zu erhalten und den Nutzen der existierenden norm-konformen Steuerungssoftware sogar zukünftig auszudehnen, wurde die nun vorliegende 2. Ausgabe erstellt. Sämtliche Änderungen und Erweiterungen sind strikt aufwärts-kompatibel, d. h. ein Anwenderprogramm, das norm-konform zur 1. Ausgabe ist, muss dies auch bezüglich 2. Ausgabe bleiben. Die einzige Ausnahme ist der Gebrauch der eckigen statt der runden Klammern bei den Datenfeldern und einiger Namensänderungen bei den Zeitdaten-Funktionen.

Deutsche Fassung

Speicherprogrammierbare Steuerungen

Teil 3: Programmiersprachen
(IEC 61131-3:2003)

Programmable controllers –
Part 3: Programming languages
(IEC 61131-3:2003)

Automates programmables –
Partie 3: Langages de programmation
(CEI 61131-3:2003)

Diese Europäische Norm wurde von CENELEC am 2002-12-01 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, Malta, den Niederlanden, Norwegen, Österreich, Portugal, Schweden, der Schweiz, der Slowakei, Spanien, der Tschechischen Republik, Ungarn 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/456/FDIS, zukünftige 2. Ausgabe von IEC 61131-3, ausgearbeitet von dem SC 65B „Devices“ des IEC TC 65 „Industrial-process measurement and control“, wurde der IEC-CENELEC Parallelen Abstimmung unterworfen und von CENELEC am 2002-12-01 als EN 61131-3 angenommen.

Diese Europäische Norm ersetzt EN 61131-3:1993.

Nachstehende Daten wurden festgelegt:

- spätestes Datum, zu dem die EN auf nationaler Ebene durch Veröffentlichung einer identischen nationalen Norm oder durch Anerkennung übernommen werden muss (dop): 2003-10-01
- spätestes Datum, zu dem nationale Normen, die der EN entgegenstehen, zurückgezogen werden müssen (dow): 2005-12-01

Anhänge, die als „normativ“ bezeichnet sind, gehören zum Norminhalt.

Anhänge, die als „informativ“ bezeichnet sind, enthalten nur Informationen.

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

Der Anhang ZA wurde von CENELEC hinzugefügt.

Anerkennungsnotiz

Der Text der Internationalen Norm IEC 61131-3:2003 wurde von CENELEC ohne irgendeine Abänderung als Europäische Norm angenommen.

Inhalt

	Seite
1 Allgemeines	9
1.1 Anwendungsbereich	9
1.2 Normative Verweisungen.....	9
1.3 Begriffe	9
1.4 Übersicht und allgemeine Anforderungen	16
1.5 Normerfüllung	22
2 Gemeinsame Elemente	24
2.1 Gebrauch der gedruckten Zeichen	24
2.2 Externe Darstellung von Daten.....	26
2.3 Datentypen	30
2.4 Variablen.....	36
2.5 Programm-Organisationseinheiten.....	45
2.6 Elemente der Ablaufsprache (AS)	83
2.7 Konfigurationselemente	106
3 Textsprachen	120
3.1 Gemeinsame Elemente	120
3.2 Anweisungsliste (AWL).....	121
3.3 Sprache ST (Strukturierter Text)	126
4 Grafische Sprachen.....	131
4.1 Gemeinsame Elemente	131
4.2 Sprache Kontaktplan (KOP)	135
4.3 Funktionsbaustein-Sprache (FBS)	139
Anhang A (normativ) Spezifikationsmethode für Textsprachen	141
A.1 Syntax	141
A.1.1 Terminale Symbole.....	141
A.1.2 Nicht-terminale Symbole.....	141
A.1.3 Produktionsregeln	142
A.2 Semantische Regeln.....	142
Anhang B (normativ) Formale Festlegungen der Sprachelemente	143
B.0 Programmiermodell	143
B.1 Gemeinsame Elemente	143
B.1.1 Buchstaben, Ziffern und Bezeichner	143
B.1.2 Konstanten.....	144
B.1.3 Datentypen	146
B.1.4 Variablen.....	148
B.1.5 Programm-Organisationseinheiten.....	151
B.1.6 Elemente der Ablaufsprache	152
B.1.7 Konfigurationselemente	153

	Seite
B.2 Sprache AWL (Anweisungsliste)	155
B.2.1 Anweisungen und Operationen.....	155
B.2.2 Operatoren	155
B.3 Sprache ST (Strukturierter Text).....	156
B.3.1 Ausdrücke	156
B.3.2 Anweisungen.....	156
Anhang C (normativ) Begrenzungszeichen und Schlüsselwörter.....	158
Anhang D (normativ) Implementierungsabhängige Parameter	162
Anhang E (normativ) Fehlerursachen	164
Anhang F (informativ) Beispiele	166
F.1 Funktion WIEGEN.....	166
F.2 Funktionsbaustein KOMMANDO_ÜBERWACHUNG	167
F.3 Funktionsbaustein VORW_RÜCKW_ÜBERWACHUNG.....	170
F.4 Funktionsbaustein STACK_INT	175
F.5 Funktionsbaustein MIX_2_ZIEGEL	179
F.6 Analogsignal-Verarbeitung.....	182
F.6.1 Funktionsbaustein FILT1	183
F.6.2 Funktionsbaustein TOTZEIT	183
F.6.3 Funktionsbaustein MITTEL	184
F.6.4 Funktionsbaustein INTEGRAL	184
F.6.5 Funktionsbaustein GRADIENT	185
F.6.6 Funktionsbaustein HYSTERESE	185
F.6.7 Funktionsbaustein GRENZ_ALARM.....	186
F.6.8 Struktur ANALOG_GRENZEN.....	186
F.6.9 Funktionsbaustein ANALOG_ÜBERWACHUNG	187
F.6.10 Funktionsbaustein PID	188
F.6.11 Funktionsbaustein DIFFGLEICH	189
F.6.12 Funktionsbaustein RAMPE	190
F.6.13 Funktionsbaustein TRANSFER	191
F.7 Programm SCHÜTTGUT	191
F.8 Programm AGV	199
F.9 Gebrauch von aufgezählten Datentypen	202
F.10 Funktionsbaustein Echtzeituhr (RTC).....	203
F.11 Funktionsbaustein ALRM_INT	203
Anhang G (informativ) Zeichensatz.....	204
Stichwortverzeichnis	207
Anhang ZA (normativ) Normative Verweisungen auf internationale Publikationen mit ihren entsprechenden europäischen Publikationen.....	223
Bilder	
Bild 1 – Software-Modell	17

	Seite
Bild 2 a) – Datenflussverbindung innerhalb eines Programms.....	18
Bild 2 b) – Kommunikation über globale Variable.....	19
Bild 2 c) – Kommunikationsfunktionsbausteine	19
Bild 2 d) – Kommunikation über Zugriffspfade.....	19
Bild 3 – Kombination der SPS-Sprachelemente	21
Bild 4 – Beispiele für den Gebrauch von Funktionen	46
Bild 5 – Gebrauch von formalen Argumentnamen	48
Bild 6 – Beispiel für Deklarationen und Gebrauch der Funktion.....	52
Bild 7 – Beispiele für explizite Typumwandlung bei überladenen Funktionen.....	54
Bild 8 – Beispiele von expliziter Typumwandlung bei Funktionen mit Typangabe	54
Bild 9 – Beispiele für Instanziierung eines Funktionsbausteins.....	67
Bild 10 – Beispiele für Deklarationen von Funktionsbausteinen.....	70
Bild 11 a) – Grafischer Gebrauch eines Funktionsbaustein-Namen als eine Eingangsvariable (Tabelle 33, Eigenschaft 5b).....	73
Bild 11 b) – Grafischer Gebrauch eines Funktionsbaustein-Namen als eine Eingangs/Ausgangsvariable (Tabelle 33, Eigenschaft 6b).....	74
Bild 11 c) – Grafischer Gebrauch eines Funktionsbaustein-Namens als eine externe Variable (Tabelle 33, Eigenschaft 7b).....	75
Bild 12 – Beispiele für den Gebrauch von Eingangs/Ausgangsvariablen in Funktionsbausteinen a) Deklarationen in grafischer und Textform b), c), d) Zulässiger Gebrauch; e) Unzulässiger Gebrauch	76
Bild 14 – Funktionsbaustein ACTION_CONTROL – Externe Schnittstelle (Unsichtbar für den Anwender) a) Mit „letztem Durchlauf“ – siehe Bild 15 a); b) Ohne „letztem Durchlauf“ – siehe Bild 15 b).....	94
Bild 15 a) – ACTION_CONTROL Funktionsbaustein-Rumpf mit „letztem Durchlauf“.....	95
Bild 15 b) – ACTION_CONTROL Funktionsbaustein-Rumpf ohne „letzten Durchlauf“	96
Bild 16 a) – Beispiel für Aktionssteuerung – AS Darstellung.....	97
Bild 16 b) – Beispiel für Aktionssteuerung – Funktional gleichartig.....	98
Bild 17 – AS-Ablaufregeln.....	103
Bild 18 a) – Beispiele für Fehler in Anlaufsprache: Eine „unsichere“ Ablaufkette (siehe 2.6.5)	104
Bild 18 b) – Beispiele für Fehler Ablaufsprache: Eine „unerreichbare“ Ablaufkette (siehe 2.6.5)	105
Bild 19 a) – Grafisches Beispiel für eine Konfiguration	107
Bild 19 b) – Gerüst der Deklarationen von Funktionsbausteinen und Programmen als Beispiel für eine Konfiguration.....	108
Bild 20 – Beispiele von Deklaration der Eigenschaften von CONFIGURATION und RESOURCE	112
Bild 21 a) – Synchronisation von Funktionsbausteinen bei expliziten Task-Zuordnungen	118
Bild 21 b) – Synchronisation von Funktionsbausteinen bei impliziten Task-Zuordnungen	119
Bild 21 c) – Explizite Task-Zuordnungen äquivalent mit Bild 21 b)	120
Bild 22 – Beispiel für die Anweisung EXIT	130
Bild 23 – Beispiel eines Rückkopplungspfads a) Explizite Schleife b) Implizite Schleife c) KOP Sprache-Äquivalent	134
Bild 24 – Beispiele für boolesches ODER a) „Verdrahtetes ODER“ in KOP b) Funktion in FBS.....	139

	Seite
Bild F.1 – Funktionsbaustein MIX_2_ZIEGEL – Physikalisches Modell	179
Bild F.2 – Schüttgut Mess- und Verladsystem	193
Bild F.3 – Deklarationen für das Programm SCHÜTTGUT	194
Bild F.4 – AS für den Programm-Rumpf von SCHÜTTGUT	195
Bild F.5 – Programm-Rumpf von SCHÜTTGUT (Fortsetzung) – Ablauf und Überwachung des Steuerungszustands	196
Bild F.6 – Aktionsrumpf von MONITOR_ACTION in FBS-Sprache	197
Bild F.7 – Rumpf des Programms SCHÜTTGUT in Textdarstellung mit AS unter Verwendung von ST Sprachelementen	198
Bild F.8 – Beispiel-Konfiguration für das Programm SCHÜTTGUT	199
Bild F.9 – Physikalisches Modell für das Programm AGV	200
Bild F.10 – Grafische Deklaration des Programms AGV	200
Bild F.11 – Eine grafische Konfiguration des Programms AGV	200
Bild F.12 – Programm-Rumpf von AGV	201
Bild F.12 – Programm-Rumpf von AGV (<i>Fortsetzung</i>)	202
 Tabellen	
Tabelle 1 – Zeichensatz	24
Tabelle 2 – Bezeichner-Eigenschaften	25
Tabelle 3 – Kommentar- Eigenschaft	26
Tabelle 3a – Pragma-Eigenschaften	26
Tabelle 4 – Numerische Literale	27
Tabelle 5 - Zeichenfolge Literal	28
Tabelle 6 – Zwei-Zeichen-Kombinationen in Zeichenfolgen	29
Tabelle 7 – Zeitdauer Literal	30
Tabelle 8 – Literale für Datum und Tageszeit	30
Tabelle 9 – Beispiele für Datum und Tageszeit	30
Tabelle 10 – Elementare Datentypen	31
Tabelle 11 – Hierarchie der allgemeinen Datentypen	32
Tabelle 12 – Eigenschaften der Datentyp-Deklarationen	34
Tabelle 13 – Voreingestellte Anfangswerte	34
Tabelle 14 – Eigenschaften der Deklaration der Anfangswerte von Datentypen	35
Tabelle 15 – Eigenschaften der Präfixe für Speicherort und Größe bei direkt dargestellten Variablen	37
Tabelle 16 a) – Schlüsselwörter für Variablen-Deklaration	39
Tabelle 16 b) – Gebrauch von VAR_GLOBAL, VAR_EXTERNAL und CONSTANT Deklarationen	40
Tabelle 17 – Eigenschaften der Typzuweisung für Variablen	41
Tabelle 18 – Zuweisung von Anfangswerten für Variablen	43
Tabelle 19 – Grafische Negation von booleschen Signalen	48
Tabelle 19 a) – Aufruf in Textform von Funktionen mit formale und nicht-formale Argumentenliste	49
Tabelle 20 – Der Gebrauch des EN Eingangs und ENO Ausgangs	50
Tabelle 20 a) – Eigenschaften der Funktion	51

