

DIN EN 61131-3

ICS 35.240.50; 35.060; 25.040.40; 35.080

Ersatz für
DIN EN 61131-3:2003-12
Siehe Anwendungsbeginn

**Speicherprogrammierbare Steuerungen –
Teil 3: Programmiersprachen
(IEC 61131-3:2013);
Deutsche Fassung EN 61131-3:2013**

Programmable controllers –
Part 3: Programming languages
(IEC 61131-3:2013);
German version EN 61131-3:2013

Automates programmables –
Partie 3: Langages de programmation
(CEI 61131-3:2013);
Version allemande EN 61131-3:2013

Gesamtumfang 255 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2013-03-27 angenommene Europäische Norm als DIN-Norm ist 2014-06-01.

Für DIN EN 61131-3:2003-12 besteht eine Übergangsfrist bis 2016-03-27.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN IEC 61131-3:2009-12.

Für dieses Dokument ist das nationale Arbeitsgremium K 962 „SPS“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (www.dke.de) zuständig.

Die enthaltene IEC-Publikation wurde vom TC 65 „Industrial-process measurement and control“ erarbeitet.

Das IEC-Komitee hat entschieden, dass der Inhalt dieser Publikation bis zu dem Datum (stability date) unverändert bleiben soll, das auf der IEC-Website unter „<http://webstore.iec.ch>“ zu dieser Publikation angegeben ist. Zu diesem Zeitpunkt wird entsprechend der Entscheidung des Komitees die Publikation

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

Änderungen

Gegenüber DIN EN 61131-3:2003-12 wurden folgende Änderungen vorgenommen:

- a) EN 61131-3:2013 ist eine kompatible Erweiterung der EN 61131-3:2003. Die hauptsächlichen Erweiterungen sind: neue Datentypen und Umwandlungsfunktionen, Verweisungen, Namensräume und objekt-orientierte Eigenschaften der Klassen und Funktionsbausteine. Siehe Anhang B.

Frühere Ausgaben

DIN 19239: 1983-05

DIN EN 61131-3: 1994-08, 2003-12

Nationaler Anhang NA (informativ)

Zusammenhang mit Europäischen und Internationalen Normen

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf eine Norm oder andere Unterlage 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 oder anderen Unterlage.

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

Eine Information über den Zusammenhang der zitierten Normen und anderen Unterlagen mit den entsprechenden Deutschen Normen und anderen Unterlagen ist in Tabelle NA.1 wiedergegeben.

Tabelle NA.1

Europäische Norm	Internationale Norm	Deutsche Norm	Klassifikation im VDE-Vorschriftenwerk
–	IEC 60050 (alle Teile)	^a	–
–	ISO/IEC/IEEE 60559	–	–
–	IEC 60617	–	–
EN 60848	IEC 60848	DIN EN 60848	–
EN 61131-1	IEC 61131-1	DIN EN 61131-1	–
EN 61131-5	IEC 61131-5	DIN EN 61131-5	–
EN 61499	IEC 61499	DIN EN 61499	–
–	ISO/IEC 10646:2012	–	–
–	ISO/IEC 14977:1996	–	–
–	ISO/AFNOR:1989	–	–
^a „Internationales Elektrotechnisches Wörterbuch – Deutsche Ausgabe“, Online-Zugang: http://www.dke.de/dke-iev .			

Nationaler Anhang NB (informativ)

Literaturhinweise

DIN EN 60848, *GRAFCET, Spezifikationssprache für Funktionspläne der Ablaufsteuerung*

DIN EN 61131-1, *Speicherprogrammierbare Steuerungen – Teil 1: Allgemeine Informationen*

DIN EN 61131-5, *Speicherprogrammierbare Steuerungen – Teil 5: Kommunikation*

DIN EN 61499, *Funktionsbausteine für industrielle Leitsysteme*

– Leerseite –

Speicherprogrammierbare Steuerungen –
Teil 3: Programmiersprachen
(IEC 61131-3:2013)

Programmable controllers –
Part 3: Programming languages
(IEC 61131-3:2013)

Automates programmables –
Partie 3: Langages de programmation
(CEI 61131-3:2013)

Diese Europäische Norm wurde von CENELEC am 2013-03-27 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 CEN-CENELEC Management Centre 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 CEN-CENELEC Management Centre mitgeteilt worden ist, hat den gleichen Status wie die offiziellen Fassungen.

CENELEC-Mitglieder sind die nationalen elektrotechnischen Komitees von Belgien, Bulgarien, Dänemark, Deutschland, der ehemaligen jugoslawischen Republik Mazedonien, Estland, Finnland, Frankreich, Griechenland, Irland, Island, Italien, Kroatien, Lettland, Litauen, Luxemburg, Malta, den Niederlanden, Norwegen, Österreich, Polen, Portugal, Rumänien, Schweden, der Schweiz, der Slowakei, Slowenien, Spanien, der Tschechischen Republik, der Türkei, Ungarn, dem Vereinigten Königreich und Zypern.

CENELEC

Europäisches Komitee für Elektrotechnische Normung
European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique

Management Centre: Avenue Marnix 17, B-1000 Brüssel

© 2013 CENELEC – Alle Rechte der Verwertung, gleich in welcher Form und in welchem Verfahren, sind weltweit den Mitgliedern von CENELEC vorbehalten.

Ref. Nr. EN 61131-3:2013 D

Vorwort

Der Text des Dokumentes 65B/858/FDIS, zukünftige Ausgabe 3 der IEC 61131-3, erarbeitet durch IEC/TC 65 „Industrial-process measurement, control and automation“, wurde zur parallelen IEC-CENELEC-Abstimmung vorgelegt und von CENELEC als EN 61131-3:2013 angenommen.

Nachstehende Daten wurden festgelegt:

- spätestes Datum, zu dem dieses Dokument auf nationaler Ebene durch Veröffentlichung einer identischen nationalen Norm oder durch Anerkennung übernommen werden muss (dop): 2013-12-27
- spätestes Datum, zu dem nationale Normen, die diesem Dokument entgegenstehen, zurückgezogen werden müssen (dow): 2016-03-27

Dieses Dokument ersetzt EN 61131-3:2003.

EN 61131-3:2013 enthält die folgenden wesentlichen technischen Änderungen gegenüber EN 61131-3:2003:

EN 61131-3:2013 ist eine kompatible Erweiterung der EN 61131-3:2003. Die hauptsächlichen Erweiterungen sind: neue Datentypen und Umwandlungsfunktionen, Verweisungen, Namensräume und objektorientierte Eigenschaften der Klassen und Funktionsbausteine. Siehe Anhang B.

Es wird auf die Möglichkeit hingewiesen, dass einige Texte dieses Dokuments Patentrechte berühren können. CENELEC [und/oder CEN] sind nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

Anerkennungsnotiz

Der Text der Internationalen Norm IEC 61131-3:2013 wurde vom CENELEC ohne irgendeine Abänderung als eine Europäische Norm angenommen.

In der offiziellen Fassung sind unter „Literaturhinweise“ zu den aufgelisteten Normen die nachstehenden Anmerkungen einzutragen:

IEC 60848 ANMERKUNG Harmonisiert als EN 60848.

IEC 61499 (alle Teile) ANMERKUNG Harmonisiert in der Reihe EN 61499.

Inhalt

	Seite
Vorwort	2
1 Anwendungsbereich	7
2 Normative Verweisungen	7
3 Begriffe	7
4 Architekturmodelle	17
4.1 Software-Modell	17
4.2 Kommunikationsmodell	19
4.3 Programmiermodell	20
5 Normerfüllung	22
5.1 Allgemeines	22
5.2 Eigenschaftentabelle	23
5.3 Erfüllungsaussage des Herstellers	23
6 Gemeinsame Elemente	25
6.1 Gebrauch der gedruckten Zeichen	25
6.2 Pragma	27
6.3 Literale – Externe Darstellung von Daten	27
6.4 Datentypen	33
6.5 Variablen	50
6.6 Programm-Organisationseinheiten (POEs)	62
6.7 Elemente der Ablaufsprache (AS)	169
6.8 Konfigurationselemente	195
6.9 Namensraum	206
7 Textsprachen	215
7.1 Gemeinsame Elemente	215
7.2 Anweisungsliste (AWL)	215
7.3 Strukturierter Text (ST)	221
8 Grafische Sprachen	230
8.1 Gemeinsame Elemente	230
8.2 Kontaktplan (KOP)	237
8.3 Funktionsbaustein-Sprache (FBS)	241
Anhang A (normativ) Formale Festlegung der Sprachelemente	242
Anhang B (informativ) Liste der wichtigen Änderungen und Erweiterungen der dritten Ausgabe	249
Literaturhinweise	250
Anhang ZA (normativ) Normative Verweisungen auf internationale Publikationen mit ihren entsprechenden europäischen Publikationen	251
Bilder	
Bild 1 – Software-Modell	18
Bild 2 – Kommunikationsmodell	20

	Seite
Bild 3 – Kombination der SPS-Sprachelemente	22
Bild 4 – Erfüllungsaussage des Herstellers (Beispiel)	24
Bild 5 – Hierarchie der allgemeinen Datentypen	36
Bild 6 – Initialisierung durch Literale und konstante Ausdrücke (Regeln)	37
Bild 7 – Schlüsselwörter für Variablen-Deklaration (Zusammenfassung)	54
Bild 8 – Gebrauch von <code>VAR_GLOBAL</code> , <code>VAR_EXTERNAL</code> und <code>CONSTANT</code> (Regeln)	55
Bild 9 – Bedingungen für den Anfangswert einer Variablen (Regeln)	61
Bild 10 – Formale und nicht formale Aufruf-Darstellung (Beispiele)	68
Bild 11 – Datentypkonvertierungsregeln – implizit und/oder explizit (Zusammenfassung)	72
Bild 12 – Unterstützte implizite Typkonvertierungen	73
Bild 13 – Gebrauch der Eingangs- und Ausgangsparameter (Regeln)	119
Bild 14 – Gebrauch der Eingangs- und Ausgangsparameter des Funktionsbausteins (Veranschaulichung der Regeln)	120
Bild 15 – Standardfunktionsbausteine Zeitgeber – Zeitdiagramme (Regeln)	129
Bild 16 – Vererbung und Interface-Implementierung – Überblick	132
Bild 17 – Vererbung von Klassen (Veranschaulichung)	142
Bild 18 – Interface mit abgeleiteten Klassen (Veranschaulichung)	152
Bild 19 – Vererbung von Interface und Klasse (Veranschaulichung)	157
Bild 20 – Funktionsbaustein mit optionalem Rumpf und Methoden (Veranschaulichung)	163
Bild 21 – Vererbung des Funktionsbaustein-Rumpfs mit <code>SUPER()</code> (Beispiel)	165
Bild 22 – Funktionsbaustein <code>ACTION_CONTROL</code> – Externe Schnittstelle (Zusammenfassung)	180
Bild 23 – <code>ACTION_CONTROL</code> -Funktionsbaustein-Rumpf (Zusammenfassung)	182
Bild 24 – Aktionssteuerung (Beispiel)	184
Bild 25 – AS-Ablaufregeln	193
Bild 26 – AS-Fehler (Beispiele)	194
Bild 27 – Konfiguration (Beispiel)	197
Bild 28 – Deklaration von <code>CONFIGURATION</code> und <code>RESOURCE</code> (Beispiel)	200
Bild 29 – Zugriffsmöglichkeiten bei Verwendung des Namensraums (Regeln)	209
Bild 30 – Gemeinsame textuelle Elemente (Zusammenfassung)	215

Tabellen

Tabelle 1 – Zeichensatz	25
Tabelle 2 – Bezeichner	25
Tabelle 3 – Kommentare	27
Tabelle 4 – Pragma	27
Tabelle 5 – Numerische Literale	29
Tabelle 6 – Zeichenfolge-Literale	30
Tabelle 7 – Zwei-Zeichen-Kombinationen in Zeichenfolgen	31
Tabelle 8 – Zeitdauer-Literale	32
Tabelle 9 – Datums- und Uhrzeitliterale	32