	Seite
Tabelle 21 – Typangabe und überladene Funktionen	53
Tabelle 22 – Funktion zur Typumwandlung	55
Tabelle 23 – Standardfunktionen mit einer numerischen Variablen	57
Tabelle 24 – Standardmäßige arithmetische Funktionen	57
Tabelle 25 – Standardmäßige Bitschiebe-Funktionen	59
Tabelle 26 – Bitweise boolesche Standardfunktionen	59
Tabelle 27 – Standardfunktionen für Auswahl ^d	60
Tabelle 28 – Standardfunktionen für Vergleich	62
Tabelle 29 – Standardfunktionen für Zeichenfolgen	63
Tabelle 30 – Funktionen für Datentypen der Zeit	64
Tabelle 31 – Funktionen für Datentypen der Aufzählung	66
Tabelle 32 – Beispiele für den Gebrauch der E/A-Variablen von Funktionsbausteinen	67
Tabelle 33 – Funktionsbaustein Deklaration und Verwendung	71
Tabelle 34 – Bistabile Funktionsbausteine ^a	77
Tabelle 35 – Standard-Funktionsbausteine Flankenerkennung	78
Tabelle 36 – Standard-Funktionsbausteine Zähler	78
Tabelle 37 – Standard Funktionsbaustein Zeitgeber	81
Tabelle 38 – Standard-Funktionsbausteine Zeitgeber – Zeitdiagramme	81
Tabelle 39 – Eigenschaften der Programm-Deklaration	83
Tabelle 40 – Eigenschaften des Schritts	85
Tabelle 41 – Transitionen und Transitionsbedingungen	87
Tabelle 42 – Deklaration von Aktionen ^{a, b}	89
Tabelle 43 – Verknüpfung Schritt/Aktion	91
Tabelle 44 – Eigenschaften des Aktionsblock	92
Tabelle 45 – Aktionsbestimmungszeichen	93
Tabelle 45 a) – Eigenschaften der Aktionssteuerung	97
Tabelle 46 – Kettenablauf	99
Tabelle 47 – Kompatibilität der Eigenschaften der Ablaufkette	105
Tabelle 48 – Mindestanforderungen der Normerfüllung	106
Tabelle 49 – Deklaration von Konfigurations- und Ressource-Eigenschaften	110
Tabelle 50 – Task-Eigenschaften	114
Tabelle 51 a) – Beispiele für Anweisungsfelder	121
Tabelle 51 b) – Eigenschaften des geklammerter Ausdrucks in Anweisungsliste	122
Tabelle 52 – Operatoren der Anweisungsliste (AWL)	122
Tabelle 53 – Eigenschaften des Funktionsbaustein-Aufrufs und Funktionsaufrufs in der Sprache AWL 124	
Tabelle 54 - Eingangsoperatoren von Standard-Funktionsbausteinen für die Sprache AWL	126
Tabelle 55 – Operatoren der Sprache ST	127
Tabelle 56 – Anweisungen der Sprache ST	128
Tabelle 57 – Darstellung von Linien und Blöcken	132

	Seite
Tabelle 58 – Beispiele von grafischen Elementen zur Ausführungssteuerung	135
Tabelle 59 – Stromschienen	136
Tabelle 60 – Verbindungselemente	137
Tabelle 61 – Kontakte ^a	138
Tabelle 62 – Spulen (Abschlussoperationen)	138
Tabelle C.1 – Begrenzungszeichen	158
Tabelle C.2 – Schlüsselwörter	160
Tabelle D.1 – Implementierungsabhängige Parameter	162
Tabelle E.1 – Fehlerursachen	164
Tabelle G.1 – Zeichen Darstellung.....	204
Tabelle G.2 – Zeichen Codierung	205

1 Allgemeines

1.1 Anwendungsbereich

Dieser Teil der IEC 61131 legt die Syntax und Semantik von Programmiersprachen für *Speicherprogrammierbare Steuerungen* fest, wie sie in Teil 1 der IEC 61131 definiert sind.

Die Funktionen der Programmeingabe, des Tests, der Beobachtung, des Betriebssystems usw. sind in Teil 1 der IEC 61131 festgelegt.

1.2 Normative Verweisungen

Die folgenden zitierten Dokumente sind für die Anwendung dieses Dokuments erforderlich. Bei datierten Verweisungen gilt nur die in Bezug genommene Ausgabe. Bei undatierten Verweisungen gilt die letzte Ausgabe des in Bezug genommenen Dokuments (einschließlich aller Änderungen).

IEC 60050 (alle Teile), *International Electrotechnical Vocabulary (IEV)*.

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*.

IEC 60617-12:1991, *Grafical symbols for diagrams – Part 12: Binary logic elements*.

IEC 60617-13:1978, *Grafical symbols for diagrams – Part 13: Analogue elements*.

IEC 60848: 2002, *GRAFCET Specification language for sequential function charts*.

IEC 61131-1, *Programmable controllers – Part 1: General information*.

IEC 61131-5, *Programmable controllers – Part 5: Communications*.

ISO/AFNOR:1989, *Dictionary of Computer Science – The standardised vocabulary*.

ISO/IEC 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*.

1.3 Begriffe

Für die Anwendung dieses Teil der IEC 61131 sind die folgenden Definitionen anzuwenden. Definitionen, die für alle Teile der IEC 61131 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 die IEC 60050 sollten für Begriffe zu Rate gezogen werden, die nicht in dieser Norm definiert sind.

1.3.1

absolute Zeit (absolute time)

Kombination von Tageszeit und Datum

1.3.2

Zugriffspfad (access path)

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

1.3.3

Aktion (action)

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)

grafisches Sprachelement, das eine boolesche Eingangsvariable benutzt, um den Wert einer booleschen Ausgangsvariablen oder die Freigabebedingung für eine *Aktion* zu bestimmen; dies erfolgt nach 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 *Eingangsvariable*, *Ausgangsvariable* oder *Ein-Aus-Variable*

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 (ISO)

1.3.9

basisbezogene Zahl (based number)

Zahl, die mit 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)

Datenelement, bestehend aus einem oder mehr Bits

1.3.12

Rumpf (body)

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)

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

1.3.14

Zeichenfolge (character string)

Aggregat, das aus einer geordneten Folge von Zeichen besteht

1.3.15

Kommentar (comment)

Sprachkonstrukt 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 der Maschinensprache oder einer Zwischenform

1.3.17**Konfiguration (configuration)**

Sprachelement, das einem *SPS-System* entspricht, wie es in IEC 61131-1 definiert ist

1.3.18**Zähler-Baustein (counter function block)**

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

1.3.19**Datentyp (data type)**

Menge von Werten zusammen mit der Menge von zulässigen Operationen (ISO)

1.3.20**Datum und Uhrzeit (date and time)**

Datum im Jahr und die Tageszeit, dargestellt als ein einzelnes Sprachelement

1.3.21**Deklaration (declaration)**

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

1.3.22**Begrenzungszeichen (delimiter)**

Zeichen oder Zeichenkombination, zur Trennung von *Sprachelementen*

1.3.23**direkte Darstellung (direct representation)**

Mittel zur Darstellung einer Variablen in einem SPS-Programm, von der eine hersteller-spezifische 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)**

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 und möglicherweise zusätzliche *Ausgangsvariable* (die Mehrfachwerte sein dürfen, z. B. ein *Feld* oder eine *Struktur*) und deren *Aufruf* in Textsprachen als ein *Operand* in einem Ausdruck benutzt werden kann

1.3.29

Funktionsbaustein-Instanz (Funktionsbaustein) (function block instance, function block)

Instanz eines Funktionsbaustein-Typs

1.3.30

Funktionsbaustein-Typ (function block type)

SPS-*Sprachelement*, bestehend aus: (1) der Definition einer Datenstruktur, geteilt in Eingangs-, Ausgangs- und interne Variablen und (2) einer Menge von Operationen, die mit den Elementen der Datenstruktur durchgeführt werden, wenn eine *Instanz* des *Funktionsbaustein-Typs* aufgerufen wird

1.3.31

Funktionsbaustein-Sprache (function block diagram)

Netzwerk, in dem die Knoten *Funktionsbaustein-Instanzen*, grafisch dargestellte *Funktionen (Prozeduren)*, *Variablen*, *Literale* und *Marken* sind

1.3.32

allgemeiner Datentyp (generic data type)

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

1.3.33

globaler Geltungsbereich (global scope)

Geltungsbereich einer Deklaration, anwendbar 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 ein 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

Ein-Aus-Variable (in-out variable)

Variable, die in einem VAR_IN_OUT ... END_VAR Block deklariert ist

1.3.38

Anfangswert (initial value)

Wert, der einer *Variablen* beim Systemanlauf zugewiesen wird

1.3.39

Eingangsparameter (Eingang) (input parameter, input)

Variable, die benutzt wird, um einer *Programm-Organisationseinheit* ein Argument zu übergeben

1.3.40

Instanz (Instanz)

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

1.3.41**Instanz-Name (Instanz name)**

Bezeichner, verknüpft mit einer festgelegten *Instanz*

1.3.42**Instanziierung (instantiation)**

Erzeugung einer *Instanz*

1.3.43**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

1.3.44**Aufruf (invocation)**

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

1.3.45**Schlüsselwort (keyword)**

lexikalische Einheit, die ein *Sprachelement* charakterisiert, z. B. „IF“

1.3.46**Marke (label)**

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

1.3.47**Sprachelement (language element)**

Element, das durch ein Symbol auf der linken Seite der Produktionsregeln in der formalen Spezifikation identifiziert wird, die in Anhang B dieser Norm angegeben ist

1.3.48**Literal (literal)**

lexikalische Einheit, die direkt einen Wert darstellt (ISO)

1.3.49**lokaler Geltungsbereich (local scope)**

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

1.3.50**logischer Speicherort (logical location)**

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

1.3.51**lange Realzahl (long real)**

Realzahl, die in einem *Langwort* dargestellt ist

1.3.52**Langwort (long word)**

64-Bit-Datenelement

1.3.53**Speicher (Anwenderdatenspeicher) (memory, user data storage)**

funktionale Einheit, in die das Anwenderprogramm Daten speichern kann und aus der es die gespeicherten Daten zurückgewinnen kann

1.3.54

bezeichnetes Element (named element)

Element einer *Struktur*, das durch seinen zugehörigen *Bezeichner* benannt ist

1.3.55

Netzwerk (network)

Anordnung von Knoten und miteinander verbundenen Zweigen

1.3.56

Ausschaltverzögerung (Einschaltverzögerung) (off-delay, on-delay timer function block)

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

1.3.57

Operand (operand)

Sprachelement, mit dem eine Operation durchgeführt wird

1.3.58

Operator (operator)

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

1.3.59

Ausgangsvariable (Ausgang) (output variable, output)

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

1.3.60

Überladen (overloaded)

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

1.3.61

Stromfluss (power flow)

symbolischer Fluss des elektrischen Stroms in einem Kontaktplan, benutzt, um das fortschreitende logische Berechnen eines Algorithmus zu beschreiben

1.3.62

Pragma (pragma)

Sprachkonstrukt zur Einbindung von Text in eine *Programm-Organisationseinheit*, das die Vorbereitung des Programms zur Ausführung beeinflussen darf

1.3.63

Programmieren (program)

Entwerfen, schreiben und testen von Anwenderprogrammen

1.3.64

Programm-Organisationseinheit (program organization unit)

Funktion, *Funktionsbaustein* oder Programm

ANMERKUNG Dieser Begriff darf sich sowohl auf einen Typ als auch auf eine Instanz beziehen.

1.3.65

Real-Literal (real literal)

Literal, das Daten vom Typ `REAL` oder `LREAL` darstellt

1.3.66

Ressource (resource)

Sprachelement, das einer „Signalverarbeitungsfunktion“ und ihrer „Mensch-Maschine-Schnittstelle“ und den „Funktionen der Sensor- und Aktor-Schnittstelle“ entspricht, sofern sie vorhanden sind, wie es in IEC 61131-1 definiert ist

1.3.67**gepufferte Daten (retentive data)**

Daten, die so gespeichert sind, dass ihr Wert nach einer Folge von Spannungsausfall/Wiederkehr unverändert erhalten bleibt

1.3.68**Rücksprung (return)**

Sprachkonstruktion innerhalb einer *Programm-Organisationseinheit*, die ein Ende der Ausführungsfolgen in der Einheit bezeichnet

1.3.69**steigende Flanke (rising edge)**

Wechsel einer booleschen Variablen von 0 nach 1

1.3.70**Geltungsbereich (scope)**

Teil eines *Sprachelements*, in dem eine *Deklaration* oder eine *Marke* gültig ist

1.3.71**Semantik (semantics)**

Beziehungen zwischen den symbolischen Elementen einer Programmiersprache und ihrer Bedeutung, Auslegung und Anwendung

1.3.72**semigrafische Darstellung (semigrafic representation)**

Darstellung der grafischen Information durch die Anwendung eines eingeschränkten Zeichensatzes

1.3.73**Einzel-Datenelement (single data element)**

Datenelement, das aus einem einzelnen Wert besteht

1.3.74**Einzelelement-Variable (single element variable)**

Variable, die ein *Einzel-Datenelement* darstellt

1.3.75**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

1.3.76**strukturierter Datentyp (structured data type)**

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

1.3.77**Indizierung (subscribing)**

Mechanismus zur Adressierung eines *Datenfeldes* durch einen Verweis auf ein Feld und einen oder mehrere Ausdrücke, die bei *Auswertung* die Position des Elementes bezeichnen

1.3.78**symbolische Darstellung (symbolic representation)**

Anwendung von *Bezeichnen*, um *Variablen* mit Namen zu versehen

1.3.79**Task (task)**

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

1.3.80

Zeitliteral (time literal)

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

1.3.81

Transition (transition)

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

1.3.82

vorzeichenlose ganze Zahl (unsigned integer)

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

1.3.83

verdrahtetes ODER (wired OR)

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

1.4 Übersicht und allgemeine Anforderungen

Dieser Teil der IEC 61131 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 grafische 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.

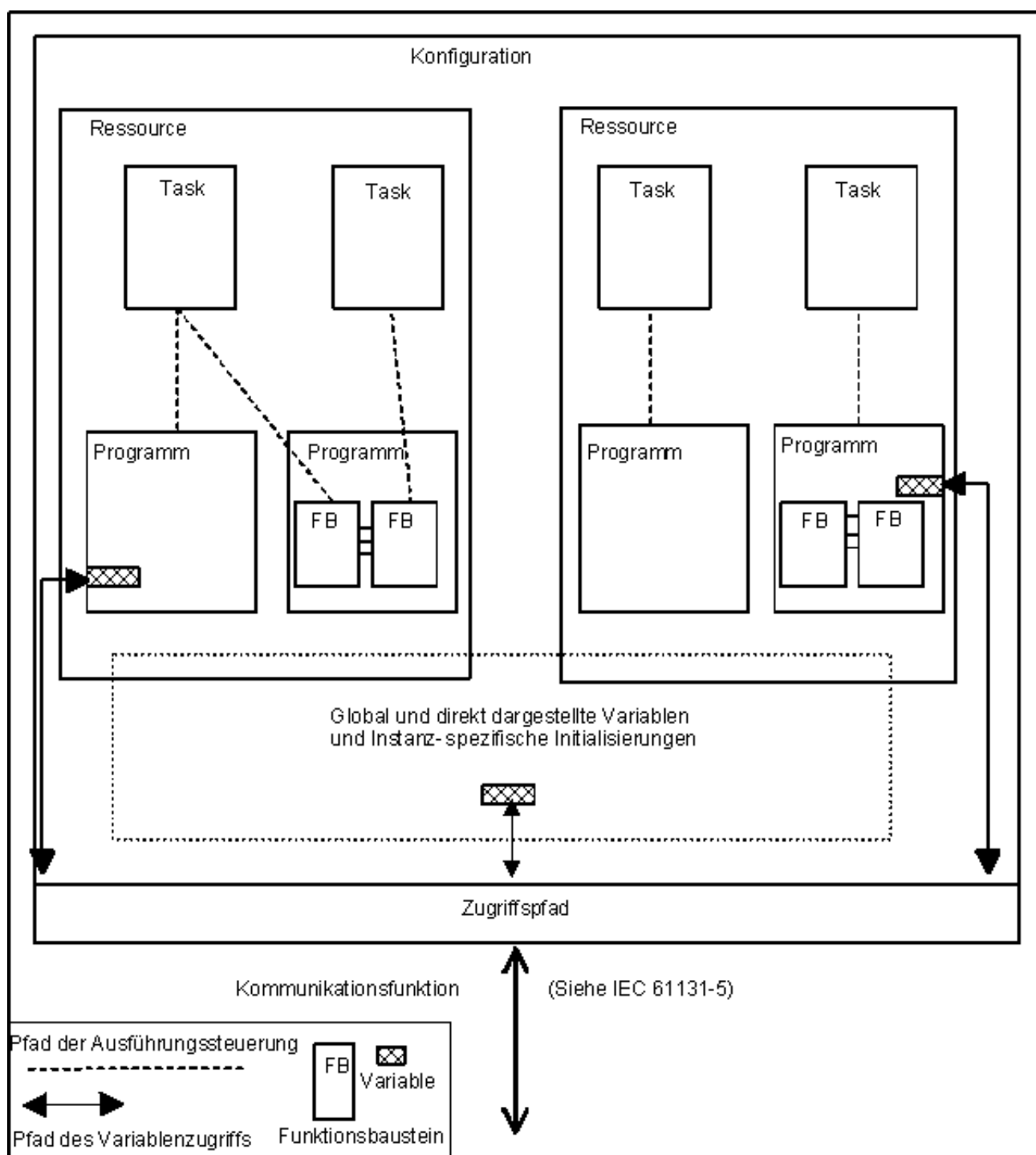
Zusätzlich sind Mittel definiert, die die Kommunikation zwischen Speicherprogrammierbaren Steuerungen und anderen Komponenten von automatisierten Systemen ermöglichen.

Die Elemente der Programmiersprachen in diesem Teil dürfen in einer interaktiven Programmierumgebung angewendet werden. Die Festlegung derartiger Umgebungen gehört nicht zum Geltungsbereich dieser Norm; eine solche Umgebung muss jedoch eine Programmdokumentation in Text- oder Grafik-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 dieser Norm definiert sind, d. h. *Programme* und *Funktionsbausteine*, des Weiteren *Konfigurationselemente* wie *Konfigurationen*, *Ressourcen*, *Tasks*, *globale Variablen* und *Zugriffspfade* und Instanz-spezifische Initialisierungen, die zur Installation von *SPS-Programmen* in den SPS-Systemen dienen.



ANMERKUNG 1 Dieses Bild dient nur zur Veranschaulichung. Die grafische 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 ein Sprachelement, das einem *SPS-System* entspricht, wie es in IEC 61131-1 definiert ist. Eine *Ressource* entspricht einer „Signalverarbeitungsfunktion“ und ihrer „Mensch-Maschine-Schnittstelle“ und sofern vorhanden den Funktionen der „Sensor- und Aktor-Schnittstelle“, wie in IEC 61131-1 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 es 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 es in IEC 61131-1 definiert ist. Das Starten einer *Konfiguration* muss die Initialisierung ihrer *globalen Variablen* bewirken, nach 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 61131-5 definiert sind.

Programme, *Ressourcen*, *globale Variablen*, *Zugriffspfade* (und ihre zugehörigen Zugriffsberechtigungen) und *Konfigurationen* können durch die „Kommunikationsfunktion“ geladen und gelöscht werden, die in IEC 61131-1 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 in diesem Abschnitt definierten Sprachelemente auf die Kommunikationsobjekte ist in IEC 61131-5 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 2 a) 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 grafischen 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 2 b) veranschaulicht. Diese Variablen müssen als `GLOBAL` in der Konfiguration und als `EXTERNAL` im Programm deklariert werden, wie es in 2.4.3 festgelegt ist.

Wie in Bild 2 c) 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 Kommunikationsbausteine verwendet werden, die in IEC 61131-5 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 2 d) veranschaulicht ist, indem die Mechanismen benutzt werden, die in IEC 61131-5 definiert sind.

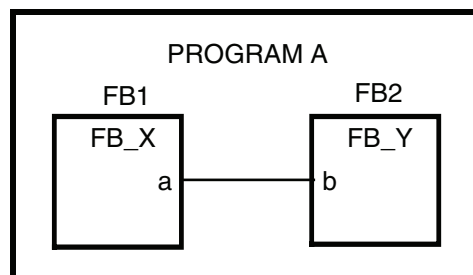


Bild 2 a) – Datenflussverbindung innerhalb eines Programms

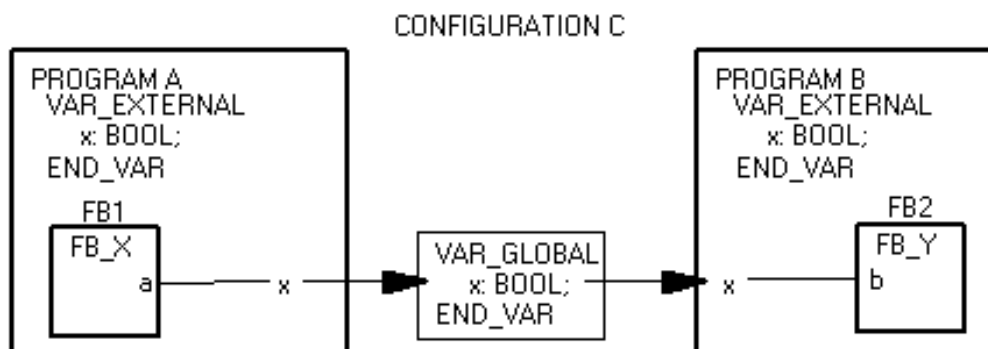


Bild 2 b) – Kommunikation über globale Variable

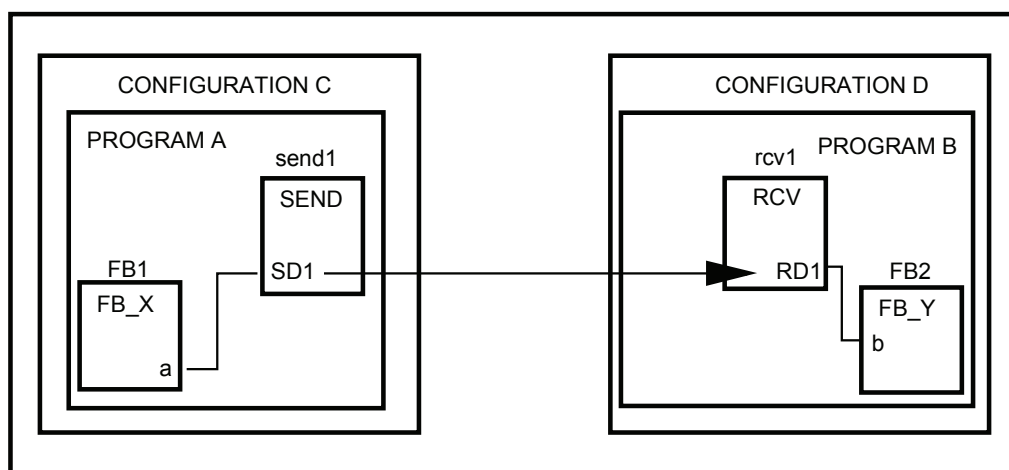


Bild 2 c) – Kommunikationsfunktionsbausteine

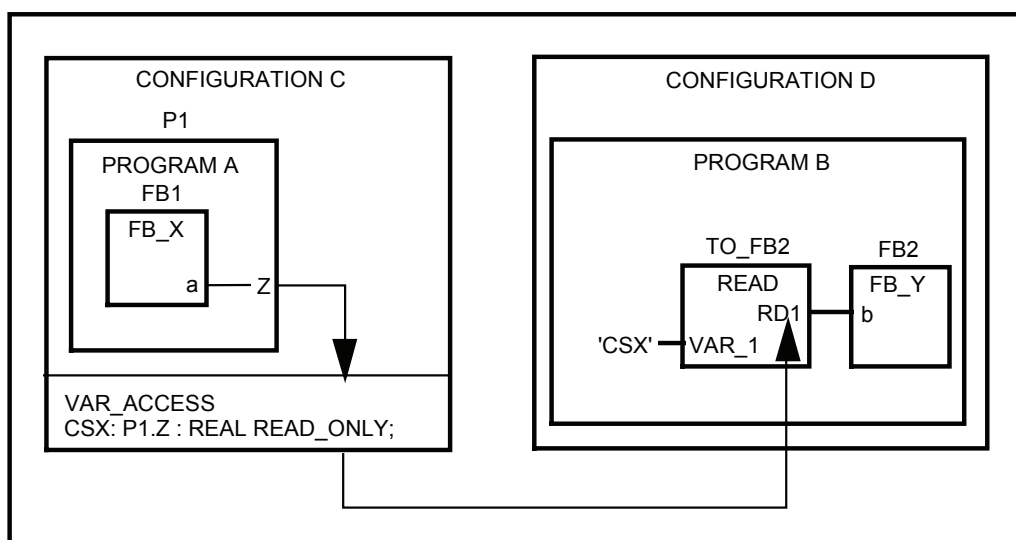


Bild 2 d) – Kommunikation über Zugriffspfade

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

ANMERKUNG 2 In diesen Beispielen sollen die Konfigurationen C und D jeweils eine einzige Ressource haben.

ANMERKUNG 3 Die Einzelheiten der Kommunikationsbausteine sind in diesem Bild nicht gezeigt. Siehe 2.5.2.3.5 und IEC 61131-5.

ANMERKUNG 4 Wie in 2.5 festgelegt, können die Zugriffspfade auf direkt dargestellte Variable, globale Variable oder Eingangs-, Ausgangs- oder interne Variable von Programmen oder Funktionsbaustein-Instanzen deklariert werden.