	Seite
Tabelle 10 – Elementare Datentypen	33
Tabelle 11 – Deklaration von anwenderdefinierten Datentypen und Initialisierung	38
Tabelle 12 – Operationen der Referenzierung	50
Tabelle 13 – Deklaration von Variablen	52
Tabelle 14 – Initialisierung von Variablen	53
Tabelle 15 – Variabel lange <code>ARRAY</code> -Variable	56
Tabelle 16 – Direkt dargestellte Variablen	59
Tabelle 17 – Teilweiser Zugriff auf <code>ANY_BIT</code> -Variablen	65
Tabelle 18 – Ausführungssteuerung mit <code>EN</code> und <code>ENO</code>	70
Tabelle 19 – Funktionsdeklaration	77
Tabelle 20 – Funktionsaufruf	79
Tabelle 21 – Typisierte und überladene Funktionen	82
Tabelle 22 – Funktionen zur Datentypkonvertierung	85
Tabelle 23 – Datenkonvertierung von numerischen Datentypen	86
Tabelle 24 – Datenkonvertierung von Bit-Datentypen	89
Tabelle 25 – Datentypkonvertierung von Bit-Typen in numerische Typen	90
Tabelle 26 – Datentypkonvertierung der Datentypen für Datum und Zeit	91
Tabelle 27 – Datentypkonvertierung von Zeichentypen	92
Tabelle 28 – Numerische und arithmetische Funktionen	93
Tabelle 29 – Arithmetische Funktionen	94
Tabelle 30 – Bitschiebe-Funktionen	95
Tabelle 31 – Bitweise boolesche Funktionen	95
Tabelle 32 – Auswahlfunktionen ^d	97
Tabelle 33 – Vergleichsfunktionen	98
Tabelle 34 – Funktionen für Zeichenfolgen	99
Tabelle 35 – Numerischen Funktionen mit Zeit und Dauer	101
Tabelle 36 – Zusätzliche Funktionen für Zeitdatentypen <code>CONCAT</code> und <code>SPLIT</code>	102
Tabelle 37 – Funktionen zur Konvertierung der Bytereihenfolge (en: endianness conversion functions)	106
Tabelle 38 – Funktionen für Aufzählungsdantentypen	106
Tabelle 39 – Validierungsfunktionen	107
Tabelle 40 – Deklaration des Funktionsbaustein-Typs	109
Tabelle 41 – Deklaration der Funktionsbaustein-Instanz	113
Tabelle 42 – Aufruf des Funktionsbausteins	114
Tabelle 43 – Bistabile Standardfunktionsbausteine ^a	124
Tabelle 44 – Standardfunktionsbausteine zur Flankenerkennung	125
Tabelle 45 – Standardfunktionsbausteine für Zähler	126
Tabelle 46 – Standardfunktionsbausteine Zeitgeber	128
Tabelle 47 – Programmdeklaration	130

	Seite
Tabelle 48 – Klasse	133
Tabelle 49 – Deklaration der Klasseninstanz	136
Tabelle 50 – Textueller Methodenaufruf – Formale und nicht formale Parameterliste	140
Tabelle 51 – Interface	151
Tabelle 52 – Zuweisungsversuch	160
Tabelle 53 – Objektorientierter Funktionsbaustein	161
Tabelle 54 – AS-Schritt.....	171
Tabelle 55 – Transition und Transitionsbedingung der AS	173
Tabelle 56 – AS-Deklaration von Aktionen	175
Tabelle 57 – Verknüpfung Schritt/Aktion	177
Tabelle 58 – Aktionsblock.....	178
Tabelle 59 – Aktionsbestimmungszeichen	179
Tabelle 60 – Eigenschaften der Aktionssteuerung	184
Tabelle 61 – Kettenablauf – grafisch	186
Tabelle 62 – Deklaration von Konfiguration und Ressource	198
Tabelle 63 – Task	202
Tabelle 64 – Namensraum	211
Tabelle 65 – Möglichkeiten der geschachtelten Namensraumdeklaration	212
Tabelle 66 – Namensraum-Direktive <code>USING</code>	214
Tabelle 67 – Geklammerter Ausdruck für AWL-Sprache	217
Tabelle 68 – Operatoren der AWL-Sprache	217
Tabelle 69 – Aufrufe für AWL-Sprache.....	219
Tabelle 70 – Operatoren von Standardfunktionsbausteinen für AWL-Sprache	221
Tabelle 71 – Operatoren der Sprache ST	223
Tabelle 72 – Anweisungen der Sprache ST	224
Tabelle 73 – Grafische Elemente der Ausführungssteuerung.....	236
Tabelle 74 – Stromschienen und Verbindungselemente.....	238
Tabelle 75 – Kontakte	238
Tabelle 76 – Spulen.....	240

1 Anwendungsbereich

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

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

Dieser Teil der IEC 61131 legt die Syntax und Semantik einer vereinheitlichten Reihe von Programmiersprachen für speicherprogrammierbare Steuerungen (SPS) fest. Diese Reihe umfasst zwei Textsprachen, Anweisungsliste (AWL) und Strukturierter Text (ST) und zwei grafische Sprachen, Kontaktplan (KOP) und Funktionsbaustein-Sprache (FBS).

Ein zusätzlicher Satz von grafischen und gleichwertigen textuellen Elementen, der Ablaufsprache (AS) genannt wird, ist 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.

2 Normative Verweisungen

Die folgenden Dokumente, die in diesem Dokument teilweise oder als Ganzes zitiert werden, 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 61131-1, *Programmable controllers – Part 1: General information*

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

ISO/IEC 10646:2012, *Information technology – Universal Coded Character Set (UCS)*

ISO/IEC/IEEE 60559, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

3 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC 61131-1 und die folgenden Begriffe.

3.1

absolute Zeit

(en: absolute time)

Kombination von Tageszeit und Datum

3.2

Zugriffspfad

(en: access path)

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

3.3

Aktion

(en: action)

boolesche Variable oder eine Sammlung von Operationen, die zusammen mit einer zugehörigen Kontrollstruktur ausgeführt wird

3.4

Aktionsblock

(en: 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

3.5

Aggregat

(en: aggregate)

strukturierte Sammlung von Datenobjekten, die einen Datentyp bilden

[QUELLE: ISO/AFNOR: 1989]

3.6

Feld

(en: array)

Aggregat, das aus Datenobjekten mit identischen Attributen besteht, von denen jedes eindeutig durch Indizieren referenziert werden kann

[QUELLE: ISO/AFNOR:1989]

3.7

Zuweisung

(en: assignment)

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

[QUELLE: ISO/AFNOR:1989]

3.8

Basis-Typ

(en: base type)

Datentyp, Funktionsbaustein-Typ oder Klasse, von dem/der weitere Typen vererbt/abgeleitet sind

3.9

basisbezogene Zahl

(en: based number)

Zahl, die mit einer festgelegten Basis ungleich zehn dargestellt ist

3.10

binär kodierte Dezimalzahl

BCD

(en: binary coded decimal)

Kodierung von Dezimalzahlen, bei der jede Ziffer durch ihre eigene Binärfolge dargestellt ist

3.11

bistabiler Funktionsbaustein

(en: bistable function block)

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

3.12

Bitfolge

(en: bit string)

Datenelement, bestehend aus einem oder mehreren Bits

3.13

Bitfolge-Literal

(en: bit string literal)

Literal, das direkt einen Bitfolge-Wert vom Type `BOOL`, `BYTE`, `WORD`, `DWORD` oder `LWORD` darstellt

3.14

Rumpf

(en: body)

Menge von Operationen der Programmorganisationseinheit

3.15

Aufruf

(en: call)

Sprachkonstrukt, das die Ausführung einer Funktion, eines Funktionsbausteins oder einer Methode bewirkt

3.16

Zeichenfolge

(en: character string)

Aggregat, das aus einer geordneten Folge von Zeichen besteht

3.17

Zeichenfolge-Literal

(en: character string literal)

Literal, das direkt einen Zeichen- oder ein Zeichenfolge-Wert vom Typ CHAR, WCHAR, STRING oder WSTRING darstellt

3.18

Klasse

(en: class)

Programm-Organisationseinheit, bestehend aus:

- der Definition einer Datenstruktur,
- einer Menge von Methoden, die auf der Datenstruktur ausgeführt werden

3.19

Kommentar

(en: comment)

Sprachkonstrukt zur Einbindung von Text, ohne Auswirkung auf die Ausführung des Programmkodes zu haben

[QUELLE: ISO/AFNOR:1989]

3.20

Konfiguration

(en: configuration)

Sprachelement, das einem SPS-System entspricht

3.21

Konstante

(en: constant)

Sprachelement, das ein Datenelement mit einen festen Wert deklariert

3.22

Zähler-Baustein

(en: counter function block)

Funktionsbaustein, der einen Wert für die Anzahl der erkannten Wechsel an einem oder mehreren festgelegten Eingängen akkumuliert

3.23

Datentyp

(en: data type)

Menge von Werten zusammen mit der Menge von zulässigen Operationen

[QUELLE: ISO/AFNOR:1989]

3.24

Datum und Uhrzeit

(en: date and time)

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

3.25

Deklaration

(en: declaration)

Mechanismus zur Festlegung der Definition eines Sprachelements

3.26

Begrenzungszeichen

(en: delimiter)

Zeichen oder Zeichenkombination, das zur Trennung von Sprachelementen verwendet wird

3.27

abgeleitete Klasse

(en: derived class)

Klasse, die durch Vererbung von einer anderen Klasse erzeugt ist

Anmerkung 1 zum Begriff: Eine abgeleitete Klasse wird auch erweiterte Klasse oder Kind-Klasse genannt.

3.28

abgeleiteter Datentyp

(en: derived datatype)

Datentyp, der durch Verwenden eines anderen Datentyp erzeugt ist

3.29

abgeleiteter Funktionsbaustein-Typ

(en: derived function block type)

Funktionsbaustein-Typ, der durch Vererbung von einem anderen Funktionsbaustein-Typ erzeugt wird

3.30

direkte Darstellung

(en: direct representation)

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

3.31

Doppelwort

(en: double word)

Datenelement, das 32 Bits enthält

3.32

dynamisches Binden

(en: dynamic binding)

Situation, in der die Instanz eines Methodenaufrufs entsprechend des aktuellen Typs einer Instanz oder eines Interface während der Laufzeit ermittelt wird

3.33

Auswertung

(en: evaluation)

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

3.34

Element zur Ausführungssteuerung

(en: execution control element)

Sprachelement, das den Fluss der Programmausführung steuert

3.35

fallende Flanke

(en: falling edge)

Wechsel von 1 nach 0 bei einer booleschen Variablen

3.36

Funktion

(en: function)

Sprachelement, das beim Ausführen typischerweise ein Ergebnis mit einem Datenelement und möglicherweise zusätzliche Ausgangsvariablen liefert

3.37

Funktionsbaustein-Instanz

(en: function block instance)

Instanz eines Funktionsbaustein-Typs

3.38

Funktionsbaustein-Typ

(en: function block type)

Sprachelement, bestehend aus:

- der Definition einer Datenstruktur, aufgeteilt in Eingangs-, Ausgangs- und interne Variablen, und
- einer Menge von Operationen oder einer Menge von Methoden, die mit den Elementen der Datenstruktur durchgeführt werden, wenn eine Instanz des Funktionsbaustein-Typs aufgerufen wird

3.39

Funktionsbaustein-Sprache

(en: function block diagram)

Netzwerk, in dem die Knoten Funktionsbaustein-Instanzen, grafisch dargestellte Funktionen oder Methodenaufrufe, Variablen, Literale und Marken sind

3.40

allgemeiner Datentyp

(en: generic data type)

Datentyp, der mehr als einen Typ darstellt

3.41

globale Variable

(en: global variable)

Variable, deren Anwendungsbereich global ist

3.42

hierarchische Adressierung

(en: hierarchical addressing)

direkte Darstellung eines Datenelements als ein Teil einer physikalischen oder logischen Hierarchie

BEISPIEL Ein Anschlusspunkt in einer Baugruppe, die in einem Baugruppenträger steckt, der wiederum in einem Schrank eingebaut ist

3.43

Bezeichner

(en: identifier)

Kombination von Buchstaben, Zahlen und Unterstrich-Zeichen, die mit einem Buchstaben oder einem Unterstrich beginnt und ein Sprachelement bezeichnet

3.44

Implementierung

(en: implementation)

Produktversion einer SPS oder einer Programmier- und Testeinrichtung, die vom Hersteller zur Verfügung gestellt wird

3.45

Hersteller

(en: implementer)

Lieferant der SPS oder der Programmier- und Testeinrichtung, die dem Anwender zum Programmieren einer SPS-Anwendung zur Verfügung gestellt wird

3.46

Vererbung

(en: inheritance)

Erzeugung einer neuen Klasse, eines neuen Funktionsbaustein-Typs oder eines neuen Interface, basierend auf einer vorhandenen Klasse, einem vorhandenen Funktionsbaustein-Typ bzw. Interface

3.47

Anfangswert

(en: initial value)

Wert, der einer Variablen beim Systemanlauf zugewiesen wird

3.48

Ein-Aus-Variable

(en: in-out variable)

Variable, die benutzt wird, um einer Programm-Organisationseinheit einen Wert zu übergeben, und die zusätzlich verwendet wird, um von der Programm-Organisationseinheit einen Wert zurückzugeben

3.49

Eingangsparameter

(en: input parameter, input)

Variable, die benutzt wird, um einer Programm-Organisationseinheit, ausgenommen der Klasse, einen Wert zu übergeben

3.50

Instanz

(en: instance)

individuelles, benanntes Exemplar einer Datenstruktur, verknüpft mit einem Funktionsbaustein-Typ, einer Klasse oder einem Programm-Typ, das ihre Werte von einem Aufruf der zugehörigen Operationen bis zum nächsten behält

3.51

Instanz-Name

(en: instance name)

Bezeichner, verknüpft mit einer festgelegten Instanz

3.52

Instanziierung

(en: instantiation)

Erzeugung einer Instanz

3.53

ganze Zahl

(en: integer)

natürliche Zahl, die positive, Null- und negative Werte enthalten kann

3.54

ganzzahliges Literal

(en: integer literal)

Literal, das direkt den Wert einer ganzen Zahl darstellt

3.55

Interface

(en: interface)

Sprachelement im Zusammenhang mit der objektorientierten Programmierung, das eine Menge von Methoden-Prototypen enthält

3.56

Schlüsselwort

(en: keyword)

lexikalische Einheit, die ein Sprachelement charakterisiert

3.57

Marke

(en: label)