ANMERKUNG 5 IEC 61131-5 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-Programmiersprachen und die Abschnitte, in denen sie in diesem Teil erscheinen, sind folgendermaßen eingeteilt:

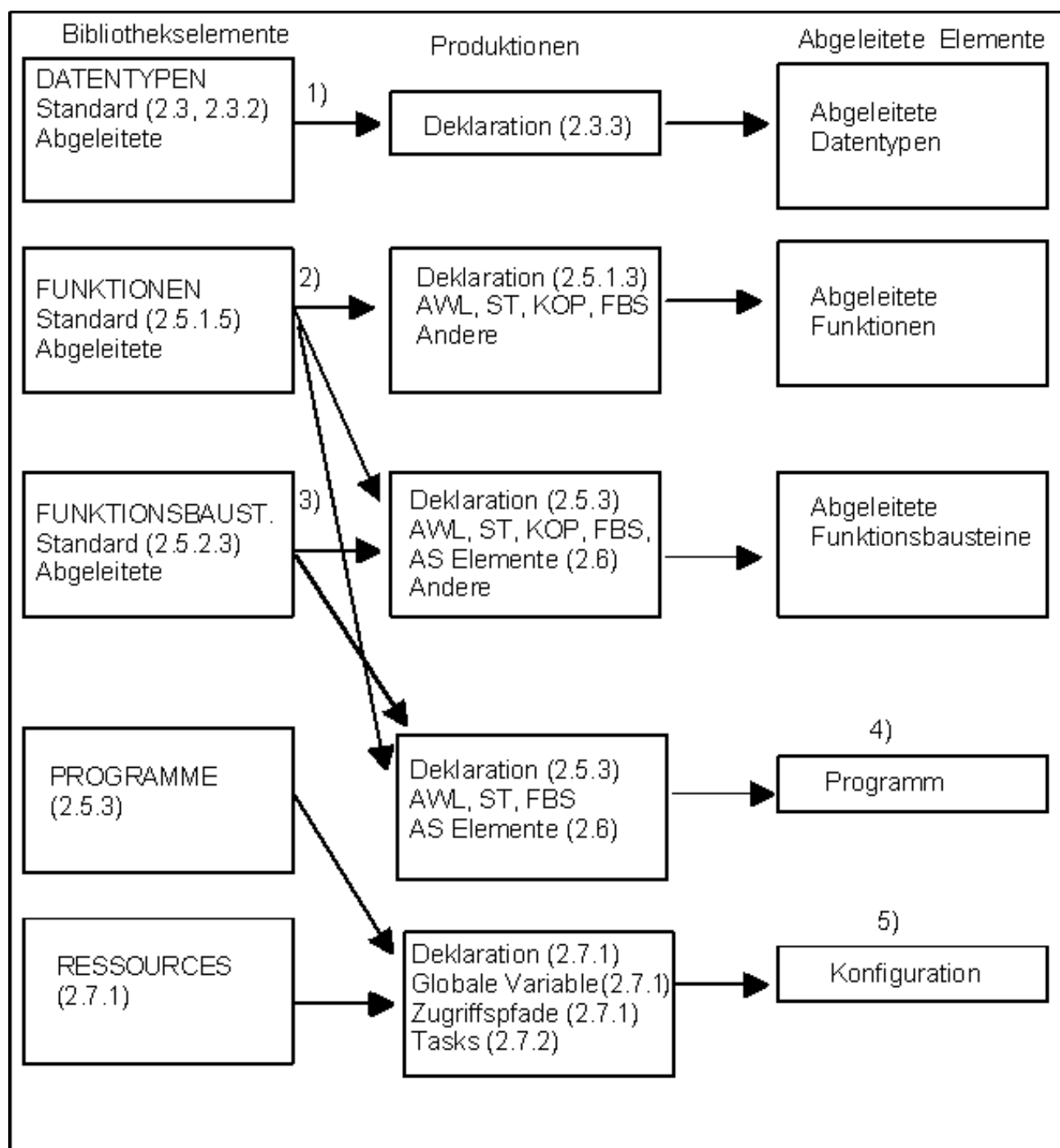
- Datentypen (2.3)
- Variable (2.4)
- Programm-Organisationseinheiten (2.5)
 - Funktionen (2.5.1)
 - Funktionsbausteine (2.5.2)
 - Programme (2.5.3)
- Ablaufsprache- (AS) Elemente (2.6)
- Konfigurationselemente (2.7)
 - Globale Variable (2.7.1)
 - Ressourcen (2.7.1)
 - Zugriffspfade (2.7.1)
 - Tasks (2.7.2)

Wie in Bild 3 gezeigt, muss die Kombination dieser Elemente die folgenden Regeln beachten:

- 1) Abgeleitete *Datentypen* müssen deklariert werden, wie es 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 es in 2.5.1.3 festgelegt ist, indem Standard- oder abgeleitete Datentypen, Standardfunktionen, die in 2.5.1.5 definiert sind, und alle vorher abgeleiteten Funktionen benutzt werden. Diese Deklaration muss die Mechanismen für die Sprache AWL, ST, KOP oder FBS definieren.
- 3) Abgeleitete *Funktionsbausteine* können deklariert werden, wie es in 2.5.2.2 festgelegt ist, indem Standard- und 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 Sprache AWL, ST, KOP oder FBS definiert sind und kann die Elemente der Ablaufsprache (AS) einschließen, die in 2.6 definiert sind.
- 4) Ein *Programm* muss deklariert werden, wie es in 2.5.3 festgelegt ist, indem Standard- oder abgeleitete Datentypen, Funktionen und Funktionsbausteine benutzt werden. Diese Deklaration muss Mechanismen benutzen, die definiert sind für die Sprache AWL, ST, KOP oder FBS und kann die Elemente der Ablaufsprache (AS) einschließen, die in 2.6. definiert sind.
- 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 Funktionsbausteins aufruft und die Zugriffe auf die zugehörigen Daten ausführt, müssen so beschaffen sein, wie in dieser Norm definiert ist.



ANMERKUNG 1 Die eingeklammerten Zahlen (1) bis (5) verweisen auf die entsprechenden Absätze in 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

1.5 Normerfüllung

Dieser Abschnitt definiert die Anforderungen, die von SPS-Systemen und -Programmen erfüllt werden müssen, die den Anspruch auf Erfüllung dieses Teils der IEC 61131 erheben.

1.5.1 Normerfüllung des Systems

Ein SPS-System, wie in IEC 61131-1 definiert ist, das den Anspruch erhebt, vollständig oder teilweise die Anforderungen dieses Teils der IEC 61131 zu erfüllen, muss dies genau so tun, wie es 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 61131-3 in folgenden Eigenschaften der Sprache:“,

gefolgt von einer Reihe von Tabellen in folgendem Format:

Tabellen-Überschrift

Tabellen-Nummer	Eigenschaftsnummer	Beschreibung der Eigenschaften
...	...	...

Die Nummern der Tabellen und Eigenschaften und die Beschreibungen sind aus den Tabellen zu entnehmen, die in den relevanten Abschnitten dieses Teils der IEC 61131 angegeben sind. Die Tabellenüberschriften sind aus der folgenden Tabelle zu entnehmen.

Tabellen-Überschrift	Für Eigenschaft in:
Gemeinsame Elemente	Abschnitt 2
Gemeinsame Text-Elemente	Abschnitt 3.1
AWL Sprach-Elemente	Abschnitten 3.2.1 bis 3.2.3
ST Sprach-Elemente	Abschnitten 3.3.1 bis 3.3.2.4
Gemeinsame Grafik-Elemente	Abschnitt 4.1
KOP-Sprach-Elemente	Abschnitt 4.2
FBS Sprach-Elemente	Abschnitt 4.3

Die Tabellen 9, 11, 13, 16a, 16b, 32, 38, 47, 48 und 51 dürfen nicht zur Normerfüllung als Eigenschaftentabellen betrachtet werden.

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

- darf nicht ersetzende oder zusätzliche Sprachelemente einschließen, um eine der Eigenschaften zu erreichen, die in diesem Teil festgelegt sind, es sei denn, solche Elemente sind unten in den Regeln e) und f) angegeben;
- muss ein Schriftstück mitliefern, das die Daten aller **implementierungsabhängiger Parameter** angibt, wie es in Anhang D aufgelistet ist;
- muss imstande sein, zu unterscheiden, ob ein Sprachelement des Anwenders eine Anforderung dieser Norm verletzt oder nicht; muss erkennen können, wo ein solcher Verstoß 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 imstande sein, 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 Anwender-definierter 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 Systemeigenschaften beschreibt, die vom System akzeptiert werden und die in diesem Teil verboten oder nicht festgelegt sind. Derartige Eigenschaften sind als „Erweiterungen zur <Sprache> Sprache, die in IEC 61131-3 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, wie es für Fehler festgelegt ist, jeden Gebrauch einer der **implementierungsabhängigen Eigenschaften**, die in Anhang D festgelegt sind, zu verarbeiten;
- h) darf nicht die in dieser Norm definierten Namen für Standard-Datentypen, Funktionen oder Funktionsbausteine für hersteller-definierte Eigenschaften nutzen, deren Funktionalität sich von dem unterscheidet, was in dieser Norm beschrieben ist; es sei denn, solche Elemente sind oben in den Regeln e) und f) angegeben;
- 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;
- j) muss imstande sein, Dateien zu lesen und zu schreiben, die Sprachelemente enthalten, die als Alternativen in der Erzeugung `library_element_declaration` in B.0 definiert sind, in der Syntax in der obigen Anforderung i) definiert sind oder nach der „ISO-646 IRV“ codiert sind, die als Tabelle 1 – Spalte 00 der ISO/IEC 10646-1 angegeben ist.

Die Formulierung „muss imstande sein“ wird in diesem Abschnitt 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 Abschnitts erfüllen.

1.5.2 Normerfüllung der Programme

Ein SPS-Programm, das die Anforderungen der IEC 61131-3 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; diese Ergebnisse sind von der Programm-Ausführungszeit, der Anwendung von **implementierungsabhängigen Eigenschaften** (wie in Anhang D aufgeführt) im Programm und der Ausführung von Prozeduren zur Fehlerbehandlung beeinflusst.

2 Gemeinsame Elemente

Dieser Abschnitt definiert Text- und Grafikelemente, die für alle SPS-Programmiersprachen dieses Teils der IEC 61131 gemeinsam sind.

2.1 Gebrauch der gedruckten Zeichen

2.1.1 Zeichensatz

Die Textsprachen und die Textelemente der Grafiksprachen müssen nach der „ISO-646 IVR“ dargestellt werden, die als Tabelle 1 – Spalte 00 der ISO/IEC 10646-1 angegeben ist.

Der Gebrauch von Zeichen der zusätzlichen Zeichensätze, z. B. des „Latin-1 Supplement“, das als Tabelle 2 Spalte 00 des ISO/IEC 10646-1 angegeben ist, ist eine typische Erweiterung der Norm. Die Codierung solcher Zeichen muss mit der ISO/IEC 10646 übereinstimmen.

Der **geforderte Zeichensatz** besteht aus allen Zeichen in den Spalten 002 bis 007 der „ISO/IEC-646 IRV“, wie es oben definiert ist, ausgenommen der Kleinbuchstaben.

Tabelle 1 – Zeichensatz

Nr.	Beschreibung
2	Kleinbuchstaben ^a
3a	Nummernzeichen (#) oder
3b	Pfundzeichen (£)
4a	Dollarzeichen (\$) oder
4b	Währungszeichen (¤)
5a	Senkrechter Strich () oder
5b	Ausrufungszeichen (!)
ANMERKUNG Die Nummerierung der Eigenschaften in dieser Tabelle ist so gestaltet, dass eine Übereinstimmung mit der ersten Ausgabe der IEC 61131-3 erhalten bleibt.	
^a Wenn Kleinbuchstaben (Eigenschaft 2) unterstützt werden, darf die Groß-/Kleinschreibung in den Sprachelementen nicht signifikant sein, ausgenommen innerhalb Kommentaren wie in 2.1.5 definiert, Zeichenfolgen wie in 2.2.2 definiert und Variablen vom Typ STRING und WSTRING wie in 2.3.1 definiert.	

2.1.2 Bezeichner

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

Die Groß-/Kleinschreibung von Buchstaben darf in Bezeichnern nicht signifikant sein; z. B. müssen die Bezeichner `abcd`, `ABCD` und `aBCd` gleich interpretiert werden.

Unterstriche müssen in Bezeichnern signifikant sein; z. B. `A_BCD` und `AB_CD` müssen als unterschiedliche Bezeichner interpretiert werden. Mehrere führende und mehrfache Unterstriche sind nicht zulässig; z. B. sind die Zeichenfolgen `__LIM_SW5` und `LIM__SW5` keine gültigen Bezeichner. Angehängte Unterstriche sind nicht zulässig; z. B. ist die Zeichenfolge `LIM_SW5_` kein gültiger Bezeichner.

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. Die maximale Anzahl von Zeichen, die in einem Bezeichner zulässig ist, ist ein **implementierungsabhängiger** Parameter.

Eigenschaften und Beispiele werden in Tabelle 2 gezeigt.

Tabelle 2 – Bezeichner-Eigenschaften

Nr.	Beschreibung	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 aufgeführte 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 dieser Norm benutzt werden, sind in Anhang C aufgelistet. Schlüsselwörter dürfen keine eingebetteten Leerzeichen enthalten. Die Groß-/Kleinschreibung der Zeichen darf in Schlüsselwörtern nicht signifikant sein, z. B. sind die Schlüsselwörter „FOR“ und „for“ syntaktisch gleichbedeutend. 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 Zeichen als „Leerzeichen“ überall im Text des SPS-Programms einzufügen, ausgenommen innerhalb von Schlüsselwörtern, Literalen, Aufzählungswerten, Bezeichnern, direkt-dargestellten Variablen, wie in 2.4.1.1 beschrieben, oder Kombinationen von Begrenzern (z. B. für Kommentare, wie es in 2.1.5 definiert ist). „Leerzeichen“ ist sowohl als SPACE Zeichen mit dem Code-Wert 32 dezimal als auch als nicht-druckbare Zeichen wie Tabulator, Neue-Zeile usw., für die keine Kodierung in ISO/IEC 10646-1 angegeben sind, definiert.

2.1.5 Kommentare

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

Der Gebrauch von geschachtelten Kommentaren, z. B. `(* (* GESCHACHTELT *) *)`, muss als ein **Fehler** nach den Regeln von 1.5.1 d) behandelt werden.

Die maximale Anzahl der Zeichen, die in einem Kommentar zulässig ist, ist ein **implementierungsabhängiger Parameter**.

Tabelle 3 – Kommentar-Eigenschaft

Nr.	Beschreibung	Beispiel
1	Kommentare	<pre>(***** (* Ein eingerahmter Kommentar *) *****)</pre>
ANMERKUNG Das Beispiel oben enthält drei einzelne Kommentare.		

2.1.6 Pragma

Wie in Tabelle 3a veranschaulicht, müssen Pragmas am Beginn und Ende durch geschweifte Klammern „{“ bzw. „}“ eingeschlossen werden. Die Syntax und Semantik von speziellen Pragma-Konstruktionen sind Implementierungsabhängig. Pragmas müssen überall in Programmen zulässig sein, wo Leerzeichen erlaubt sind, ausgenommen innerhalb von Zeichen folgen-Literalen, wie in 2.2.2 definiert.

ANMERKUNG Geschweifte Klammern innerhalb eines Kommentars haben keine semantische Bedeutung; Kommentare innerhalb geschweiften Klammern dürfen abhängig von der Implementierung eine semantische Bedeutung haben oder auch nicht.

Tabelle 3a – Pragma-Eigenschaften

Nr.	Beschreibung der Eigenschaften	Beispiele
1	Pragmas	<pre>{VERSION 3.1} {AUTHOR JHC} {x := 256, y := 384}</pre>

2.2 Externe Darstellung von Daten

Externe Darstellungen von Daten bei den verschiedenen SPS-Programmiersprachen müssen die numerischen Literale, Zeichenfolge 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 Zehnerpotenz an, mit der die vorausgehende 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. Die Basis muss in dezimaler Angabe erfolgen. Für die Basis 16 muss ein erweiterter Satz von Zeichen benutzt werden, der aus den Buchstaben A bis F besteht, die die übliche 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) oder durch die Schlüsselwörter FALSE oder TRUE dargestellt werden.

Die Eigenschaften und Beispiele für numerische Literale sind in Tabelle 4 gezeigt.

Der *Datentyp* von booleschen oder numerischen Literalen darf durch Voranstellen eines Datentyp-Präfixes festgelegt werden und erfolgt durch den Namen des Datentyps und einem „#“-Zeichen. Für Beispiele vergleiche Eigenschaft 9 in der Tabelle 4.

Tabelle 4 – Numerische Literale

Nr.	Beschreibung	Beispiele
1	Ganzzahlige (integer) Literale	-12 0 123_456 +986
2	Reele (real) Literale	-12.0 0.0 0.456 3.14159_26
3	Reele (real) 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 (224 dezimal)
5	Basis-8 Literale	8#377 (255 dezimal) 8#340 (224 dezimal)
6	Basis-16 Literale	16#FF oder 16#ff (255 dezimal) 16#E0 oder 16#e0 (224 dezimal)
7	Boolesche Null und Eins	0 1
8	Boolesche FALSE und TRUE	FALSE TRUE
9	Literale mit Typangabe	DINT#5 (DINT Darstellung von 5) UINT#16#9AF (UINT Darstellung des hexadezimalen Werts 9AF) BOOL#0 BOOL#1 BOOL#TRUE BOOL#FALSE
ANMERKUNG Die Schlüsselwörter FALSE and TRUE korrespondieren mit den booleschen Werten 0 und 1.		

2.2.2 Zeichenfolge-Literale

Zeichenfolge-Literale sind entweder Einzel-Byte- oder Doppel-Byte-Zeichen.

Ein Zeichenfolge-Literal, das ein Einzel-Byte-Zeichen ist, ist eine Sequenz von null oder mehr Zeichen aus der Reihe 00 des ISO/IEC 10646-1 Zeichensatzes, die durch Einzel-Anführungszeichen (') eingeleitet und abgeschlossen ist. In Einzel-Byte-Zeichenfolgen muss die 3-Zeichen-Kombination, bestehend aus einem Dollar-Zeichen (\$) und gefolgt von zwei hexadezimalen Ziffern, als die hexadezimale Darstellung des 8-Bit-Zeichencode interpretiert werden. Dies ist in Eigenschaft 1 von Tabelle 5 gezeigt.

Ein Zeichenfolge-Literal, das ein Doppel-Byte-Zeichen ist, ist eine Sequenz von null oder mehr Zeichen aus dem ISO/IEC 10646-1 Zeichensatz, die durch Doppel-Anführungszeichen (") eingeleitet und abgeschlossen ist. In Doppel-Byte-Zeichenfolgen muss die 5-Zeichen-Kombination, bestehend aus einem Dollar-Zeichen (\$) und gefolgt von vier hexadezimalen Ziffern, als die hexadezimale Darstellung des 16-Bit-Zeichencode interpretiert werden. Dies ist in Eigenschaft 2 von Tabelle 5 gezeigt.

2-Zeichen-Kombinationen, die mit dem Dollar-Zeichen beginnen, müssen interpretiert werden, wie es in Tabelle 6 gezeigt ist, wenn sie in Zeichenfolgen auftreten.

Tabelle 5 – Zeichenfolge Literal

Nr.	Beispiel	Erläuterung
1	Einzel-Byte-Zeichenfolge	
	' '	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 „Einzel-Anführungszeichen“
	' '' '	Zeichenfolge der Länge eins mit „Doppel-Anführungszeichen“
	'\$R\$L'	Zeichenfolge der Länge zwei mit CR und LF Zeichen
	'\$0A'	Zeichenfolge der Länge zwei mit LF Zeichen
	'\$\$1.00'	Zeichenfolge der Länge fünf, die als „\$1.00“ gedruckt wird
	'ÄË'	Gleiche Zeichenfolgen der Länge 2
	'\$C4\$CB'	
2	Doppel-Byte-Zeichenfolge	
	""	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 „Einzel-Anführungszeichen“
	"\$ " "	Zeichenfolge der Länge eins mit „Doppel-Anführungszeichen“
	"\$R\$L"	Zeichenfolge der Länge zwei mit LF Zeichen
	"\$\$1.00"	Zeichenfolge der Länge fünf, die als „\$1.00“ gedruckt wird
	"ÄË"	Gleiche Zeichenfolgen der Länge 2
	"\$00C4\$00CB"	
3	Einzel-Byte-Zeichenfolge-Literal mit Typangabe	
	STRING#'OK'	Zeichenfolge der Länge zwei mit zwei Einzel-Byte-Zeichen
4	Doppel-Byte Zeichenfolge-Literal mit Typangabe	
	WSTRING#'OK'	Zeichenfolge der Länge zwei mit zwei Doppel-Byte-Zeichen
ANMERKUNG Wenn eine bestimmte Implementierung die Eigenschaft Nr. 4 unterstützt, jedoch nicht die Eigenschaft Nr. 2 unterstützt, darf der Implementierer eine herstellerspezifische Syntax und Semantik für den Gebrauch der Doppel-Anführungszeichen festlegen.		

Tabelle 6 – Zwei-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 (Form feed, page)
7	\$ R oder \$ r	Wagenrücklauf (Carriage return)
8	\$ T oder \$ t	Tabulator (Tab)
9	\$ "	Doppel-Anführungszeichen

ANMERKUNG 1 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 der nächsten Zeile.

ANMERKUNG 2 Die Kombination \$ ' ist nur innerhalb von Zeichenfolge-Literalen in Einzel-Anführungszeichen zulässig.

ANMERKUNG 3 Die Kombination \$ " ist nur innerhalb von Zeichenfolge-Literalen in Doppel-Anführungszeichen zulässig.

2.2.3 Zeitlitterale

Es ist notwendig, externe Darstellungen für zwei unterschiedliche Typen von Zeitdaten zur Verfügung zu stellen: Daten für die *Zeitdauer* zur Messung oder 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#` begrenzt werden. Die Darstellung von Daten für 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.

Wie in Tabelle 7 veranschaulicht, sind sowohl positive wie auch negative Werte für die Zeitdauerangaben zulässig.

Tabelle 7 – Zeitdauer Literal

Nr.	Beschreibung	Beispiele
1a	Zeitdauer ohne Unterstriche: kurzes Präfix	T#14ms T#-14ms T#14.7s T#14.7m T#14.7h t#14.7d t#25h15m t#5d14h12m18s3.5ms
1b	langes Präfix	TIME#14ms TIME#-14ms time#14.7s
2a	Zeitdauer mit Unterstrichen: kurzes Präfix	t#25h_15m t#5d_14h_12m_18s_3.5ms
2b	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 sind in Tabelle 8 gezeigt. Wie in Tabelle 9 veranschaulicht, muss die Darstellung der Information von Tageszeit und Datum so erfolgen, wie es durch die Syntax festgelegt ist, die in B.1.2.3.2 angegeben ist.

Tabelle 8 – Literale für Datum und Tageszeit

Nr.	Beschreibung	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

Durch diese Norm wird eine Anzahl von elementaren (vordefinierten) Datentypen zugelassen. Zusätzlich werden allgemeine (generische) Datentypen zur Anwendung bei der Definition von überladenen (typunabhängigen) Funktionen (siehe 2.5.1.4) definiert. Außerdem ist ein Verfahren für den Anwender oder den Hersteller definiert, um zusätzliche Datentypen festzulegen.

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	N ^a
1	BOOL	boolesche	1 ^h
2	SINT	kurze ganze Zahl (short integer)	8 ^c
3	INT	ganze Zahl (integer)	16 ^c
4	DINT	doppelte ganze Zahl (double integer)	32 ^c
5	LINT	lange ganze Zahl (long integer)	64 ^c
6	USINT	vorzeichenlose kurze ganze Zahl (unsigned short integer)	8 ^d
7	UINT	vorzeichenlose ganze Zahl	16 ^d
8	UDINT	vorzeichenlose doppelte ganze Zahl	32 ^d
9	ULINT	vorzeichenlose lange ganze Zahl	64 ^d
10	REAL	reelle Zahl	32 ^e
11	LREAL	lange reelle Zahl	64 ^f
12	TIME	Zeitdauer	-- ^b
13	DATE	Datum (nur)	-- ^b
14	TIME_OF_DAY oder TOD	Uhrzeit (nur)	-- ^b
15	DATE_AND_TIME oder DT	Datum und Uhrzeit	-- ^b
16	STRING	variabel-lange Zeichenfolge	8 ^{i, g}
17	BYTE	Bit-Folge 8	8 ^{j, g}
18	WORD	Bit-Folge 16	16 ^{j, g}
19	DWORD	Bit-Folge 32	32 ^{j, g}
20	LWORD	Bit-Folge 64	64 ^{j, g}
21	WSTRING	Variable lange Doppel-Byte-Zeichenfolge	16 ^{i, g}
^a Einträge in dieser Spalte müssen so interpretiert werden, wie es in den Fußnoten festgelegt ist. ^b Der Wertebereich und die Genauigkeit der Darstellung in diesen Datentypen ist implementierungsabhängig. ^c Der Wertebereich für Variablen dieses Datentyps reicht von $-(2^{N-1})$ bis $(2^{N-1}) - 1$. ^d Der Wertebereich für Variablen dieses Datentyps reicht von 0 bis $(2^N) - 1$. ^e Der Wertebereich für Variablen dieses Datentyps muss festgelegt sein, wie es in IEC 60559 für das Basis-Gleitpunkt-Format mit einfacher Länge definiert ist. ^f Der Wertebereich für Variablen dieses Datentyps muss festgelegt sein, wie es in IEC 60559 für das Basis-Gleitpunkt-Format mit doppelter Länge definiert ist. ^g Ein numerischer Wertebereich ist für diesen Datentyp nicht anwendbar. ^h Die möglichen Werte für Variablen dieses Datentyps müssen 0 und 1 sein, entsprechend den Schlüsselwörtern FALSE und TRUE. ⁱ Der Wert N gibt die Anzahl der Bits/Zeichen für diesen Datentyp an. ^j Der Wert N gibt die Anzahl der Bits in der Bitfolge für diesen Datentyp an.			

2.3.2 Allgemeine (generische) Datentypen

Zusätzlich zu den Datentypen, die in Tabelle 10 gezeigt sind, kann die Hierarchie der allgemeinen Datentypen, die in Tabelle 11 gezeigt ist, bei der Festlegung von Eingängen und Ausgängen von Standardfunktionen und Funktionsbausteinen angewendet werden (siehe 2.5.1.4). Die allgemeinen Datentypen werden durch die Vorsilbe „ANY“ identifiziert. Der Gebrauch von allgemeinen Datentypen ist in den folgenden Regeln beschrieben:

- 1) Allgemeine Datentypen dürfen nicht in anwender-deklarierten Programm-Organisationseinheiten, die in 2.5 definiert sind, verwendet werden.
- 2) Der allgemeine Typ eines abgeleiteten *Unterbereichstyps* (Merkmal 3 von Tabelle 12) muss `ANY_INT` sein.
- 3) Der allgemeine Typ eines *direkt abgeleiteten* Typs (Eigenschaft 1 der Tabelle 12) muss derselbe sein wie der elementare Typ, von dem er abgeleitet ist.
- 4) Der allgemeine Typ von allen anderen abgeleiteten Typen, die in Tabelle 12 definiert sind, muss `ANY_DERIVED` sein.

Tabelle 11 – Hierarchie der allgemeinen Datentypen

ANY
ANY_DERIVED (Abgeleitete Datentypen – siehe vorausgegangener Text)
ANY_ELEMENTARY
ANY_MAGNITUDE
ANY_NUM
ANY_REAL
LREAL
REAL
ANY_INT
LINT, DINT, INT, ULINT, SINT
ULDINT, UDINT, UINT, USINT
TIME
ANY_BIT
LWORD, DWORD, WORD, BYTE, BOOL
ANY_STRING
STRING
WSTRING
ANY_DATE
DATE_AND_TIME
DATE, TIME_OF_DAY

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 angewendet werden, wie in 2.4.3 gezeigt ist.

Eine Deklaration für den Datentyp *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. Die Aufzählungsliste definiert eine geordnete Menge von aufgezählten Werten, beginnend mit dem ersten Bezeichner der Liste und endend mit dem letzten. Verschiedene Aufzählungstypen dürfen die gleichen Bezeichner für aufgezählte Werte verwenden. Die maximal zulässige Anzahl der aufgezählten Werte ist ein **implementierungsabhängiger Parameter**.

Um eine eindeutige Identifikation in einem bestimmten Kontext zu gewährleisten, dürfen die aufgezählten Literale durch ein Präfix beschrieben werden, das aus dem zugehörigen Datentypnamen und den '#'-Zeichen besteht, so wie die Literale mit Typangabe, die in 2.2.1 definiert sind. Ein derartiges Präfix darf nicht innerhalb einer Aufzählungsliste verwendet werden. Es ist ein **Fehler**, wenn für ein aufgezähltes Literal nicht ausreichend Information zur Verfügung gestellt wird, um seinen Wert eindeutig zu bestimmen.