Sprachkonstrukt, das eine Anweisung, ein Netzwerk oder eine Gruppe von Netzwerken bezeichnet und einen Bezeichner einschließt

3.58

Sprachelement

(en: language element)

Element, das durch ein Symbol auf der linken Seite der Produktionsregeln in der formalen Spezifikation identifiziert wird

3.59

Literal

(en: literal)

lexikalische Einheit, die direkt einen Wert darstellt

[QUELLE: ISO/AFNOR:1989]

3.60

logischer Speicherort

(en: 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

3.61

lange Realzahl

(en: long real)

Realzahl, die in einem Langwort dargestellt ist

3.62

Langwort

(en: long word)

64-Bit-Datenelement

3.63

Methode

(en: method)

Sprachelement ähnlich einer Funktion, das nur im Anwendungsbereich eines Funktionsbaustein-Typs definiert werden kann und das impliziten Zugriff auf Variablen der Funktionsbaustein-Instanz oder Klasseninstanz hat

3.64

Methoden-Prototyp

(en: method prototype)

Sprachelement, das nur die Signatur einer Methode enthält

3.65

bezeichnetes Element

(en: named element)

Element einer Struktur, das durch seinen zugehörigen Bezeichner benannt ist

3.66

Netzwerk

(en: network)

Anordnung von Knoten und miteinander verbundenen Zweigen

3.67

numerisches Literal

(en: numeric literal)

Literal, das direkt einen numerischen Wert darstellt, d. h. ein ganzzahliges Literal oder ein Real-Literal

3.68

Operation

(en: operation)

Sprachelement, das eine elementare Funktionalität darstellt, die zu einer Programm-Organisationseinheit oder Methode gehört

3.69

Operand

(en: operand)

Sprachelement, mit dem eine Operation durchgeführt wird

3.70

Operator

(en: operator)

Symbol, das die Aktion darstellt, die in einer Operation auszuführen ist

3.71

override

(en: override)

Schlüsselwort, das mit einer Methode in einer abgeleiteten Klasse oder in einem Funktionsbaustein-Typ für eine Methode mit der gleichen Signatur und wie eine Methode der Basis-Klasse oder des Funktionsbaustein-Typs und mit einem neuen Methodenrumpf verwendet wird

3.72

AusgangsvARIABLE

(en: output variable, output)

Variable, die benutzt wird, um einen Wert von der Programm-Organisationseinheit, ausgenommen der Klasse, zurückzugeben

3.73

Parameter

(en: parameter)

Variable, die verwendet wird, um einer Programm-Organisationseinheit einen Wert (als Eingangs- oder Eingangs-Ausgangs-Parameter) zur Verfügung zu stellen, oder eine Variable, die verwendet wird, um einen Wert von einer Programm-Organisationseinheit (als Ausgangs- oder Eingangs-Ausgangsparameter) zurückzugeben

3.74

Referenz

(en: reference)

anwenderdefinierte Daten eines bestimmten Typs, die die Adresszuordnung zu einer Variablen oder zu einer Funktionsbaustein-Instanz enthalten

3.75

Stromfluss

(en: power flow)

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

3.76

Pragma

(en: pragma)

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

3.77

programmieren

(en: program)

entwerfen, schreiben und testen von Anwenderprogrammen

3.78

Programm-Organisationseinheit

(en: program organization unit)

Funktion, Funktionsbaustein, Klasse oder Programm

3.79

Real-Literal

(en: real literal)

Literal, das den Wert vom Typ `REAL` oder `LREAL` direkt darstellt

3.80

Ressource

(en: resource)

Sprachelement, das einer „Signalverarbeitungsfunktion“ und ihrer „Mensch-Maschine-Schnittstelle“ und den „Funktionen der Sensor- und Aktor-Schnittstelle“ entspricht, sofern sie vorhanden sind

3.81

Ergebnis

(en: result)

Wert, der als ein Ausgang einer Programm-Organisationseinheit zurückgegeben wird

3.82

Rücksprung

(en: return)

Sprachkonstrukt innerhalb einer Programm-Organisationseinheit, das ein Ende der Ausführungsfolgen in der Einheit bezeichnet

3.83

steigende Flanke

(en: rising edge)

Wechsel einer booleschen Variablen von 0 nach 1

3.84

Anwendungsbereich

(en: scope)

Menge von Programm-Organisationseinheiten, in der eine Deklaration oder eine Marke gültig ist

3.85

Semantik

(en: semantics)

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

3.86

semigrafische Darstellung

(en: semigraphical representation)

Darstellung der grafischen Information durch die Anwendung eines begrenzten Zeichensatzes

3.87

Signatur

(en: signature)

Menge von Informationen, die eindeutig die Identität der Parameter-Schnittstelle einer Methode, definiert, bestehend aus ihrem Namen und den Namen, Typen und der Reihenfolge aller ihrer Parameter (d. h. Eingänge, Ausgänge, Ein-Aus-Variablen und Ergebnistyp)

3.88

Einzelelement-Variable

(en: single element variable)

Variable, die ein einzelnes Datenelement darstellt

3.89

statische Variable

(en: static variable)

Variable, deren Wert von einem Aufruf bis zum nächsten gespeichert wird

3.90

Schritt

(en: 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

3.91

strukturierter Datentyp

(en: structured data type)

zusammengesetzter Datentyp, der unter Verwendung einer Deklaration `STRUCT` oder `FUNCTION_BLOCK` deklariert ist

3.92

Indizieren

(en: subscripting)

Mechanismus zur Referenzierung eines Feldelements mittels einer Feldreferenz und eines oder mehrerer Ausdrücke, die die Position des Elements angeben, wenn sie ausgewertet werden

3.93

Task

(en: task)

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

3.94

Zeitliteral

(en: time literal)

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

3.95

Transition

(en: transition)

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

3.96

vorzeichenlose ganze Zahl

(en: unsigned integer)

ganze Zahl, die nur positive und Null-Werte enthält

3.97

vorzeichenloses ganzzahliges Literal

(en: unsigned integer literal)

ganzzahliges Literal, das kein führendes Pluszeichen (+) oder Minuszeichen (–) enthält

3.98

anwenderdefinierter Datentyp

(en: user-defined data type)

Datentyp, der vom Anwender definiert ist

BEISPIEL Aufzählung, Feld oder Struktur.

3.99

Variable

(en: variable)

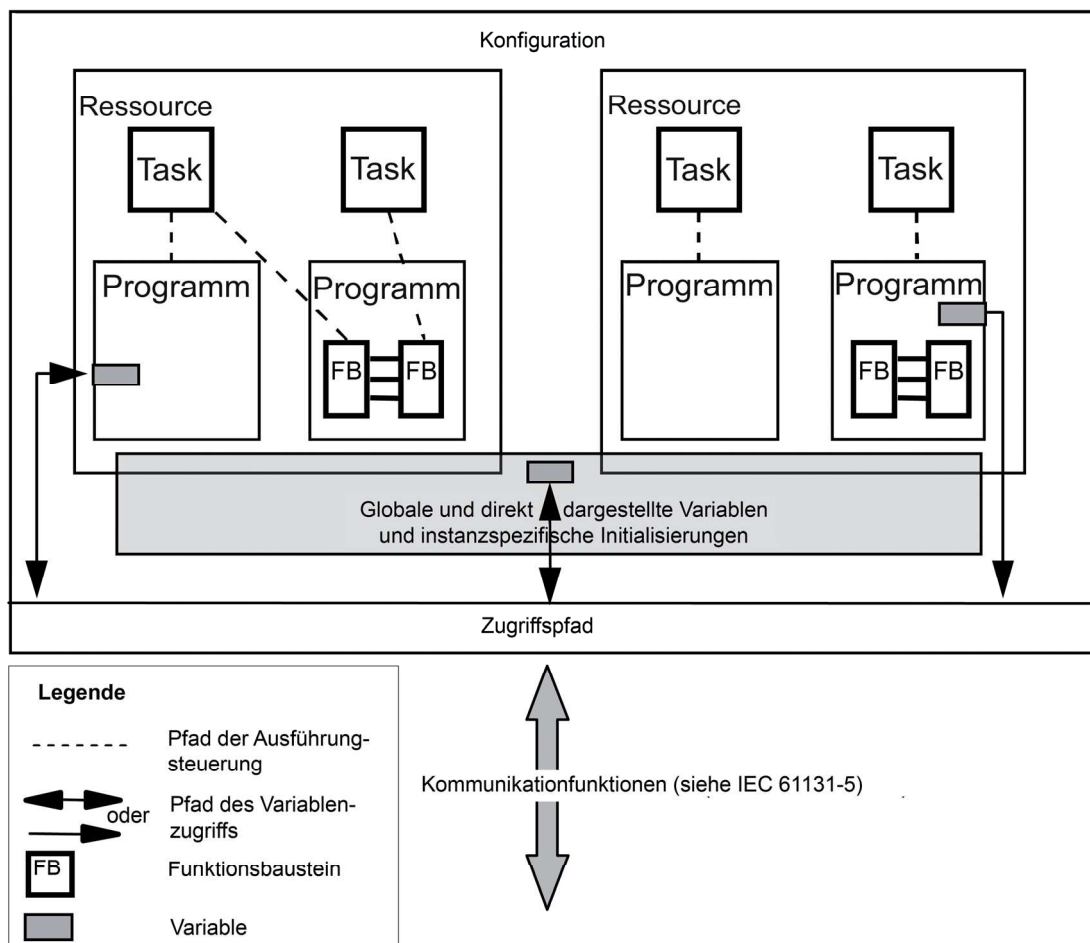
Softwareelement, das jeweils verschiedene Werte annehmen kann, einen zur Zeit

4 Architekturmodelle

4.1 Software-Modell

Die grundlegenden Hochsprachenelemente und ihre Beziehungen untereinander sind in Bild 1 veranschaulicht.

Sie bestehen aus Elementen, die durch die Anwendung der Sprachen programmiert sind, die in dieser Norm definiert sind, das sind Programme, Funktionsbausteine, Klassen, Funktionen und Konfigurationselemente wie Konfigurationen, Ressourcen, Tasks, globale Variablen, Zugriffspfade und instanzspezifische Initialisierungen, die zur Installation von SPS-Programmen in den SPS-Systeme dienen.



ANMERKUNG 1 Bild 1 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 Funktionsbaustein-Instanzen oder andere Sprachelemente enthalten, wie sie in diesem Teil der IEC 61131 definiert sind.

Eine Task ist in der Lage, die Ausführung einer Menge von Programmen und Funktionsbaustein-Instanzen z. B. periodisch anzustoßen.

Konfigurationen und Ressourcen können mit Hilfe der Funktionen „Bedienerschnittstelle“, „Programmierung, Testen und Beobachten“ oder „Betriebssystem“ gestartet oder gestoppt werden, so wie es in IEC 61131-1 definiert ist. Das Starten einer Konfiguration muss die Initialisierung ihrer globalen Variablen bewirken, 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 6.8.2 definiert, während die Mechanismen zum Starten und Stoppen von Konfigurationen und Ressourcen über Kommunikationsfunktionen in IEC 61131-5 definiert sind.

Programme, Ressourcen, globale Variable, Zugriffspfade (und ihre zugehörigen Zugriffsberechtigungen) und Konfigurationen können durch die „Kommunikationsfunktion“ geladen und gelöscht werden, die in IEC 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 dieser Norm definiert.

Die Abbildung der Sprachelemente auf die Kommunikationsobjekte muss so erfolgen, wie es in IEC 61131-5 definiert ist.

4.2 Kommunikationsmodell

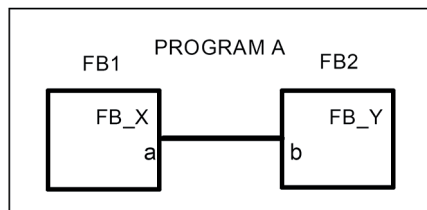
Bild 2 veranschaulicht die Wege, über die die Werte von Variablen zwischen Software-Elementen ausgetauscht werden können.

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

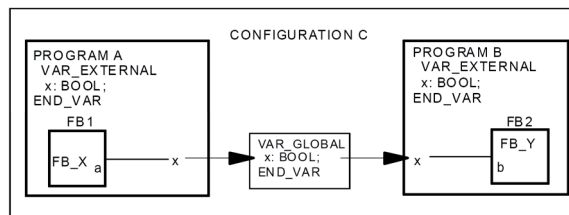
Variablenwerte können zwischen Programmen derselben Konfiguration über globale Variablen ausgetauscht werden, wie es die Variable x in Bild 2 b) veranschaulicht. Diese Variablen müssen als `GLOBAL` in der Konfiguration und als `EXTERNAL` im Programm deklariert werden.

Wie in Bild 2 c) veranschaulicht, können die Werte von Variablen zwischen verschiedenen 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 sind.

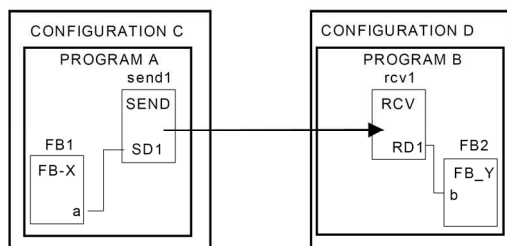
Außerdem können speicherprogrammierbare Steuerungen oder Nicht-SPS-Systeme Daten übertragen, die durch die Zugriffspfade zur Verfügung gestellt werden, wie es in Bild 2 d) veranschaulicht ist, indem die Mechanismen benutzt werden, die in IEC 61131-5 definiert sind.