Eine Deklaration für den *Unterbereich* legt fest, dass der Wert 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. Es ist ein **Fehler**, wenn der Wert eines Unterbereichstyps außerhalb der festgelegten Bereichswerte liegt.

Eine Deklaration für Struktur (STRUCT) legt fest, dass Datenelemente dieses Typs Unterelemente von festgelegten Typen enthalten müssen, auf die mit festgelegten Namen zugegriffen werden kann. Z. B. 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. Die maximale Anzahl von Strukturelementen, die maximale Menge von Daten, die in einer Struktur einhalten sein kann, und die maximale Anzahl von geschachtelten Ebenen von zugehörigen Strukturelementen sind **implementierungsabhängige** Parameter.

Eine Deklaration für Feld (ARRAY) legt fest, dass eine ausreichende Menge von Datenspeicher für jedes Element dieses Typs zugewiesen werden muss, um alle die Daten zu speichern, auf die mit dem (den) festgelegten Index-Unterbereich(en) 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. Die maximale Anzahl der Feldindizes, die maximale Feldgröße und der maximale Bereich von Index-Werten sind **implementierungsabhängige** Parameter.

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. Z. B. haben, wie in den Tabellen 12, Nr. 2 und 14, Nr. 2 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_DATEN gleich -4095, 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 TYP-Deklaration 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 und WSTRING 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. Die maximal zulässige Länge von STRING und WSTRING ist ein **implementierungsabhängiger Parameter**.

Tabelle 12 – Eigenschaften der Datentyp-Deklarationen

Nr.	Eigenschaft/Beispiel in Textform
1	Direkte Ableitung von elementaren Typen, z. B.: TYPE RU_REAL : 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.: <pre> 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 </pre>
ANMERKUNG Beispiele für den Gebrauch dieser Typen in Deklarationen von Variablen siehe 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)
WSTRING	" " (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.: <pre>TYPE FREQ : REAL := 50.0 ; END_TYPE</pre>
2	Initialisierung von Datentypen für Aufzählung, z. B.: <pre>TYPE ANALOG_SIGNAL_RANGE : (BIPOLAR_10V, (* -10 to +10 VDC *) UNIPOLAR_10V, (* 0 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</pre>
3	Initialisierung von Datentypen für Bereich, z. B.: <pre>TYPE ANALOG_DATAZ : INT (-4095..4095) := 0 ; END_TYPE</pre>
4	Initialisierung von Datentypen für Feld, z. B.: <pre>TYPE ANALOG_16_INPUT_DATAI : ARRAY [1..16] OF ANALOG_DATA := [8(-4095), 8(4095)] ; END_TYPE</pre>
5	Initialisierung von Datentypen für Struktur, z. B.: <pre>TYPE ANALOG_CHANNEL_CONFIGURATIONI : STRUCT RANGE : ANALOG_SIGNAL_RANGE ; MIN_SCALE : ANALOG_DATA := -4095 ; MAX_SCALE : ANALOG_DATA := 4095 ; END_STRUCT ; END_TYPE</pre>
6	Initialisierung von Datentypen für abgeleitete Struktur, z. B.: <pre>TYPE ANALOG_CHANNEL_CONFIGZ : ANALOG_CHANNEL_CONFIGURATIONI := (MIN_SCALE := 0, MAX_SCALE := 4000) ; END_TYPE</pre>

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 Einzelement-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 `RU_REAL` und `FREQ`, 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 *Variablen* die Mittel zur Identifizierung von Datenobjekten, deren Inhalt sich ändern darf; z. B. Daten, die Eingängen, Ausgängen oder Speicherplatz der SPS zugeordnet 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, darstellt, oder die einen abgeleiteten Typ für Aufzählung oder Unterbereich, wie in 2.3.3.1 definiert, darstellt, oder die einen abgeleiteten Typ, dessen „Elternschaft“, wie in 2.3.3.3 rekursiv definiert, darstellt, der zu einem elementaren Aufzählungs- oder Unterbereichstyp zurückverfolgt werden kann. Dieser Abschnitt definiert die Mittel zur Darstellung derartiger Variablen auf *symbolische* Art oder alternativ in einer Weise, die *direkt* die Zuordnung 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, welches durch folgende Aneinanderreihung gebildet wird: Das Prozentzeichen „%“ (Zeichencode 037 dezimal in Tabelle 1 – Reihe 00 von ISO/IEC 10646-1), 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.

Im dem Fall, dass eine direkt dargestellte Variable in einer Speicherortzuweisung zu einer internen Variablen im Deklarationsteil eines *Programm*- oder eines *Funktionsbausteintyps*, wie in 2.4.3.1 definiert, verwendet wird, muss ein Stern (Asterisk) „*“ anstelle des Präfix für die Größe und eine oder mehrere vorzeichenlose ganze Zahlen in der Aneinanderreihung verwendet werden, um anzuzeigen, dass die direkte Darstellung noch nicht vollständig spezifiziert ist. Das Prozentzeichen und das Präfix für den Speicherort `I`, `Q` oder `M` aus Tabelle 15 muss immer in der direkten Darstellung vorhanden sein.

In beiden Fällen erfordert die Anwendung dieser Eigenschaft, dass der so deklarierte Speicherort der Variablen vollständig innerhalb der Konstruktion `VAR_CONFIG...END_VAR` der Konfiguration, wie in 2.7.1 definiert, für alle Instanzen des Typs, der das Konstrukt enthält, festgelegt sein muss.

Es ist in einer Instanz von Programm- oder Funktionsbausteintypen, die solche unvollständigen Festlegungen enthalten, ein **Fehler**, wenn eine der vollständigen Festlegungen in der `VAR_CONFIG...END_VAR` Konstruktion für eine unvollständige Adressenfestlegung fehlt, die durch die Stern-Schreibweise ausgedrückt ist.

Beispiele

%QX75 und %Q75	Ausgangsbit 75
%IW215	Eingangswort Speicherort 215
%QB7	Ausgangsbyte Speicherort 7
%MD48	Merker-Doppelwort Speicherort 48
%IW2.5.7.1	siehe Erläuterung unten
%Q*	Ausgang an einem noch nicht festgelegten Speicherort

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. Wenn eine direkte Darstellung mit zusätzlichen ganzzahligen Feldern erweitert ist, die durch Punkte getrennt sind, muss dies als *hierarchische* physikalische oder logische Adresse interpretiert werden; dabei stellt das linke Feld die höchste Ebene der Hierarchie dar, die folgenden niederen Ebenen erscheinen auf der rechten Seite. Z. B. 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 einer SPS den Datenzugriff in einer weiteren SPS zu erlauben, muss in einer Spracherweiterung betrachtet werden.

Der Gebrauch von direkt dargestellten Variablen ist nur in *Funktionsbausteinen*, wie in 2.5.2 definiert, in *Programmen*, wie in 2.5.3 definiert, und in *Konfigurationen* und *Ressourcen*, wie in 2.7.1 definiert, zugelassen. Die maximale Anzahl der Ebenen einer hierarchischen Adressierung ist ein **implementierungsabhängiger Parameter**.

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

Nr.	Präfix	Bedeutung	Voreingestellter Datentyp
1	I	Speicherort Eingang	
2	Q	Speicherort Ausgang	
3	M	Speicherort Merker	
4	X	(Einzel-) Bit-Größe	BOOL
5	kein	(Einzel-) Bit-Größe	BOOL
6	B	Byte- (8 Bits) Größe	BYTE
7	W	Wort- (16 Bits) Größe	WORD
8	D	Doppelwort- (32 Bits) Größe	DWORD
9	L	Langwort- (64 Bits) Größe	LWORD
10	Der Gebrauch eines Stern (Asterisk, *) zur Anzeige eines noch nicht festgelegten Speicherorts (ANMERKUNG 2)		
ANMERKUNG 1 Nationale Normungsorganisationen können Tabellen mit Übersetzungen dieser Präfixe herausgeben.			
ANMERKUNG 2 Der Gebrauch der Eigenschaft 10 in dieser Tabelle erfordert die Eigenschaft 11 in Tabelle 49 und umgekehrt.			

2.4.1.2 Multielement-Variable

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

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. In der ST-Sprache, die in 3.3 definiert ist, muss ein Index ein Ausdruck sein, der einen

Wert entsprechend einem der Untertypen des allgemeinen Typs `ANY_INT`, wie in Tabelle 11 definiert, liefert. Die Form von Feldindizes in der AWL Sprache, definiert in 3.2, und in den grafischen Sprachen, definiert in Abschnitt 4, ist auf *Einzelelement-Variablen* oder *ganzzahlige Literale* beschränkt.