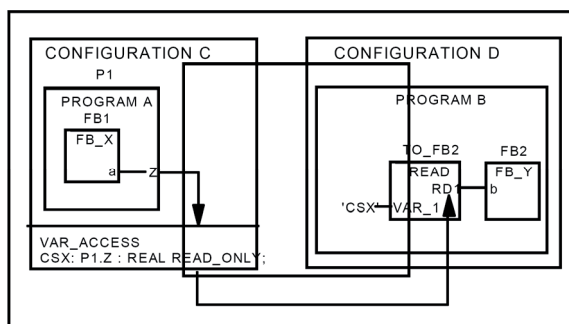
a) Datenflussverbindung innerhalb eines Programms



b) Kommunikation über globale Variable



c) Kommunikationsfunktionsbausteine



d) Kommunikation über Zugriffspfade

ANMERKUNG 1 Bild 2 dient nur zur Veranschaulichung. Die grafische Darstellung ist nicht normativ.

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

ANMERKUNG 3 Die Einzelheiten der Kommunikationsbausteine sind in Bild 2 nicht gezeigt.

ANMERKUNG 4 Die Zugriffspfade können 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.

Bild 2 – Kommunikationsmodell

4.3 Programmiermodell

In Bild 3 sind die SPS-Sprachelemente zusammengefasst. Die Kombinationen dieser Elemente müssen den folgenden Regeln gehorchen:

- 1) Datentypen müssen deklariert werden, indem die Standard-Datentypen und alle vorher abgeleiteten Datentypen benutzt werden.
- 2) Funktionen können deklariert werden, indem Standard- oder anwenderdefinierte Datentypen, Standardfunktionen und alle vorher definierte Funktionen benutzt werden.

Diese Deklaration muss die Mechanismen benutzen, die für die Sprache AWL, ST, KOP oder FBS definiert sind.

- 3) Funktionsbaustein-Typen können deklariert werden, indem Standard- und anwenderdefinierte Datentypen, Funktionen, Standardfunktionsbaustein-Typen und alle vorher definierten Funktionsbaustein-Typen benutzt werden.

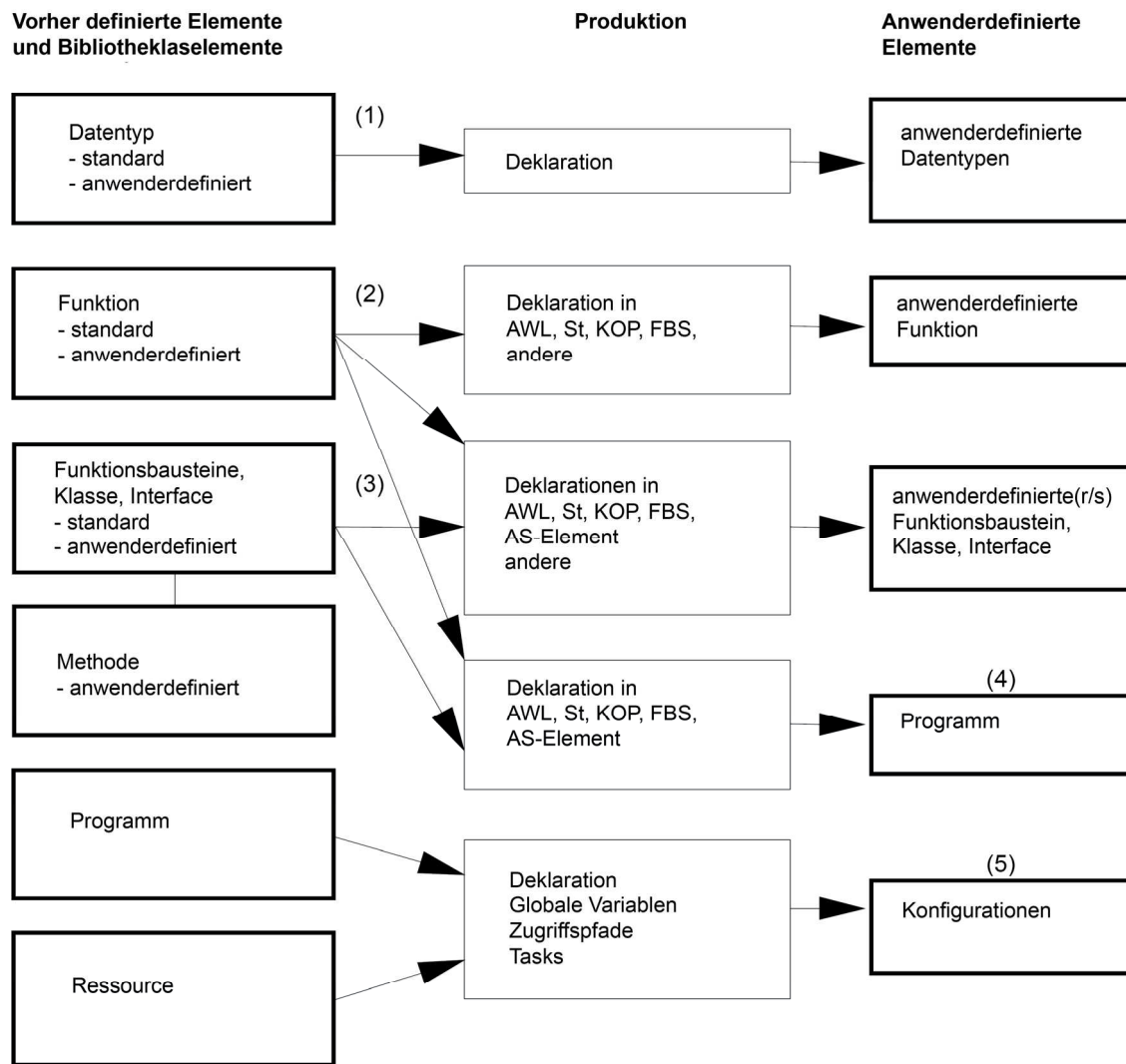
Diese Deklarationen müssen die Mechanismen benutzen, die für die Sprache AWL, ST, KOP oder FBS definiert sind, und können die Elemente der Ablaufsprache (AS) einschließen.

Zusätzlich dürfen objektorientierte Funktionsbaustein-Typen oder Klassen definiert werden, die Methoden und Interfaces verwenden.

- 4) Ein Programm muss deklariert werden, indem Standard- oder anwenderdefinierte Datentypen, Funktionen, Funktionsbausteine und Klassen benutzt werden.
Diese Deklaration muss Mechanismen benutzen, die für die Sprache AWL, ST, KOP oder FBS definiert sind, und kann die Elemente der Ablaufsprache (AS) einschließen.
- 5) Programme können zu Konfigurationen kombiniert werden, indem die Elemente benutzt werden, d. h. globale Variable, Ressourcen, Tasks und Zugriffspfade.

Der Verweis auf „vorher definierte“ Datentypen, Funktionen und Funktionsbausteine in den obigen Regeln soll bedeuten, dass, wenn ein derartiges vorher definiertes Element einmal deklariert wurde, seine Definition z. B. in einer „Bibliothek“ mit vorher definierten Elementen zur Verfügung steht, um sie in weiteren Definitionen zu benutzen.

Es darf auch eine andere Programmiersprache als die, die in dieser Norm definiert sind, zur Programmierung einer Funktion, eines Funktionsbaustein-Typs und von Methoden verwendet werden.



KOP: Kontaktplan
FBS: Funktionsbaustein-Sprache
AWL: Anweisungsliste
ST: Strukturierter Text
Andere: Andere Programmiersprachen

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

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

5 Normerfüllung

5.1 Allgemeines

Eine SPS-Programmier- und Testeinrichtung (PUTE), wie in IEC 61131-1 definiert, die den Anspruch erhebt, vollständig oder teilweise die Anforderungen dieses Teils der IEC 61131 zu erfüllen, muss dies genau so tun, wie es nachstehend beschrieben ist.

- a) Sie muss eine Untermenge der Eigenschaften zur Verfügung stellen und die entsprechende Erfüllungsaussage des Herstellers zu Verfügung stellen, so wie es unten definiert ist.
- b) Sie darf nicht die Einbeziehung von ersetzenden oder zusätzlichen Sprachelementen erforderlich machen, um irgendwelche Eigenschaften zu erreichen.
- c) Sie muss ein Dokument zur Verfügung stellen, dass alle herstellerspezifische Erweiterungen festlegt. Dies sind alle Eigenschaften, die vom System nicht zugelassenen oder nicht festgelegt sind.
- d) Sie muss ein Dokument zur Verfügung stellen, das alle herstellerspezifische Eigenschaften spezifiziert. Dies umfasst die Implementierungsabhängigkeiten, die explizit in diesem Teil von IEC 61131 genannt sind, und die Grenzparameter wie maximale Länge, Anzahl, Größe, Wertebereich, die nicht explizit hier aufgezählt sind.
- e) Sie muss ein Dokument zur Verfügung stellen, das alle Fehler festlegt, die erkennbar sind und von der Implementierung gemeldet werden. Dies umfasst die Fehler, die explizit in diesem Teil genannt sind, und die Fehler, die während der Aufbereitung des Programms für die Ausführung und während der Ausführung des Programms erkennbar sind.

ANMERKUNG Fehler, die während der Programmausführung auftreten, sind in diesem Teil von IEC 61131 nur teilweise festgelegt.

- f) Sie darf keinen der in dieser Norm definierten Namen von Datentypen, Funktionen oder Funktionsbausteinen für herstellerspezifische Eigenschaften verwenden, deren Funktionalität sich von dem unterscheidet, was in diesem Teil der IEC 61131 beschrieben ist.

5.2 Eigenschaftentabelle

Alle Tabellen in diesem Teil der IEC 61131 werden für einen besonderen Zweck in einer gemeinsamen Weise verwendet. Die erste Spalte enthält die „Nummer der Eigenschaft“, die zweite Spalte gibt die „Beschreibung der Eigenschaft“, die folgenden Spalten können Beispiele oder weitere Informationen enthalten. Diese Tabellenstruktur wird in der Erfüllungsaussage des Hersteller verwendet.

5.3 Erfüllungsaussage des Herstellers

Der Hersteller darf eine beliebige konsistente Untermenge von Eigenschaften, die in den Eigenschaftentabellen aufgezählt sind, definieren, und er muss die zur Verfügung gestellte Untermenge in der „Erfüllungsaussage des Herstellers“ bekannt geben.

Die Erfüllungsaussage des Herstellers muss in der Dokumentation enthalten sein, die dem System beigelegt ist, oder sie muss vom System selbst erzeugt werden.

Das Format der Erfüllungsaussage des Herstellers muss die folgenden Informationen liefern. Bild 4 zeigt ein Beispiel.

- Die allgemeinen Informationen müssen den Namen und die Adresse des Herstellers, den Namen und die Version des Produkts, den Typ und die Version der Steuerung und das Ausgabedatum umfassen.
- Für jede implementierte Eigenschaft muss die Nummer der entsprechenden Eigenschaftentabelle, die Eigenschaftensnummer und die anwendbare Programmiersprache angegeben werden.
- Der Titel und der Untertitel der Eigenschaftentabelle, die Beschreibung der Eigenschaft, Beispiele, Herstellerbemerkungen usw. sind optional.

Tabelle und Eigenschaften, die nicht implementiert sind, dürfen entfallen.

IEC 61131-3 SPS-Programmiersprachen						
Hersteller: Herstellername, Adresse, usw.						
Produkt: Produktname, Version usw. Steuerungstypspezifische Untermenge usw.						
Datum: 2012-05-01						
Dieses Produkt erfüllt die Anforderungen der Norm für die folgenden Spracheigenschaften:						
Eigen- schaft Nr.	Tabellennummer und Titel / Beschreibung der Eigenschaft	Normgerecht implementiert in der Sprache: (✓)				Anmerkung des Herstellers
		KOP	FBS	ST	AWL	
	Tabelle 1 – Zeichensatz					
1	ISO/IEC 10646:2012, Information technology – Universal Coded Character SET (UCS)	✓	✓	✓	✓	
2a	Kleinbuchstaben a, b, c, ...	✓	✓	✓		Kein „ß, ü, ä, ö“
2b	Nummernzeichen # Siehe Tabelle 5	✓				
2c	Dollarzeichen \$ Siehe Tabelle 6		✓			
	Tabelle 2 – Bezeichner					
1	Großbuchstaben und Zahlen IW215					
2	Groß- und Kleinbuchstaben, Zahlen, eingebettete Unterstriche					
3	Groß und Kleinschreibung, Zahlen, führende und eingebettete Unterstriche					
	Tabelle 3 – Kommentare					
1	Einzeilenkommentar //...					
2a	Mehrzeilenkommentar (* ... *)					
2b	Mehrzeilenkommentar /* ... */					
3a	Geschachtelter Kommentar (* ..(* .. *) ..*)					
3b	Geschachtelter Kommentar /* .. /* .. */ .. */					
	Tabelle 4 – Pragma					
1	Pragma mit geschweiften Klammern { ... }					
	Tabelle 5 – Numerische Literale					
1	Ganzzahliges (Integer) Literal -12					
2	Reeles Literal -12.0					
3	Reeles Literal mit Exponent -1.34E-12					
4	Binäres Literal 2#1111_1111					
5	Oktales Literal 8#377					
6	Hexadezimaler Literal 16#FF					
7	Boolesche Null oder Eins					
8	Boolesches FALSE und TRUE					
9	Literal mit Typangabe INT#-123					
	Usw.					

Bild 4 – Erfüllungsaussage des Herstellers (Beispiel)

6 Gemeinsame Elemente

6.1 Gebrauch der gedruckten Zeichen

6.1.1 Zeichensatz

Tabelle 1 zeigt den Zeichensatz der textuellen Sprachen und textuellen Elemente von grafischen Sprachen. Die Zeichen werden nach ISO/IEC 10646 dargestellt.

Tabelle 1 – Zeichensatz

Nr.	Beschreibung
1	ISO/IEC 10646
2a	Kleinbuchstaben ^a : a, b, c
2b	Nummernzeichen: # Siehe Tabelle 5
2c	Dollarzeichen: \$ Siehe Tabelle 6
^a Wenn Kleinbuchstaben unterstützt werden, darf die Groß-/Kleinschreibung in den Sprachelementen nicht signifikant sein, ausgenommen innerhalb von Kommentaren wie in 6.1.5 definiert, Zeichenfolgen wie in 6.3.3 definiert und Variablen vom Typ <code>STRING</code> und <code>WSTRING</code> wie in 6.3.3 definiert.	

6.1.2 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.

Das Zeichen Unterstrich muss in Bezeichnern signifikant sein, z. B. müssen `A_BCD` und `AB_CD` müssen als unterschiedliche Bezeichner interpretiert werden. Mehrere führende und mehrfach eingebettete 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.

Zur Eindeutigkeit müssen mindestens sechs Zeichen in allen Systemen unterstützt werden, die den Gebrauch von Bezeichnern unterstützen, 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 herstellerspezifisch.

Die Eigenschaften von Bezeichnern und Beispiele sind in Tabelle 2 gezeigt.

Tabelle 2 – Bezeichner

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

6.1.3 Schlüsselwörter

Schlüsselwörter sind eindeutige Kombinationen von Zeichen, die als individuelle syntaktische Elemente angewendet werden. 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. Sie dürfen für keinen anderen Zweck genutzt werden, z. B. als Variablennamen oder Erweiterungen.

6.1.4 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, Bezeichnungen, direkt dargestellten Variablen oder Kombinationen von Begrenzen, z. B. für Kommentare. Ein „Leerzeichen“ ist sowohl als SPACE-Zeichen mit dem Kode-Wert 32 dezimal als auch als nicht druckbare Zeichen wie Tabulator, neue Zeile usw., für die keine Kodierung in ISO/IEC 10646 angegeben ist, definiert.

6.1.5 Kommentare

Es gibt verschiedene Arten von Anwenderkommentaren, die in Tabelle 3 aufgelistet sind:

- 1) Einzelzeilen-Kommentare müssen mit der Zeichenkombination `//` beginnen und beim nächsten Zeilenvorschub, bei neuer Zeile, bei Formularvorschub (Seite) oder „Wagenrücklauf“ enden.

In Einzelzeilen-Kommentaren hat die spezielle Zeichenkombination `(*` und `*)` oder `/*` und `*/` keine besondere Bedeutung.

- 2) Mehrzeilen-Kommentare müssen am Anfang und Ende von der speziellen Zeichenkombination `(*` und `*)` eingeschlossen sein.

Ein alternativer Mehrzeilen-Kommentar darf zur Verfügung gestellt werden, der die speziellen Zeichenkombinationen `/*` und `*/` verwendet.

In Mehrzeilen-Kommentaren hat die spezielle Zeichenkombination `//` keine spezielle Bedeutung.

Kommentare müssen überall im Programm erlaubt sein, wo Leerzeichen erlaubt sind, ausgenommen innerhalb von Zeichenfolgen-Literalen.

Kommentare dürfen keine syntaktische oder semantische Bedeutung in einer der Sprachen haben, die in diese Norm definiert sind. Sie werden wie ein Leerzeichen behandelt.

Geschachtelte Kommentare werden folgendermaßen verwendet:

- Paare von `(*, *)`, z. B. `(* ... (* geschachtelt *) ... *)` oder
- Paare von `/*, */`, z. B. `/* ... /* geschachtelt */ ... */`.

Tabelle 3 – Kommentare

Nr.	Beschreibung	Beispiele
1	Einzeilige Kommentare mit // ...	X := 13; // Kommentar für eine Zeile // Ein einzeiliger Kommentar kann am Anfang der Zeile // beginnen.
2a	Mehrzeilige Kommentare mit (* ... *)	(* Kommentar *) (***** Ein eingerahmter Kommentar über drei Zeilen *****)
2b	Mehrzeilige Kommentare mit /* ... */	/* Kommentar in einer oder mehr Zeilen */
3a	Geschachtelter Kommentar mit (* ..(*) ..*)	(* (* GESCHACHTELT *) *)
3b	Geschachtelter Kommentar mit /* .. /* .. */ .. */	/* /* GESCHACHTELT */ /*

6.2 Pragma

Wie in Tabelle 4 veranschaulicht, müssen Pragmas am Beginn und Ende durch geschweifte Klammern { und } eingeschlossen werden. Die Syntax und Semantik von speziellen Pragma-Konstrukten sind herstellereinspezifisch. Pragmas müssen überall in Programmen zulässig sein, wo Leerzeichen erlaubt sind, ausgenommen innerhalb von Zeichenfolgen-Literalen.

Tabelle 4 – Pragma

Nr.	Beschreibung	Beispiele
1	Pragma mit { ... } geschweiften Klammern	{VERSION 2.0} {AUTOR JHC} {x := 256, y := 384}

6.3 Literale – Externe Darstellung von Daten

6.3.1 Allgemeines

Externe Darstellungen von Daten bei den verschiedenen SPS-Programmiersprachen müssen die numerischen Literale, Zeichenfolge-Literale und Zeitlitterale umfassen.

Es ist notwendig, die äußere Darstellungen für zwei unterschiedliche Typen vom zeitbezogenen Daten zu unterstützen:

- Daten für die Zeitdauer zum Messen und Steuern von abgelaufener Zeit eines Steuerereignisses
- und Daten für die Uhrzeit, die auch Datumsinformation zur Synchronisation des Beginns und des Ende eines Steuerungsereignisses zu einer absoluten Zeitreferenz haben dürfen.

6.3.2 Numerisch Literale und Zeichenfolge-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 sein. 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.

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 Dezimalzahlen 10 bis 15 haben. Basisbezogene Zahlen dürfen kein Vorzeichen „+“ oder „-“ beinhalten. Sie werden als Bitfolge-Literale interpretiert.

Numerische Literale, die eine positive ganze Zahl darstellen, dürfen als Bitfolge-Literale verwendet werden.

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 5 gezeigt.

Der Datentyp eines booleschen oder numerischen Literals darf durch Voranstellen eines Datentyp-Präfixes festgelegt werden und besteht aus dem Namen eines elementaren Datentyps und einem „#“-Zeichen. Beispiele siehe in Eigenschaft 9 in Tabelle 5.

Tabelle 5 – Numerische Literale

Nr.	Beschreibung	Beispiele	Erklärung
1	Ganzzahliges (integer) Literal	-12, 0, 123_456, +986	
2	Reelles (real) Literal	0.0, 0.4560, 3.14159_26	
3	Reelles (real) Literal mit Exponent	-1.34E-12, -1.34e-12 1.0E+6, 1.0e+6 1.234E6, 1.234e6	
4	Binäres Literal	2#1111_1111 2#1110_0000	Basis 2 Literal 255 dezimal 224 dezimal
5	Oktales Literal	#8#377 8#340	Basis 8 Literal 255 dezimal 224 dezimal
6	Hexadezimal Literal	16#FF, 16#ff 16#E0, 16#e0	Basis 16 Literal 255 dezimal 224 dezimal
7	Boolesche Null und Eins	0, 1	
8	Boolesches FALSE und TRUE	FALSE TRUE	
9	Literal mit Typangabe	INT#-123 INT#16#7FFF WORD#16#AFF WORD#1234 UINT#16#89AF CHAR#16#41 BOOL#0 BOOL#1 BOOL#FALSE BOOL#TRUE	INT Darstellung des Dezimalwerts -123 INT Darstellung des Dezimalwerts 32767 WORD Darstellung des Hexadezimalwerts 0AFF WORD Darstellung des Dezimalwerts 1234 = 16#4D2 UINT Darstellung des Hexadezimalwerts 89AF CHAR Darstellung von 'A'
ANMERKUNG 1 Die Schlüsselwörter FALSE und TRUE korrespondieren mit den booleschen Werten 0 und 1.			
ANMERKUNG 2 Die Eigenschaft 5 'Oktales Literal' ist nicht mehr erwünscht und könnte in der nächste Ausgabe dieses Teils von IEC 61131 nicht mehr enthalten sein.			

6.3.3 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, 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-Zeichenkode interpretiert werden. Dies ist in Eigenschaft 1 von Tabelle 6 gezeigt.

- Ein Zeichenfolge-Literal, das ein Doppel-Byte-Zeichen ist, ist eine Sequenz von null oder mehr Zeichen aus der ISO/IEC 10646, 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-Zeichenkodes interpretiert werden. Dies ist in Eigenschaft 2 von Tabelle 6 gezeigt.

ANMERKUNG Zusammenhang von ISO/IEC 10646 und Unicode:

Obwohl die Zeichenkodes und Kodierformate zwischen Unicode und ISO/IEC 10646 aufeinander abgestimmt sind, hat die Unicode-Norm zusätzliche Anforderungen an die Implementierungen, um sicherzustellen, dass sie die Zeichen einheitlich über Plattformen und Anwendungen behandeln. Daher liefert der Unicode eine beträchtliche Menge von funktionalen Zeichenspezifikationen, Zeichendaten, Algorithmen und substanziellem Hintergrundmaterial, die nicht in ISO/IEC 10646 enthalten sind.

2-Zeichen-Kombinationen, die mit dem Dollar-Zeichen beginnen, müssen so interpretiert werden, wie es in Tabelle 7 gezeigt ist, wenn sie in Zeichenfolgen auftreten.

Tabelle 6 – Zeichenfolge-Literale

Nr.	Beschreibung	Beispiele
Einzelbyte-Zeichen oder Zeichenfolgen eingeschlossen von ‘ ‘		
1a	Leere Zeichenfolge (Länge null)	' '
1b	Zeichenfolge der Länge eins oder Zeichen CHAR, das ein Einzelzeichen enthält	'A'
1c	Zeichenfolge der Länge eins oder Zeichen CHAR, das ein „Leerzeichen“ enthält	' '
1d	Zeichenfolge der Länge eins, die ein „Einzel-Anführungszeichen“ enthält	'\$''
1e	Zeichenfolge der Länge eins, die ein „Doppel-Anführungszeichen“ enthält	'\"'
1f	Unterstützung der Zweizeichen-Kombination aus Tabelle 7	'\$R\$L'
1g	Unterstützung der Zeichendarstellung mit „\$“ und zwei hexadezimalen Zeichen	'\$0A'
Doppelbyte-Zeichen oder Zeichenfolge eingeschlossen in \" \" (siehe Anmerkung)		
2a	Leere Zeichenfolge (Länge null)	\" \"
2b	Zeichenfolge der Länge eins oder Zeichen WCHAR, das ein einzelnes Zeichen enthält	\"A\"
2c	Zeichenfolge der Länge eins oder Zeichen WCHAR, das ein „Leerzeichen“ enthält	\" \"
2d	Zeichenfolge der Länge eins oder Zeichen WCHAR, das ein „Einzel-Anführungszeichen“ enthält	\" '\$ '\"
2e	Zeichenfolge der Länge eins oder Zeichen WCHAR, das ein „Doppel-Anführungszeichen“ enthält	\" \"\$ '\"
2f	Unterstützung der Zweizeichen-Kombination aus Tabelle 7	\"\$R\$L\"
2g	Unterstützung der Zeichendarstellung mit „\$“ und vier hexadezimalen Zeichen	\"\$00C4\"
Einzelbyte-Zeichen mit Typangabe oder Zeichenliteral mit #		
3a	Zeichenfolge mit Typangabe	STRING# 'OK'
3b	Zeichen mit Typangabe	CHAR# 'X'
Doppelbyte-Zeichenfolge-Literal mit # (siehe Anmerkung)		
4a	Doppelbyte-Zeichenfolge mit Typangabe (verwendet „Doppel-Anführungszeichen“)	WSTRING# \"OK\"
4b	Doppelbyte-Zeichen (verwendet „Doppel-Anführungszeichen“)	WCHAR# \"X\"
4c	Doppelbyte-Zeichenfolge mit Typangabe (verwendet „Einzel-Anführungszeichen“)	WSTRING# 'OK'
4d	Doppelbyte-Zeichen (verwendet „Einzel-Anführungszeichen“)	WCHAR# 'X'
ANMERKUNG Wenn eine bestimmte Implementierung die Eigenschaft 4, jedoch nicht die Eigenschaft 2 unterstützt, darf der Hersteller eine herstellerspezifische Syntax und Semantik für den Gebrauch der Doppel-Anführungszeichen festlegen.		