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

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

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 einzelne Punkte (.) getrennt sind. Der erste Bezeichner stellt den Namen des strukturierten Elements dar und 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 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;
```

2.4.2 Initialisierung

Wenn ein Konfigurationselement (*Ressource* oder *Konfiguration*), wie in 1.4.1 definiert, „gestartet“ wird, kann jede der Variablen, die mit dem Konfigurationselement und seinen *Programmen* verbunden ist, einen der folgenden Anfangswerte annehmen:

- den Wert, den die Variable hatte, als das Konfigurationselement „gestoppt“ wurde (ein *gepuffert* Wert);
- einen anwender-spezifisierten 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` benutzt, das in Tabelle 16 a) festgelegt ist, 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 61131-1 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 61131-1 definiert, müssen die Anfangswerte der *gepufferten* Variablen die anwender-spezifisierten Anfangswerte oder den voreingestellten Wert annehmen, wie in 2.3.3.2 für den zugehörigen Datentyp für jede Variable definiert ist, für die kein Anfangswert durch den Anwender spezifiziert wurde.
- 3) Nicht-gepufferte Variable müssen mit den anwender-spezifisierten Anfangswerten oder mit dem voreingestellten Wert initialisiert werden, wie in 2.3.3.2 für den zugehörigen Datentyp für jede Variable definiert ist, für die kein Anfangswert durch den Anwender spezifiziert wurde.
- 4) Variable, die *Eingänge* eines SPS-Systems, wie in IEC 61131-1 definiert, darstellen, müssen in einer **implementierungsabhängigen** Weise initialisiert werden.

2.4.3 Deklaration

Jede Typdeklaration eines Typs einer Programm-Organisationseinheit (d. h. jede Deklaration eines *Programms*, einer *Funktion* oder eines *Funktionsbausteins*, wie in 2.5 definiert) muss an seinem Anfang mindestens einen *Deklarationsteil* enthalten, der die Typen (und falls notwendig, den physikalischen oder

logischen Speicherort) der Variablen festlegt, die in der Organisationseinheit benutzt werden. Dieser Deklarationsteil muss die Textform eines der Schlüsselwörter VAR, VAR_INPUT oder VAR_OUTPUT, wie in Tabelle 16 a) definiert, besitzen; dabei folgt im Falle von VAR null mal oder einmal die Bestimmungszeichen RETAIN, NON_RETAIN oder das Bestimmungszeichen CONSTANT; und im Falle von VAR_INPUT oder VAR_OUTPUT folgen null mal oder einmal die Bestimmungszeichen RETAIN oder NON_RETAIN, gefolgt von einer oder mehreren 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 im Deklarationsteil (bzw. in Deklarationsteilen) geschehen, wie es in diesem Abschnitt definiert ist.

Tabelle 16 a) – 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
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)
VAR_TEMP	Temporäre Speicherung von Variablen in Funktionsbausteinen und Programmen (2.4.3)
VAR_CONFIG	Instanz-spezifische Initialisierung und Zuweisung des Speicherorts
RETAIN ^{b, c, d, e}	Gepufferte Variablen (siehe vorangehenden Text)
NON_RETAIN ^{b, c, d, e}	Nicht-gepufferte Variablen (siehe vorangehenden Text)
CONSTANT ^a	Konstante (Variable kann nicht geändert werden)
AT	Zuweisung des Speicherorts (2.4.3.1)
ANMERKUNG 1 Der Gebrauch dieser Schlüsselwörter ist eine Eigenschaft der Programm-Organisationseinheit oder des Konfigurationselementes, in dem sie benutzt werden. Normative Anforderungen des Gebrauchs dieser Schlüsselwörter sind in 2.5 und 2.7 angegeben.	
ANMERKUNG 2 Beispiele für den Gebrauch der VAR_IN_OUT Variablen sind in Bild 11 b und Bild 12 angegeben.	
^a Der Bezeichner CONSTANT darf nicht zur Deklaration von <i>Funktionsbaustein-Instanzen</i> verwendet werden, die in 2.5.2.1 beschrieben sind. ^b Die Bezeichner RETAIN und NON_RETAIN dürfen für <i>Variable</i> verwendet werden, die in VAR, VAR_INPUT, VAR_OUTPUT und VAR_GLOBAL Blöcken deklariert sind, jedoch nicht in VAR_IN_OUT Blöcken und nicht für einzelne Elemente von Strukturen. ^c Der Gebrauch von RETAIN und NON_RETAIN für <i>Funktionsbausteine</i> und <i>Programm-Instanzen</i> ist erlaubt. Die Wirkung ist, dass alle Elemente der Instanz als RETAIN oder NON_RETAIN behandelt werden, ausgenommen, wenn: – das Element ausdrücklich als RETAIN oder NON_RETAIN in der Definition des Funktionsbausteins oder Programms deklariert ist; – das Element selbst ein <i>Funktionsbaustein</i> ist. ^d Der Gebrauch von RETAIN und NON_RETAIN für <i>Instanzen</i> von strukturierten Datentypen ist erlaubt. Die Wirkung ist, dass alle Strukturelemente, auch die von geschachtelten Strukturen, als RETAIN oder NON_RETAIN behandelt werden. ^e Sowohl RETAIN als auch NON_RETAIN sind Eigenschaften. Wenn eine Variable weder ausdrücklich als RETAIN noch als NON_RETAIN deklariert ist, ist das „Warmstart“-Verhalten der Variablen implementierungsabhängig .	

Innerhalb von *Funktionsbausteinen* und *Programmen* können Variablen in einer VAR_TEMP...END_VAR Konstruktion deklariert werden. Diese Variablen werden bei jedem Aufruf einer Instanz der Programm-Organisationseinheit zugeordnet und initialisiert und bleiben nicht über Aufrufe hinweg bestehen.

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.

Es muss ein **Fehler** sein, wenn:

- eine Programm-Organisationseinheit versucht, den Wert einer Variablen zu verändern, die mit dem Bestimmungszeichen **CONSTANT** deklariert wurde;
- eine Variable, die als VAR_GLOBAL CONSTANT in einem Konfigurationselement oder einer Programm-Organisationseinheit (das „enthaltende Element“) in einer VAR_EXTERNAL Deklaration (ohne das **CONSTANT** Bestimmungszeichen) eines Elements verwendet wird, das im enthaltenden Element eingeschlossen ist, wie unten veranschaulicht ist.

Die maximale Anzahl von Variablen, die in einen Variablen-Deklarationsblock erlaubt sind, ist ein **implementierungsabhängiger Parameter**.

Tabelle 16 b) – Gebrauch von VAR_GLOBAL, VAR_EXTERNAL und CONSTANT Deklarationen

Deklaration von enthaltenden Elementen	Deklaration von enthaltenen Elementen	Erlaubt?
VAR_GLOBAL X ...	VAR_EXTERNAL CONSTANT X...	Ja
VAR_GLOBAL X ...	VAR_EXTERNAL X...	Ja
VAR_GLOBAL CONSTANT X ...	VAR_EXTERNAL CONSTANT X...	Ja
VAR_GLOBAL CONSTANT X ...	VAR_EXTERNAL X...	Nein

2.4.3.1 Typzuweisung

Wie in Tabelle 17 gezeigt, muss die Konstruktion VAR...END_VAR zur Festlegung von Datentypen und Pufferungsverhalten von direkt dargestellten Variablen angewendet werden. Diese Konstruktion muss auch zur Festlegung von Datentypen, Pufferverhalten und (falls nötig, nur in *Programmen* und VAR_GLOBAL Deklarationen) den physikalischen und logischen Speicherort für symbolisch dargestellte Einzel- und Multi-Elementvariable benutzt werden. Der Gebrauch der Konstruktionen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT ist in 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 der Variablen zu einem geeigneten Speicherort im SPS-Speicher erfolgen.

Die Schreibweise mit Stern (Asterisk) (Eigenschaft Nr. 10 in Tabelle 15) kann bei Adresszuweisungen innerhalb von Typen von Programmen und Funktionsbausteinen verwendet werden, um noch nicht vollständig festgelegte Speicherorte für direkt dargestellte Variablen anzugeben.

Tabelle 17 – Eigenschaften der Typzuweisung für Variablen

Nr.	Eigenschaften/Beispiele	
1 ^a	Deklaration von direkt dargestellten Variablen	
	<pre>VAR AT %IW6.2 : WORD; AT %MW6 : INT ; END_VAR</pre>	16-Bit-Folge (Anmerkung 2) 16-Bit-Integer, Anfangswert = 0
2 ^a	Deklaration von direkt dargestellten gepufferten Variablen	
	<pre>VAR RETAIN AT %QW5 : WORD ; END_VAR</pre>	Beim Kaltstart, wird mit einer 16-Bit-Folge mit dem Wert 16#0000 initialisiert.
3 ^a	Deklaration von Speicherorten bei symbolischen Variablen	
	<pre>VAR_GLOBAL LIM_SW_S5 AT %IX27 : BOOL;</pre>	Weist Eingangsbit 27 der boolesche Variablen LIM_SW_5 zu. (Anmerkung 2)
	<pre>CONV_START AT %QX25 : BOOL;</pre>	Weist Ausgangsbit 25 der boolesche Variablen CONV_START zu.
	<pre>TEMPERATURE AT %IW28 : INT;</pre>	Weist Eingangswort 28 der ganzzahligen Variablen TEMPERATURE zu. (Anmerkung 2)
	<pre>VAR C2 AT %Q* : BYTE ; END_VAR</pre>	Weist noch nicht zugeordnetes Ausgangsbyte der Bit-Folge-Variable C2 der Länge 8 Bit zu.
4 ^a	Zuweisung von Speicherort bei Feld	
	<pre>VAR INARY AT %IW6 : ARRAY [0..9] OF INT; END_VAR</pre>	Deklariert ein Feld von 10 ganzen Zahlen, das dem zusammenhängenden Eingangsspeicherort beginnend ab %IW6 zugeordnet ist. (Anmerkung 2)
5	Automatische Speicherzuteilung für symbolische Variablen	
	<pre>VAR CONDITION_RED : BOOL; IBOUNCE : WORD ; MYDUB : DWORD ; AWORD, BWORD, CWORD : INT; MYSTR: STRING[10] ; END_VAR</pre>	Teilt ein Speicher-Bit der booleschen Variablen CONDITION_RED zu. Teilt ein Speicher-Wort der 16-Bit-Folge-Variablen IBOUNCE zu. Teilt ein Speicher-Doppelwort der 32-Bit-Folge-Variablen MYDUB zu. Teilt 3 separate Speicher-Wörter den ganzzahligen Variablen AWORD, BWORD und CWORD zu. Teilt Speicher zu, der eine Zeichenfolge mit einer Maximallänge von 10 Zeichen enthält. Nach der Initialisierung hat die Folge die Länge 0 und enthält die leere Folge ' '.
6	Deklaration für Feld	
	<pre>VAR THREE : ARRAY [1..5, 1..10, 1..8] OF INT; END_VAR</pre>	Teilt für ein drei-dimensionales Feld von ganzen Zahlen 400 Speicher-Wörter zu.
7	Deklaration für gepuffertes Feld	
	<pre>VAR RETAIN RTBT: ARRAY [1..2, 1..3] OF INT; END_VAR</pre>	Deklariert das gepufferte Feld RTBT mit den „Kaltstart“-Anfangswerten 0 für alle Elemente.

Nr.	Eigenschaften/Beispiele	
8	Deklaration für strukturierte Variablen	
	<pre>VAR MODULE_8_CONFIG : ANALOG_16_INPUT_CONFIGURATION; END_VAR</pre>	Deklaration einer Variablen mit abgeleitetem Datentyp (siehe Tabelle 12).
ANMERKUNG 1 Die Initialisierung von Systemeingängen ist implementierungsabhängig , siehe 2.4.2.		
ANMERKUNG 2 Die Anmerkungen zu Tabelle 16 a) gelten auch für diese Tabelle.		
^a Wenn direkt dargestellten Variablen explizit der Speicherort zugewiesen wird, können die Eigenschaften 1 bis 4 nur in PROGRAM und VAR_GLOBAL Deklarationen angewendet werden, wie in 2.5.3 bzw. 2.7.1 definiert. Wenn die Schreibweise mit Stern von Eigenschaft 10 in Tabelle 15 verwendet wird, um die Instanz-spezifische Speicherort-Zuweisung für eine teilweise festgelegte direkt repräsentierte Variable anzugeben, können die Eigenschaften 1 und 2 nicht verwendet werden, und die Eigenschaften 3 und 4 können nur in Deklarationen von internen Variablen von Funktionsbausteinen und Programmen verwendet werden, wie in 2.5.2 bzw. 2.5.3 definiert.		

2.4.3.2 Anfangswert-Zuweisung

Die VAR...END_VAR Konstruktion, die in Tabelle 18 gezeigt ist, kann benutzt werden, um die Anfangswerte von direkt dargestellten Variablen oder symbolisch dargestellten Einzel- oder Multi-Element-Variablen darzustellen.

Anfangswerte können auch durch Anwenden der Instanz-spezifischen Initialisierung festgelegt werden, die durch das VAR_CONFIG...END_VAR Konstrukt geboten wird, das in 2.7.1 beschrieben wird (Tabelle 49, Eigenschaft 11). Instanz-spezifische Anfangswerte überschreiben immer die Typ-spezifischen Anfangswerte.

ANMERKUNG Der Gebrauch der Konstruktionen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT ist in 2.5 definiert.

Die 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 übrig gebliebenen Feldeinträge mit den voreingestellten Werten des zugehörigen Datentyps aufgefüllt werden. In beiden Fällen muss der Anwender wegen dieses Vorgangs während der Aufbereitung des Programms gewarnt 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 voreingestellten Anfangswerte besitzen, die für diese Elemente in der Datentyp-Deklaration deklariert sind.

Wenn eine Variable als *Funktionsbaustein-Instanz*, wie in 2.5.2.2 definiert, deklariert ist, können die Anfangswerte für die Eingangs- und die zugänglichen Variablen des Funktionsbausteins in einer eingeklammerten Liste deklariert werden, die auf den Zuweisungsoperator folgt, der dem Bezeichner des Funktionsbausteintyps folgt, wie es in Tabelle 18 gezeigt ist. Elemente, für die keine Anfangswerte in der Liste angegeben sind, müssen die voreingestellten Anfangswerte besitzen, die für diese Elemente in der Funktionsbaustein-Deklaration deklariert sind.

Tabelle 18 – Zuweisung von Anfangswerten für Variablen