Tabelle 7 – Zwei-Zeichen-Kombinationen in Zeichenfolgen

Nr.	Beschreibung	Kombinationen
1	Dollar-Zeichen	\$ \$
2	Einzel-Anführungszeichen	\$ '
3	Zeilenvorschub (Line feed)	\$ L oder \$ l
4	Neue Zeile (Newline)	\$ N oder \$ n
5	Neue Seite (Form feed, page)	\$ P oder \$ p
6	Wagenrücklauf (Carriage return)	\$ R oder \$ r
7	Tabulator (Tab)	\$ T oder \$ t
8	Doppel-Anführungszeichen	\$ "
ANMERKUNG 1 Das Zeichen „neue Zeile“ bietet ein implementierungsabhängiges Mittel, um das Ende einer Datenzeile zu definieren. Beim Drucken bewirkt es das Beenden einer Datenzeile und das Fortsetzen am Anfang 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.		

6.3.4 Zeitlitterale

Die Daten für Zeitdauer müssen links durch das Schlüsselwort `T#`, `TIME#` oder `LTIME#` begrenzt werden. Die Darstellung von Daten für Zeitdauer wie Tage, Stunden, Minuten, Sekunden und Bruchteile einer Sekunde oder alle Kombinationen hiervon muss in der Weise erfolgen, wie es in Tabelle 8 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 8 veranschaulicht, sind sowohl positive als auch negative Werte für die Zeitdauerangaben zulässig.

Tabelle 8 – Zeitdauer-Literale

Nr.	Beschreibung	Beispiele
Zeitdauer-Abkürzungen		
1a	d	Tag
1b	h	Stunde
1c	m	Minute
1d	s	Sekunde
1e	ms	Millisekunde
1f	us (kein μ verfügbar)	Mikrosekunde
1g	ns	Nanosekunde
Zeitdauer-Literale ohne Unterstriche		
2a	Kurzes Präfix	T#14ms T#-14ms LT#14.7s T#14.7m T#14.7h t#14.7d t#25h15m lt#5d14h12m18s3 5ms t#12h4m34ms230us400ns
2b	Langes Präfix	TIME#14ms TIME#-14ms time#14.7s
Zeitdauer-Literale mit Unterstrich		
3a	Kurzes Präfix	t#25h_15m t#5d_14h_12m_18s_3.5ms LTIME#5m_30s_500ms_100.1us
3b	Langes Präfix	TIME#25h_15m ltime#5d_14h_12m_18s_3.5ms LTIME#34s_345ns

6.3.5 Datums- und Uhrzeitliteral

Die vorangestellten Schlüsselwörter (Präfix) für die Literale Datum und Uhrzeit sind in Tabelle 9 gezeigt.

Tabelle 9 – Datums- und Uhrzeitlitterale

Nr.	Beschreibung	Beispiele
1a	Datumsliteral (langes Präfix)	DATE#1984-06-25, date#2010-09-22
1b	Datumsliteral (kurzes Präfix)	D#1984-06-25
2a	Langes Datumsliteral (langes Präfix)	LDATE#2012-02-29
2b	Langes Datumsliteral (kurzes Präfix)	LD#1984-06-25
3a	Uhrzeitliteral (langes Präfix)	TIME_OF_DAY#15:36:55.36
3b	Uhrzeitliteral (kurzes Präfix)	TOD#15:36:55.36
4a	Langes Uhrzeitliteral (kurzes Präfix)	LTOD#15:36:55.36
4b	Langes Uhrzeitliteral (langes Präfix)	LONG_TIME_OF_DAY#15:36:55.36
5a	Datum und Uhrzeit (langes Präfix)	DATE_AND_TIME#1984-06-25-15:36:55.360227400
5b	Datum und Uhrzeit (langes Präfix)	DT#1984-06-25-15:36:55.360_227_400
6a	Langes Datums- und Uhrzeitliteral (langes Präfix)	LONG_DATE_AND_TIME#1984-06-25-15:36:55.360_227_400
6b	Langes Datums- und Uhrzeitliteral (kurzes Präfix)	LDT#1984-06-25-15:36:55.360_227_400

6.4 Datentypen

6.4.1 Allgemeines

Ein Datentyp ist eine Klassifizierung, die für Literale und Variable die möglichen Werte definiert, die Operationen, die durchgeführt werden können, und die Art, wie die Werte gespeichert werden.

6.4.2 Elementare Datentypen (BOOL, INT, REAL, STRING usw.)

6.4.2.1 Festlegung der elementaren Datentypen

Eine Menge von (vordefinierten) elementaren Datentypen ist durch diese Norm festgelegt.

Die elementaren Datentypen, die Schlüsselwörter für jeden Datentyp, die Anzahl der Bits je Datenelement und der Wertebereich für jeden elementaren Datentyp müssen Tabelle 10 folgen.

Tabelle 10 – Elementare Datentypen (1 von 2)

Nr.	Beschreibung	Schlüsselwort	Voreingestellter Anfangswert	N (Bits) ^a
1	Boolesche (boolean)	BOOL	0, FALSE	1 ^h
2	Kurze ganze Zahl (short integer)	SINT	0	8 ^c
3	Ganze Zahl (integer)	INT	0	16 ^c
4	Doppelte ganze Zahl (double integer)	DINT	0	32 ^c
5	Lange ganze Zahl (long integer)	LINT	0	64 ^c
6	Vorzeichenlose kurze ganze Zahl (unsigned short integer)	USINT	0	8 ^d
7	Vorzeichenlose ganze Zahl (unsigned integer)	UINT	0	16 ^d
8	Vorzeichenlose doppelte ganze Zahl (unsigned double integer)	UDINT	0	32 ^e
9	Vorzeichenlose lange ganze Zahl (unsigned long integer)	ULINT	0	64 ^d
10	Reelle Zahl (real number)	REAL	0.0	32 ^e
11	Lange reelle Zahl (long real)	LREAL	0.0	64 ^f
12a	Zeitdauer (duration)	TIME	T#0s	-- ^b
12b	Zeitdauer (long duration)	LTIME	LTIME#0s	64 ^{m,q}
13a	Datum (nur) (date (only))	DATE	(siehe Anmerkung)	-- ^b
13b	Langes Datum (nur) (long date (only))	LDATE	DT#1970-01-01	64 ⁿ
14a	Uhrzeit (nur) (time of day (only))	TIME_OF_DAY oder TOD	TOD#00:00:00	-- ^b
14b	Uhrzeit (nur) (time of day only))	LTIME_OF_DAY oder LTOD	LTOD#00:00:00	64 ^{p,q}
15a	Datum und Uhrzeit (Date and time of day)	DATE_AND_TIME oder DT	(siehe Anmerkung)	-- ^b
15b	Langes Datum und Uhrzeit (Long date and time of day)	LDATE_AND_TIME oder LDT	DT#1970-01-01-00:00:00	64 ^{p,q}
16a	Einzelbyte-Zeichenfolge mit variabler Länge	STRING	' ' (leer)	8 ^{i,g,k,l}

Tabelle 10 – Elementare Datentypen (2 von 2)

Nr.	Beschreibung	Schlüsselwort	Voreingestellter Anfangswert	N (Bits) ^a
16b	Doppelbyte-Zeichenfolge mit variabler Länge	WSTRING	"" (leer)	16 ^{i, g, k, l}
17a	Einzelbyte-Zeichen	CHAR	'\$00'	8 ^{g, l}
17b	Doppelbyte-Zeichen	WCHAR	"\$0000"	16 ^{g, l}
18	Bitfolge der Länge 8	BYTE	16#00	8 ^{j, g}
19	Bitfolge der Länge 16	WORD	16#0000	16 ^{j, g}
20	Bitfolge der Länge 32	DWORD	16#0000_0000	32 ^{j, g}
21	Bitfolge der Länge 64	LWORD	16#0000_0000_0000_0000	64 ^{j, g}
ANMERKUNG Herstellerspezifisch, weil das spezielle Anfangsdatum sich vom Datum 0001-01-01 unterscheidet.				
^a Einträge in dieser Spalte müssen so interpretiert werden, wie es in den Tabellen-Fußnoten festgelegt ist. ^b Der Wertebereich und die Genauigkeit der Darstellung in diesen Datentypen sind herstellerspezifisch. ^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 $-(2^{N-1}) - 1$. ^e Der Wertebereich für Variablen dieses Datentyps muss festgelegt sein wie in IEC 60559 für das Basis-Gleitpunkt-Format mit einfacher Länge. Ergebnisse der arithmetischen Anweisungen mit denormalisierten Werten, unendlich oder Werten, die keine Zahl sind, sind herstellerspezifisch. ^f Der Wertebereich für Variablen dieses Datentyps muss festgelegt sein wie in IEC 60559 für das Basis-Gleitpunkt-Format mit doppelter Länge. Ergebnisse der arithmetischen Anweisungen mit renormalisierten Werten, unendlich oder Werten, die keine Zahl sind, sind herstellerspezifisch. ^g Ein numerischer Wertebereich ist für diesen Datentyp nicht anwendbar. ^h Die möglichen Werte von Variablen dieses Datentyps müssen 0 und 1 sein entsprechend den Schlüsselwörtern FALSE und TRUE. ⁱ Der Wert von 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. ^k Die maximal zulässige Länge von STRING- und WSTRING-Variablen ist herstellerspezifisch. ^l Der Zeichenkode, der für CHAR, STRING, WCHAR und WSTRING verwendet wird, ist ISO/IEC 10646 (siehe 6.3.3). ^m Der Datentyp LTIME ist eine 64-Bit- Ganzzahl mit Vorzeichen und hat die Einheit Nanosekunden. ⁿ Der Datentyp LDATE ist eine 64-Bit- Ganzzahl mit Vorzeichen, hat die Einheit Nanosekunden und das Startdatum 1970-01-01. ^o Der Datentyp LTD ist eine 64-Bit- Ganzzahl mit Vorzeichen, hat die Einheit Nanosekunden und die Startzeit 1970-01-01-00:00:00. ^p Der Datentyp LTOD ist eine 64-Bit-Ganzzahl mit Vorzeichen, hat die Einheit Nanosekunden und die Startzeit Mitternacht TOD#00:00:00. ^q Die Genauigkeit der Aktualisierung der Werte dieses Zeitformats ist herstellerspezifisch, d. h. der Wert ist in Nanosekunden angegeben, er kann aber alle Mikrosekunde oder Millisekunde aktualisiert werden.				

6.4.2.2 Elementare Datentyp-Zeichenfolge (STRING, WSTRING)

Die unterstützte Maximallänge von Elementen des Typs STRING und WSTRING muss ein herstellerspezifischer Wert sein, der die Maximallänge von STRING und WSTRING definiert, die von der Programmier- und Testeinrichtung unterstützt wird.

Die explizite Maximallänge wird in der zugehörigen Deklaration durch eine in Klammern angegebene Maximallänge festgelegt. Sie darf den herstellerspezifisch unterstützten Maximalwert nicht überschreiten.

Der Zugriff auf einzelne Zeichen einer Zeichenfolge, die Elemente vom Datentyp CHAR und WCHAR verwendet, muss durch Verwenden eckiger Klammern und die Position des Zeichens in der Zeichenfolge unterstützt werden, wobei mit der Position 1 begonnen wird.

Es muss ein Fehler sein, wenn auf Doppelbyte-Zeichenfolgen durch Verwenden von Einzelbyte-Zeichen oder wenn auf Einzelbyte-Zeichenfolgen durch Verwenden von Doppelbyte-Zeichen zugegriffen wird.

BEISPIEL 1 STRING, WSTRINGs und CHAR, WCHAR.

a) Deklaration

```
VAR
  String1:   STRING[10] := 'ABCD';
  String2:   STRING[10] := '';
  aWStrings: ARRAY [0..1] OF WSTRING := ["1234", "5678"];
  Char1:     CHAR;
  WChar1:    WCHAR;

END_VAR
```

b) Verwendung von STRING und CHAR

```
Char1      := String1[2]; //ist äquivalent mit Char1 := 'B'
String1[3] := Char1;     //Ergebnisse in String1 := 'ABBD'
String1[4] := 'B';       //Ergebnisse in String1 := 'ABBB'
String1[1] := String1[4]; //Ergebnisse in String1 := 'BBBB'
String2     := String1[2]; (* Ergebnisse in String2 := 'B',
                           wenn implizite Konvertierung CHAR_TO_STRING implementiert wurde *)
```

c) Usage of WSTRING und WCHAR

```
WChar1:= aWStrings[1][2]; //ist äquivalent mit WChar1:= '6';
aWStrings[1][3]:=WChar1; //Ergebnisse in aWStrings[1]:= "5668"
aWStrings[1][4]:= "6";    //Ergebnisse in aWStrings[1]:= "5666"
aWStrings[1][1]:= aWStrings[1][4]; //Ergebnisse in String1:= "6666"
aWStrings[0]    := aWStrings[1][4]; (* Ergebnisse in aWStrings[0]:= "6";
                                     wenn implizite Konvertierung WCHAR_TO_WSTRING implementiert wurde *)
```

d) Äquivalente Funktions (siehe 6.6.2.6.11)

```
Char1 := String1[2];
      ist äquivalent mit
Char1 := STRING_TO_CHAR(Mid(IN := String1, L := 1, P := 2));
aWStrings[1][3]:=WChar1;
      ist äquivalent mit
REPLACE(IN1:=aWStrings[1], IN2 := WChar1, L := 1, P := 3);
```

e) Fehlerfälle

```
Char1 := String1[2]; //vermischt WCHAR, STRING
String1[2] := String2;
//erfordert implizite Konvertierung STRING_TO_CHAR, die nicht zulässig ist.
```

ANMERKUNG Die Datentypen für Einzelzeichen (CHAR) und WCHAR) können nur ein Zeichen enthalten. Zeichenfolgen können mehrere Zeichen enthalten. Daher dürfen Zeichenfolgen zusätzliche Verwaltungsinformation erforderlich machen, die bei den Einzelzeichen nicht benötigt wird.

BEISPIEL 2

Wenn der Typ STR10 deklariert ist durch

```
TYPE STR10: STRING[10] := 'ABCDEF'; END_TYPE
```

dann ist die Maximallänge von STR10 10 Zeichen, der voreingestellte Anfangswert 'ABCDEF' und die Anfangslänge der Datenelemente von Typ STR10 6 Zeichen.

6.4.3 Allgemeine Datentypen

Zusätzlich zu den elementaren Datentypen, die in Tabelle 10 gezeigt sind, kann die Hierarchie der allgemeinen (en: generic) Datentypen, die in Bild 5 gezeigt ist, bei der Festlegung von Eingängen und Ausgängen von Standardfunktionen und Funktionsbausteinen angewendet werden. Die allgemeinen Datentypen werden durch die Vorsilbe „ANY“ identifiziert.

Der Gebrauch von allgemeinen Datentypen ist in den folgenden Regeln beschrieben:

- 1) Der allgemeine Typ eines direkt abgeleiteten Typs muss derselbe sein wie der elementare Typ, von dem er abgeleitet ist.
- 2) Der allgemeine Typ eines Unterbereichstyps muss ANY_INT sein.
- 3) Der allgemeine Typ von allen anderen abgeleiteten Datentypen, die in Tabelle 11 definiert sind, muss ANY_DERIVED sein.

Der Gebrauch von allgemeine Datentypen in anwenderdeklarierten Programm-Organisationseinheiten ist nicht im Anwendungsbereich dieses Norm.

Allgemeine Datentypen		Allgemeine Datentypen	Gruppen von elementaren Datentypen
ANY		g)	
	ANY_DERIVED		
	ANY_ELEMENTARY		
	ANY_MAGNITUDE		
	ANY_NUM		
	ANY_REAL	h)	REAL, LREAL
	ANY_INT	ANY_UNSIGNED	USINT, UINT, UDINT, ULINT
		ANY_SIGNED	SINT, INT, DINT, LINT
	ANY_DURATION		TIME, LTIME
	ANY_BIT		BOOL, BYTE, WORD, DWORD, LWORD
	ANY_CHARS		
	ANY_STRING		STRING, WSTRING
	ANY_CHAR		CHAR, WCHAR
	ANY_DATE		DATE_AND_TIME, LDT, DATE, TIME_OF_DAY, LTOD

Bild 5 – Hierarchie der allgemeinen Datentypen

6.4.4 Anwenderdefinierte Datentypen

6.4.4.1 Deklaration (TYPE)

6.4.4.1.1 Allgemeines

Der Zweck der anwenderdefinierten Datentypen ist die Verwendung in der Deklaration von weiteren Datentypen und in Variablendeklarationen.

Ein anwenderdefinierter Typ kann überall verwendet werden, wo ein Basistyp verwendet werden kann.

Die anwenderdefinierten Datentypen müssen zwischen dem textuellen Konstrukt TYPE...END_TYPE deklariert werden.

Eine Typdeklaration besteht aus:

- dem Namen des Typs;
- einem „:“ (Doppelpunkt);
- der Deklaration des Typs selbst, wie es in den folgenden Unterabschnitten definiert ist.

BEISPIEL Typdeklaration.

```
TYPE
  myDatatype1: <Datentyp-Deklaration mit optionaler Initialisierung>;
END_TYPE
```

6.4.4.1.2 Initialisierung

Anwenderdefinierte Datentypen können mit anwenderdefinierten Werten initialisiert werden. Diese Initialisierung hat Vorrang vor dem voreingestellten Anfangswert.

Die anwenderdefinierte Initialisierung folgt der Typdeklaration und beginnt mit dem Zuweisungsoperator ':=', gefolgt von dem Anfangswert bzw. den Anfangswerten.

Es dürfen Literale (z. B. -123, 1.55, "abc") oder konstante Ausdrücke (z. B. 12*24) verwendet werden. Die verwendeten Anfangswerte müssen einen passenden Typ haben, d. h. vom selbem Typ sein oder einem Typ, der mit der impliziten Typkonvertierung konvertiert werden kann.

Es müssen die Regeln für die Initialisierung der Datentypen gelten, die in Bild 6 angegeben sind.

Allgemeiner Datentyp	Initialisiert durch Literal	Ergebnis
ANY_UNSIGNED	Nicht negatives ganzzahliges Literal oder nicht negativer konstanter Ausdruck	Nicht negativer ganzzahliger Wert
ANY_SIGNED	Ganzzahliges Literal oder konstanter Ausdruck	Ganzzahliger Wert
ANY_REAL	Numerisches Literal oder konstanter Ausdruck	Numerischer Wert
ANY_BIT	Vorzeichenloses ganzzahliges Literal oder vorzeichenloser konstanter Ausdruck	Vorzeichenloser ganzzahliger Wert
ANY_STRING	Zeichenfolge-Literal	Zeichenfolgewart
ANY_DATE	Datums- und Uhrzeitliteral	Datums- und Uhrzeitwert
ANY_DURATION	Zeitdauer-Literal	Zeitdauer-Wert

Bild 6 – Initialisierung durch Literale und konstante Ausdrücke (Regeln)

Tabelle 11 definiert die Eigenschaften der Deklaration von anwenderdefinierten Datentypen und Initialisierung.

Tabelle 11 – Deklaration von anwenderdefinierten Datentypen und Initialisierung (1 von 2)

Nr.	Beschreibung	Beispiel	Erklärung
1a 1b	Datentypen für Aufzählung (en: enumerated)	<pre> TYPE ANALOG_SIGNAL_RANGE: (BIPOLAR_10V, UNIPOLAR_10V, UNIPOLAR_1_5V, UNIPOLAR_0_5V, UNIPOLAR_4_20_MA, UNIPOLAR_0_20_MA) := UNIPOLAR_1_5V; END_TYPE </pre>	Initialisierung
2a 2b	Datentypen für benannte Werte (en: named values)	<pre> TYPE Colors: (Red := 16#00FF0000, Green := 16#0000FF00, Blue := 16#000000FF, White := Red OR Green OR Blue, Black := Red AND Green AND Blue) := Green; END_TYPE </pre>	Initialisierung
3a 3b	Datentypen für Bereich (en: subrange)	<pre> TYPE ANALOG_DATA: INT(-4095 .. 4095) := 0; END_TYPE </pre>	
4a 4b	Datentypen für Feld (en: array)	<pre> TYPE ANALOG_16_INPUT_DATA: ARRAY [1..16] OF ANALOG_DATA := [8(-4095), 8(4095)]; END_TYPE </pre>	ANALOG_DATA s. oben Initialisierung
5a 5b	Funktionsbaustein-Typen und Klassen als Feldelemente	<pre> TYPE TONs: ARRAY[1..50] OF TON := [50 (PT:=T#100ms)]; END_TYPE </pre>	FB TON als Feldelement-Initialisierung
6a 6b	Datentyp für Struktur (en: structure)	<pre> TYPE ANALOG_CHANNEL_CONFIGURATION: STRUCT RANGE: ANALOG_SIGNAL_RANGE; MIN_SCALE: ANALOG_DATA := -4095; MAX_SCALE: ANALOG_DATA := 4095; END_STRUCT; END_TYPE </pre>	ANALOG_SIGNAL_RANGE siehe oben
7a 7b	Funktionsbaustein-Typen und Klassen als Strukturelemente	<pre> TYPE Cooler: STRUCT Temp: INT; Cooling: TOF := (PT:=T#100ms); END_TYPE </pre>	FB TOF als Strukturelement

Tabelle 11 – Deklaration von anwenderdefinierten Datentypen und Initialisierung (2 von 2)

Nr.	Beschreibung	Beispiel	Erklärung
8a 8b	Strukturierter Datentyp mit relativer Adressierung AT	<pre> TYPE Com1_data: STRUCT head AT %B0: INT; length AT %B2: USINT := 26; flag1 AT %X3.0: BOOL; end AT %B25: BYTE; END_STRUCT; END_TYPE </pre>	Explizite Ablage ohne Überlappung
9a	Strukturierter Datentyp mit relativer Adressierung AT und OVERLAP	<pre> TYPE Com2_data: STRUCT OVERLAP head AT %B0: INT; length AT %B2: USINT; flag2 AT %X3.3: BOOL; data1 AT %B5: BYTE; data2 AT %B5: REAL; end AT %B19: BYTE; END_STRUCT; END_TYPE </pre>	Explizite Ablage mit Überlappung
10a 10b	Direkt dargestellte Elemente einer Struktur – teilweise festgelegt mit Verwendung von „*“	<pre> TYPE HW_COMP: STRUCT; IN AT %I*: BOOL; OUT_VAR AT %Q*: WORD := 200; ITNL_VAR: REAL := 123.0; // noch ohne Speicherort END_STRUCT; END_TYPE </pre>	Weist die Elemente einer Struktur den Eingängen und Ausgängen zu, die noch keinen Speicherort haben, siehe Anmerkung 2.
11a 11b	Direkt abgeleitete Datentypen	<pre> TYPE CNT: UINT; FREQ: REAL := 50.0; ANALOG_CHANNEL_CONFIG: ANALOG_CHANNEL_CONFIGURATION := (MIN_SCALE := 0, MAX_SCALE := 4000); END_TYPE </pre>	Initialisierung Neue Initialisierung
12	Initialisierung mit Verwendung von konstanten Ausdrücken	<pre> TYPE PIx2: REAL := 2 * 3.1416; END_TYPE </pre>	Verwendet einen konstanten Ausdruck
Die Datentypdeklaration ist ohne Initialisierung (Eigenschaft a) oder mit Initialisierung (Eigenschaft b) möglich. Wenn nur die Eigenschaft (a) unterstützt wird, muss der Datentyp mit dem voreingestellten Wert initialisiert werden. Wenn die Eigenschaft (b) unterstützt wird, muss der Datentyp mit dem angegebenen Wert initialisiert werden oder mit dem voreingestellten Anfangswert, wenn kein Anfangswert angegeben ist. ANMERKUNG Variable mit direkt dargestellten Elementen einer Struktur, die mit Verwendung von „*“ teilweise festgelegt sind, dürfen nicht in den VAR_INPUT- oder VAR_IN_OUT-Sektionen verwendet werden.			

6.4.4.2 Datentyp Aufzählung

6.4.4.2.1 Allgemeines

Die Deklaration eines Datentyps Aufzählung (en: enumeration) legt fest, dass der Wert jedes Datenelements dieses Typs nur einen der Werte annehmen kann, die in der zugehörigen Liste der Bezeichner angegeben sind, so wie es in Tabelle 11 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 herstellerspezifisch.

Um eine eindeutige Identifikation in einem bestimmten Kontext zu gewährleisten, dürfen aufgezählte Literale durch ein Präfix beschrieben werden, das aus ihrem zugehörigen Datentypnamen und dem '#'-Zeichen (Doppelkreuz) besteht, so wie die Literale mit Typangabe. 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 (siehe Beispiel unten).

BEISPIEL Datentyp-Aufzählung.

```
TYPE
  Traffic_light: (Red, Amber, Green);
  Painting_colors: (Red, Yellow, Green, Blue) := Blue;
END_TYPE

VAR
  My_Traffic_light: Traffic_light := Red;
END_VAR

IF My_Traffic_light = Traffic_light#Amber THEN ...           // OK
IF My_Traffic_light = Traffic_light#Red THEN ...             // OK
IF My_Traffic_light = Amber THEN ...                         // OK - Amber ist eindeutig
IF My_Traffic_light = Red THEN ...                           // FEHLER - Red ist nicht
eindeutig
```

6.4.4.2.2 Initialisierung

Der voreingestellte Initialwert eines Datentyps Aufzählung muss der erste Bezeichner in der zugehörigen Aufzählungsliste sein.

Der Anwender kann den Datentyp mit einem anwenderdefinierten Wert aus der Liste der aufgezählten Werte initialisieren. Diese Initialisierung hat Vorrang.

Wie in Tabelle 11 für `ANALOG_SIGNAL_RANGE` gezeigt, ist dem anwenderdefinierten Initialwert des Aufzählungsdantentyps der Wert `UNIPOLAR_1_5V` zugewiesen.

Die anwenderdefinierte Zuweisung des Initialwerts eines Datentyps ist eine Eigenschaft in Tabelle 11.