Nr.	Eigenschaft/Beispiele	
1 ^a	Initialisierung von direkt dargestellten Variablen	
	<pre>VAR AT %QX5.1 : BOOL := 1; AT %MW6 : INT := 8; END_VAR</pre>	Boolescher Typ, Anfangswert = 1 Initialisiert ein Speicher-Wort mit der ganzen Zahl 8
2 ^a	Initialisierung von direkt dargestellten gepufferten Variablen	
	<pre>VAR RETAIN AT %QW5 : WORD := 16#FF00 ; END_VAR</pre>	Beim Kaltstart werden die 8 höchstwertigen Bits der 16-Bit-Folge des Ausgangs- worts 5 mit 1 und die 8 niederwertigen Bits mit 0 initialisiert.
3 ^a	Zuweisung von Speicherort und Anfangswert für symbolische Variablen	
	<pre>VAR VALVE_POS AT %QW28 : INT := 100; END_VAR</pre>	Weist das Ausgangswort 28 der ganzzahligen Variablen VALVE_POS mit einem Anfangswert 100 zu.
4 ^a	Zuweisung von Speicherort und Initialisierung für Feld	
	<pre>VAR OUTARY AT %QW6 : ARRAY [0..9] OF INT := [10(1)]; END_VAR</pre>	Deklariert ein Feld von 10 ganzen Zahlen, die dem zusammenhängenden Ausgangsbereich beginnend ab %QW6 jeweils mit einem Anfangswert von 1 zugeteilt werden.
5	Initialisierung von symbolischen Variablen	
	<pre>VAR MYBIT : BOOL := 1 ; OKAY : STRING[10] := 'OK'; END_VAR</pre>	Weist ein Speicher-Bit der booleschen Variablen MYBIT mit einem Anfangswert von 1 zu. Weist 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' /dezimal 79 bzw. 75), in einer für das Ausdrucken als Zeichenfolge geeigneten Anordnung.

Nr.	Eigenschaft/Beispiele	
6	Initialisierung für Feld	
	<pre> 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 </pre>	Weist 8 Speicher-Bits die Anfangswerte zu <pre> BITS[0] := 1, BITS[1] := 1, ..., BITS[6] := 0, BITS[7] := 0. </pre> Weist einem 2-mal-3 Feld TBT von ganzen Zahlen die Anfangswerte zu <pre> TBT[1,1] := 1, TBT[1,2] := 2, TBT[1,3] := 4, TBT[2,1] := 4, TBT[2,2] := 4, TBT[2,3] := 6. </pre>
7	Deklaration und Initialisierung für gepuffertes Feld	
	<pre> VAR RETAIN RTBT : ARRAY (1..2, 1..3) OF INT := [1, 2, 3(4)]; END_VAR </pre>	Deklariert das gepufferte Feld RTBT mit den „Kaltstart“-Anfangswerten <pre> RTBT[1,1] := 1, RTBT[1,2] := 2, RTBT[1,3] := 4, RTBT[2,1] := 4, RTBT[2,2] := 4, RTBT[2,3] := 0. </pre>
8	Initialisierung von strukturierten Variablen	
	<pre> VAR MODULE_8_CONF IG : ANALOG_16_INP UT_CONFIGURAT ION := (SIGNAL_TYPE := DIFFERENTIAL, CHANNEL := [4 ((RANGE := UNIPOLAR_1_5_ V)) , (RANGE := BIPOLAR_10_V, MIN_SCALE := 0, MAX_SCALE := 500)]) ; END_VAR </pre>	Initialisierung einer Variablen mit abgeleitetem Datentyp (siehe Tabelle 12) Dieses Beispiel veranschaulicht die Deklaration eines nicht-voreingestellten Werts für das fünfte Element des Feldes CHANNEL der Variablen MODULE_8_CONFIG.
9	Initialisierung von Konstanten	
	<pre> VAR CONSTANT PI : REAL := 3.141592 ; END_VAR </pre>	

Nr.	Eigenschaft/Beispiele	
10	Initialisierung von Funktionsbaustein-Instanzen	
	<pre> VAR TempLoop : PID := (ProbBand := 2.5, Integral := T#5S); END_VAR </pre>	Weist die Anfangswerte den Eingängen und Ausgängen einer Funktionsbaustein-Instanz zu.
^a 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.		

2.5 Programm-Organisationseinheiten

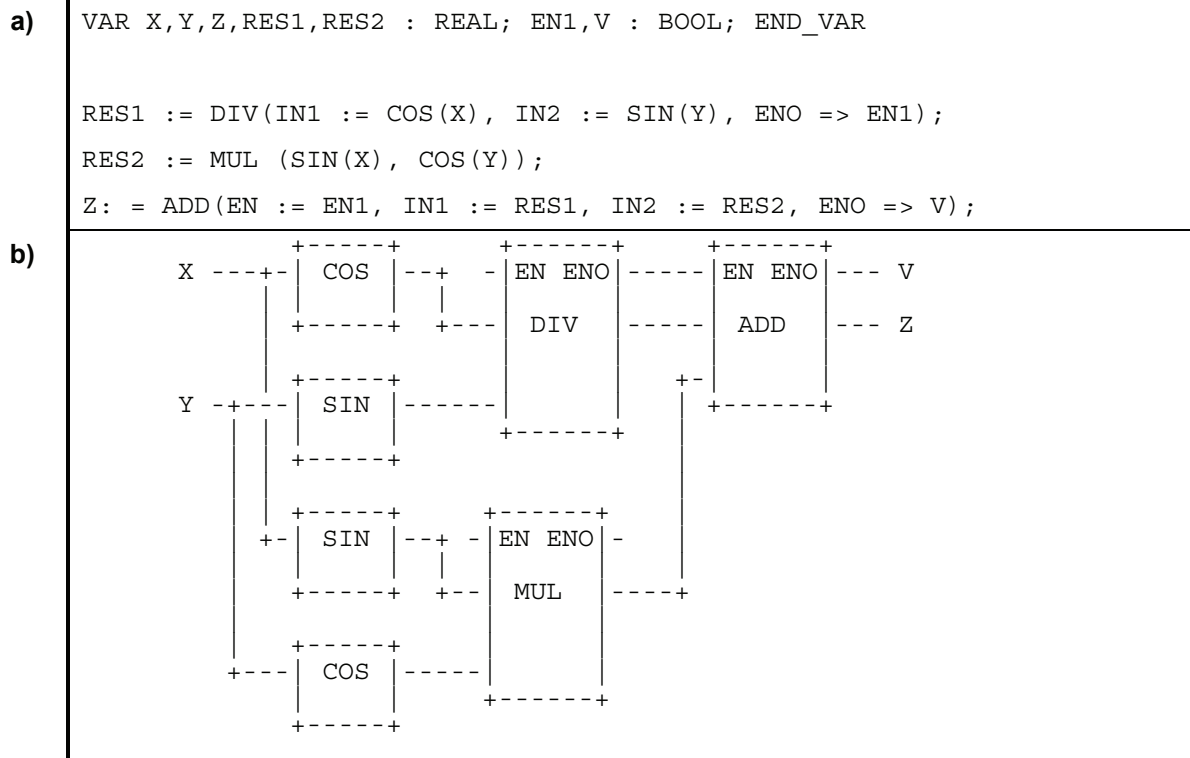
Die in diesem Teil von IEC 61131 definierten Programm-Organisationseinheiten sind die *Funktion*, der *Funktionsbaustein* und das *Programm*. Diese Programm-Organisationseinheiten können vom Hersteller geliefert werden oder vom Anwender mit den Mitteln programmiert werden, die in diesem Teil der Norm 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.

Die zur Bestimmung der Programm-Ausführungszeiten notwendige Information darf aus einem oder mehreren **implementierungsspezifischen Parametern** bestehen.

2.5.1 Funktionen

Für die Zwecke der SPS-Programmiersprachen ist eine *Funktion* als eine Programm-Organisationseinheit definiert, die bei Ausführung genau ein Datenelement, das als das Funktionsergebnis betrachtet werden kann, und beliebig viele zusätzliche Ausgangselemente (VAR_OUTPUT und VAR_IN_OUT) liefert. Das Funktionsergebnis kann wie bei jedem Datenelement mehrere Werte haben, z. B. ein Feld oder eine Struktur. Der Aufruf einer Funktion kann in Textsprachen als ein Operand in einem Ausdruck verwendet werden. Z. B. könnten die SIN- und COS-Funktionen, wie in Bild 4 gezeigt, angewendet werden.



a) Strukturierter Text (ST) – siehe 3.3

b) Funktionsbaustein Sprache (FBS) – siehe 4.3

ANMERKUNG Dieses Bild zeigt zwei verschiedene Darstellungen derselben Funktionalität. Es ist nicht erforderlich, eine automatische Umformung zwischen diesen zwei Darstellungsformen zu unterstützen.

Bild 4 – Beispiele für den Gebrauch von Funktionen

Funktionen dürfen keine internen Zustandsinformationen beinhalten; d. h. der Aufruf einer Funktion mit denselben Argumenten (Eingangsvariablen `VAR_INPUT` und Ein-Aus-Variablen `VAR_IN_OUT`) muss immer dieselben Werte (Ausgangsvariable `VAR_OUTPUT`, Ein-Aus-Variable `VAR_IN_OUT` und das Funktionsergebnis) liefern. Es muss ein **Fehler** sein, wenn externe Variable, wie in 2.4.3 definiert, eine Verletzung dieser Regel verursachen.

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 grafisch oder in Textform dargestellt werden.

In den Textsprachen, die in Abschnitt 3 dieser Norm definiert sind, muss der Aufruf von Funktionen nach den folgenden Regeln erfolgen:

- 1) Die Zuweisung von Eingangsargumenten muss den Regeln in Tabelle 19 a) folgen.
- 2) Die Zuweisungen von Ausgangsvariablen von Funktionen müssen entweder leer sein oder an Variable erfolgen.
- 3) Zuweisungen an `VAR_IN_OUT` Argumente müssen Variable sein.
- 4) Zuweisungen an `VAR_INPUT` Argumente dürfen leer sein (Eigenschaft 1 von Tabelle 19 a)) oder sind Konstante, Variable oder Funktionsaufrufe. In letzterem Fall wird das Funktionsergebnis als das aktuelle Argument verwendet.

In den grafischen Sprachen, die in Abschnitt 4 dieser Norm definiert sind, müssen Funktionen als grafische Schaltzeichen (Block) nach den folgenden Regeln dargestellt sein:

- 1) Die Form des Schaltzeichens muss rechteckig oder quadratisch sein.
- 2) Die Größe und das Seitenverhältnis 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 (Eingangsvariable auf der linken Seite und Ausgangsvariable 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 Eingangs- bzw. Ausgangsvariablen innen auf den linken bzw. rechten Seiten des Schaltzeichens erscheinen, wenn das Schaltzeichen Folgendes darstellt:
 - Eine der Standardfunktionen, die in Abschnitt 2.5.1.5 definiert sind, wenn die angegebene grafische Form die Variablennamen enthält oder
 - eine zusätzliche Funktion, die wie in 2.5.1.3 festgelegt ist.

Dieser Gebrauch unterliegt den folgenden Bedingungen:

- a) Wenn keine Namen für die Eingangsvariablen in Standardfunktionen angegeben sind, müssen die voreingestellten Namen `IN1`, `IN2`, ... von oben nach unten gelten.
- b) Wenn eine Standardfunktion einen einzelnen Eingang ohne Namen hat, muss der voreingestellte Name `IN` gelten.
- c) Die voreingestellten Namen, die oben beschrieben sind, können, aber müssen nicht links innerhalb der grafischen Darstellung erscheinen.
- 6) Es darf ein zusätzlicher Eingang `EN` und/oder Ausgang `ENO`, wie in 2.5.1.2 festgelegt, verwendet werden. Sie müssen, falls vorhanden, an den obersten Positionen auf der linken bzw. rechten Seite des Schaltzeichens gezeigt werden.
- 7) Das Funktionsergebnis muss an der obersten Position auf der rechten Seite des Schaltzeichens gezeigt werden, ausgenommen, es gibt einen `ENO` Ausgang. Dann muss das Funktionsergebnis an der nächsten Position unter dem `ENO` Ausgang gezeigt werden. Da der Name der Funktion für die Zuweisung ihrer Ausgangswerte, wie in 2.5.1.3 festgelegt, verwendet wird, darf kein Ausgangsvariablen-Name auf der rechten Seite des Schaltzeichens gezeigt werden.
- 8) Die Verbindungen der Argumente (einschließlich des Funktionsergebnisses) müssen durch Signalfusslinien angezeigt werden.
- 9) Die Negation von booleschen Signalen muss durch Angabe eines offenen Kreises direkt außerhalb der Verbindung der Eingangs- oder Ausgangslinie mit dem Schaltzeichen gezeigt werden. Dies muss in dem Zeichensatz, der in 2.1.1 definiert ist, durch den Großbuchstaben "O" dargestellt werden, wie in Tabelle 19 gezeigt ist.
- 10) Alle Eingänge und Ausgänge (einschließlich dem Funktionsergebnis) einer grafisch dargestellten Funktion müssen als eine einfache Linie außen an der entsprechenden Seite des Schaltzeichens dargestellt werden; das gilt auch, wenn das Datenelement ein Mehrfach-Element ist.
- 11) Funktionsergebnisse und Funktionsausgänge (`VAR_OUTPUT`) können mit einer Variablen verbunden werden, als Eingang mit anderen Funktionsbausteinen oder Funktionen verbunden werden oder unverbunden bleiben.
- 12) Es muss ein **Fehler** sein, wenn eine `VAR_IN_OUT` Variable eines Funktionsbausteinaufrufs oder eines Funktionsaufrufs in einer POE nicht „korrekt abgebildet“ ist. Eine `VAR_IN_OUT` Variable ist „korrekt abgebildet“, wenn sie grafisch auf der linken Seite mit einer Variablen verbunden ist oder bei einem Aufruf in Textsprache unter Verwendung des „:=“ Operators einer Variablen zugewiesen ist, die in einem `VAR_IN_OUT`, `VAR`, `VAR_OUT` oder `VAR_EXTERNAL` Block (ohne das Bestimmungszeichen `CONSTANT`) der enthaltenden Programm-Organisationseinheit deklariert ist, oder wenn sie mit einer „korrekt abgebildeten“ `VAR_IN_OUT` Variablen einer anderen Funktionsbaustein-Instanz oder eines anderen Funktionsaufrufs verbunden ist.
- 13) Eine „korrekt abgebildete“ (siehe oben Regel 12) `VAR_IN_OUT` Variable einer Funktionsbaustein-Instanz oder eines Funktionsaufrufs kann grafisch rechts mit einer Variablen verbunden sein oder unter Verwendung des „:=“ Operators in einer Zuweisung in Textsprache einer Variablen zugewiesen sein, die in einem `VAR`, `VAR_OUTPUT` oder `VAR_EXTERNAL` Block einer enthaltenden Programm-Organisations-

einheit deklariert ist. Es muss ein **Fehler** sein, wenn eine solche Verbindung zu einem mehrdeutigen Wert der so verbundenen Variablen führt.

Tabelle 19 – Grafische Negation von booleschen Signalen

Nr.	Eigenschaft ^{a, b}	Darstellung
1	Negierter Eingang	<pre> +----+ ---O --- +----+ </pre>
2	Negierter Ausgang	<pre> +----+ --- O--- +----+ </pre>
^a Falls eine dieser Eigenschaften für Funktionen unterstützt wird, muss sie auch für Funktionsbausteine, die in 2.5.2 definiert sind, unterstützt werden und umgekehrt. ^b Der Gebrauch dieser Konstrukte ist für Ein-Aus-Variable verboten.		

Bild 5 veranschaulicht sowohl den grafischen als auch den gleichbedeutenden Gebrauch in Textform von Funktionen, einschließlich des Gebrauchs einer Standardfunktion (ADD), ohne definierte formale Argument-Namen; es veranschaulicht eine Standardfunktion (SHL) mit definierten Argumentnamen, die gleiche Funktion mit zusätzlichem Gebrauch von EN Eingang und negiertem ENO Ausgang und eine anwender-definierte Funktion (INC) mit definierten formalen Argumentnamen.

Beispiel	Erläuterung
<pre> +-----+ ADD B--- ---A C--- D--- +-----+ </pre>	Grafischer Gebrauch der ADD Funktion (Siehe 2.5.1.5.2) (FBS Sprache; siehe 4.3) (Keine formalen Variablennamen)
A := ADD(B, C, D);	Textueller Gebrauch der ADD Funktion (ST Sprache; siehe 3.3)
<pre> +-----+ SHL B--- IN ---A C--- N +-----+ </pre>	Grafischer Gebrauch der SHL Funktion (Siehe 2.5.1.5.3) (FBS Sprache; siehe 4.3) (Formale Argumentnamen)
A := SHL(IN := B, N := C);	Textueller Gebrauch der SHL Funktion (ST Sprache; siehe 3.3)
<pre> +-----+ SHL ENABLE--- EN ENO O---NO_ERR B--- IN ---A C--- N +-----+ </pre>	Grafischer Gebrauch der SHL Funktion (Siehe 2.5.1.5.3) (FBS Sprache; siehe 4.3) (Formale Argumentnamen; Gebrauch von EN Eingang und negiertem ENO Ausgang)
A := SHL(EN:=ENABLE, IN:=B, N:=C, NOT ENO => NO_ERR);	Gebrauch der SHL Funktion in Textform (ST Sprache; siehe 3.3)
<pre> +-----+ INC X--- V---V ---X +-----+ </pre>	Grafischer Gebrauch der anwender-definierten INC Funktion (FBD Sprache, siehe 4.3) (Formale Argumentnamen für VAR_IN_OUT)
A := INC(V := X);	Gebrauch der INC Funktion in Textform (ST Sprache, siehe 3.3)

Bild 5 – Gebrauch von formalen Argumentnamen

Eigenschaften für den Aufruf von Funktionen in Textform sind in Tabelle 19 a) definiert. Der Aufruf einer Funktion in Textform muss aus den Funktionsnamen, gefolgt von einer Liste von Argumenten bestehen. In