6.4.4.3 Datentyp mit benannten Werten

6.4.4.3.1 Allgemeines

Verwandt mit dem Datentyp Aufzählung – bei dem die Werte von aufgezählten Bezeichnern dem Anwender nicht bekannt sind – ist ein Aufzählungsdantentyp mit benannten Werten. Die Deklaration legt den Datentyp fest und weist die Werte von benannten Werten zu, so wie es in Tabelle 11 veranschaulicht ist.

Das Deklarieren benannter Werte begrenzt nicht die Verwendung des Wertebereichs der Variablen dieses Datentyps, d. h. es können weitere Konstante zugewiesen werden oder durch Berechnungen entstehen.

Um bei Verwendung in einem bestimmtem Kontext eine eindeutige Identifizierung zu ermöglichen, können gewählte Werte durch ein Präfix bestimmt werden, das aus dem zugehörigen Datentypnamen und dem '#'-Zeichen (Doppelkreuz) besteht, wie bei den typisierten Literalen.

Ein derartiges Präfix darf nicht innerhalb einer Deklarationsliste verwendet werden. Es ist ein Fehler, wenn nicht genügend Information bei einem aufgezählten Literal zur Verfügung steht, um seinen Wert eindeutig zu bestimmen (siehe Beispiel unten).

BEISPIEL Datentyp mit benannten Werten.

```

TYPE
  Traffic_light:  INT (Red := 1, Amber := 2, Green := 3) := Green;
END_TYPE

VAR
  My_Traffic_light: Traffic_light;
END_VAR

My_Traffic_light := 27;           // Zuweisung mit einer Konstanten
My_Traffic_light := Amber + 1;    // Zuweisung mit einem Ausdruck
                                // Anmerkung: Dies ist für Aufzählungswerte nicht
möglich
My_Traffic_light := Traffic_light#Red + 1;

IF My_Traffic_light = 123 THEN ...           // OK
IF My_Traffic_light = Traffic_light#Amber THEN ... // OK
IF My_Traffic_light = Traffic_light#Red THEN ... // OK
IF My_Traffic_light = Amber THEN ...         // OK, weil Amber eindeutig ist
IF My_Traffic_light = Red THEN ...           // Fehler, weil Rot nicht
eindeutig ist

```

6.4.4.3.2 Initialisierung

Der voreingestellte Wert eines Datentyps mit benannten Werten ist das erste Datenelement in der Aufzählungsliste. Im Beispiel oben für `Traffic_light` ist dieses Element `Red`.

Der Anwender kann den Datentyp mit einen anwenderdefinierten Wert initialisieren. Die Initialisierung ist nicht auf die benannten Werte beschränkt. Es darf jeder passende Wert innerhalb des Bereichs des Basisdatentyps verwendet werden. Diese Initialisierung hat Vorrang.

Im Beispiel ist der anwenderdefinierte Anfangswert des Datentyps Aufzählung für `Traffic_light` `Green`.

Die anwenderdefinierte Zuweisung des Anfangswerts des Datentyps ist eine Eigenschaft in Tabelle 11.

6.4.4.4 Datentyp Bereich

6.4.4.4.1 Allgemeines

Eine Deklaration von Bereich (en: subrange) legt fest, dass der Wert jedes Datenelements dieses Typs nur Werte zwischen und einschließlich den festgelegten oberen und unteren Grenzen annehmen kann, wie es Tabelle 11 veranschaulicht ist.

Die Grenzen eines Bereichs müssen Literale oder konstante Ausdrücke sein.

BEISPIEL

```

TYPE
  ANALOG_DATA: INT(-4095 .. 4095) := 0;
END_TYPE

```

6.4.4.4.2 Initialisierung

Die voreingestellten Anfangswerte für Datentypen mit Bereich müssen die erste (untere) Grenze des Bereichs sein.

Der Anwender kann den Datentyp mit einem anwenderdefinierten Wert aus dem Bereich initialisieren. Diese Initialisierung hat Vorrang.

Wie das Beispiel in Tabelle 11 zeigt, ist der voreingestellte Wert der Elemente vom Typ `ANALOG_DATA` ohne explizite Initialisierung `-4095` während mit impliziter Initialisierung der voreingestellte Anfangswert (wie deklariert) gleich null wäre.

6.4.4.5 Datentyp Feld

6.4.4.5.1 Allgemeines

Die Deklaration eines Datentyps Feld (en: Array) legt fest, dass eine ausreichende Menge Datenspeicher für alle Elemente dieses Typs ausgewiesen sein muss, so dass alle Daten, die durch den (die) festgelegten Indexbereich(e) indiziert werden können, wie es in Tabelle 11 veranschaulicht ist, gespeichert werden können.

Ein Feld ist eine Sammlung von Datenelementen desselben Typs. Als Typ eines Feldelements können elementare und anwenderdefinierte Datentypen, Funktionsbaustein-Typen und Klassen verwendet werden. Diese Sammlung von Datentypen wird durch einen Index oder mehrere Indizes adressiert, die in eckige Klammern eingeschlossen und durch Kommas getrennt sind. Es muss ein Fehler sein, wenn der Wert eines Index außerhalb des Bereichs liegt, der in der Feld-Deklaration festgelegt ist.

ANMERKUNG Dieser Fehler kann für berechnete Indizes nur während der Laufzeit entdeckt werden.

Die maximale Anzahl der Feldindizes, die maximalen Feldgröße und der maximale Bereich der Indexwerte sind herstellerspezifisch.

Die Grenzen des (der) Indexbereich(e) müssen ein Literal oder ein konstanter Ausdruck sein. Felder mit variabler Länge sind in 6.5.3 definiert.

In der Sprache ST muss eine Index ein Ausdruck sein, der einen Wert liefert, der zu einem der Untertypen des allgemeinen Typs `ANY_INT` gehört.

Die Form von Indizes in der Sprache AWL und in den grafischen Sprachen, die in Abschnitt 8 definiert sind, ist auf Einzelelement-Variable oder ganzzahlige Literale beschränkt.

BEISPIEL

a) Deklaration eines Feldes

```
VAR myANALOG_16: ARRAY [1..16] OF ANALOG_DATA  
    := [8(-4095), 8(4095)];    // Anwenderdefinierte Anfangswerte  
END_VAR
```

b) Eine Anwendung von Feld-Variablen in der Sprache ST könnte sein:

```
OUTARY[6, SYM] := INARY[0] + INARY[7] - INARY[i] * %IW62;
```

6.4.4.5.2 Initialisierung

Der voreingestellte Anfangswert jedes Feldelements ist der Anfangswert, der für den Datentyp der Feldelemente definiert wurde.

Der Anwender kann einen Felddatentyp mit einem anwenderdefinierten Wert initialisieren. Dieser Wert hat Vorrang.

Der anwenderdefinierte Anfangswert eines Feldes wird in Form einer Liste zugewiesen, die runde Klammern verwenden darf, um Wiederholungen auszudrücken.

Bei der Initialisierung der Felddatentypen muss der ganz rechtsstehende Index am schnellsten laufen, um das Feld mit den Liste der Anfangswerte aufzufüllen.

BEISPIEL Initialisierung eines Feldes.

```
A: ARRAY [0..5] OF INT := [2(1, 2, 3)]
    entspricht der Initialisierungssequenz 1, 2, 3, 1, 2, 3
```

Wenn die Anzahl der Anfangswerte, die in der Initialisierungsliste angegeben ist, die Anzahl der Feldeinträge überschreitet, muss der Zugriff auf die ganz rechts stehenden Anfangswerte ignoriert werden. Wenn die Anzahl der Anfangswerte kleiner als die Anzahl der Feldeinträge ist, müssen die restlichen Feldeinträge mit den voreingestellten Anfangswerten des entsprechenden Datentyps aufgefüllt werden. In jedem Fall muss der Anwender, wenn er das Programm für die Ausführung vorbereitet, bei der Programmerstellung für die Ausführung vor dieser Situation gewarnt werden.

Die anwenderdefinierte Zuweisung des Anfangswerts des Datentyps ist eine Eigenschaft in Tabelle 11.

6.4.4.6 Datentyp Struktur

6.4.4.6.1 Allgemeines

Die Deklaration eines strukturierten Datentyps (**STRUCT**) legt fest, dass dieser Datentyp eine Sammlung von Unterelementen der festgelegten Typen beinhalten muss, auf die mit den festgelegten Namen zugegriffen werden kann, wie es in Tabelle 11 veranschaulicht ist.

Ein Element eines strukturierten Datentyps muss durch zwei oder mehr Bezeichner oder Feldzugriffe, die durch einfache Punkte getrennt „.“ sind, dargestellt werden. Der erste Bezeichner stellt den Namen des strukturierten Elements dar, und die folgenden Bezeichner stellen die Sequenz von Elementnamen dar, um auf das bestimmte Datenelement innerhalb der Struktur zugreifen zu können. Als Typ eines Strukturelements können elementare und anwenderdefinierte Datentypen, Funktionsbaustein-Typen und Klassen verwendet werden.

Zum Beispiel enthält ein Element des Typs **ANALOG_CHANNEL_CONFIGURATION**, wie in Tabelle 11 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 enthalten sein kann, und die maximale Anzahl von geschachtelten Ebenen von Strukturelementen sind herstellerspezifisch.

Zwei strukturierte Variable sind bei der Zuweisung nur dann kompatibel, wenn sie denselben Datentyp haben.

BEISPIEL Deklaration und Gebrauch eines strukturierten Datentyps und einer strukturierten Variable.

a) Deklaration eines strukturierten Datentyps

```
TYPE
  ANALOG_SIGNAL_RANGE: (BIPOLAR_10V, UNIPOLAR_10V);
  ANALOG_DATA: INT (-4095 .. 4095);
  ANALOG_CHANNEL_CONFIGURATION:
    STRUCT
      RANGE:      ANALOG_SIGNAL_RANGE;
      MIN_SCALE: ANALOG_DATA;
      MAX_SCALE: ANALOG_DATA;
    END_STRUCT;
END_TYPE
```

b) Deklaration einer strukturierten Variable

```
VAR
  MODULE_CONFIG: ANALOG_CHANNEL_CONFIGURATION;
  MODULE_8_CONF: ARRAY [1..8] OF ANALOG_CHANNEL_CONFIGURATION;
END_VAR
```

c) Gebrauch von strukturierten Variablen in der ST-Sprache

```
MODULE_CONFIG.MIN_SCALE := -2047;
MODULE_8_CONF[5].RANGE := BIPOLAR_10V;
```

6.4.4.6.2 Initialisierung

Die voreingestellten Werte von Komponenten einer Struktur werden durch ihre jeweiligen Datentypen angegeben.

Der Anwender kann die Komponenten der Struktur mit anwenderdefinierten Werten initialisieren. Diese Initialisierung hat Vorrang.

Der Anwender kann auch eine vorher definierte Struktur initialisieren, indem er eine Liste von Zuweisungen an die Komponenten der Struktur verwendet. Diese Initialisierung hat eine höhere Priorität als die voreingestellte Initialisierung und die Initialisierung der Komponenten.

BEISPIEL Initialisierung einer Struktur.

a) Deklaration mit Initialisierung eines strukturierten Datentyps

```
TYPE
  ANALOG_SIGNAL_RANGE:
    (BIPOLAR_10V,
     UNIPOLAR_10V) := UNIPOLAR_10V;
  ANALOG_DATA: INT (-4095 .. 4095);
  ANALOG_CHANNEL_CONFIGURATION:
    STRUCT
      RANGE: ANALOG_SIGNAL_RANGE;
      MIN_SCALE: ANALOG_DATA := -4095;
      MAX_SCALE: ANALOG_DATA := 4096;
    END_STRUCT;
  ANALOG_8BI_CONFIGURATION:
    ARRAY [1..8] OF ANALOG_CHANNEL_CONFIGURATION
      := [8 ((RANGE:= BIPOLAR_10V))];
END_TYPE
```

b) Deklaration mit Initialisierung einer strukturierten Variable

```
VAR
  MODULE_CONFIG: ANALOG_CHANNEL_CONFIGURATION
    := (RANGE:= BIPOLAR_10V, MIN_SCALE:= -1023);
  MODULE_8_SMALL: ANALOG_8BI_CONFIGURATION
    := [8 ((MIN_SCALE:= -2047, MAX_SCALE:= 2048))];
END_VAR
```

6.4.4.7 Relativer Speicherort für Elemente von strukturierten Datentypen (AT)

6.4.4.7.1 Allgemeines

Die Speicherorte (Adressen) von Elementen eines strukturierten Datentyps können relativ zum Beginn der Struktur definiert sein.

In diesem Fall muss der Name jeder Komponente dieser Struktur vom Schlüsselwort `AT` und einem relativen Speicherort (en: location) gefolgt werden. Die Deklaration darf Lücken in der Speicherbelegung enthalten.

Der relative Speicherort besteht aus einem '%' (Prozent), dem Bestimmungszeichen für den Speicherort, und einem Bit- oder Byte-Speicherort. Der Byte-Speicherort ist ein vorzeichenloses ganzzahliges Literal, das den Byte-Offset angibt. Der Bit-Speicherort besteht aus einem Byte-Offset, gefolgt von einem '.' (Punkt) und dem Bit-Offset als vorzeichenloses ganzzahliges Literal von 0 bis 7. Leerzeichen innerhalb des relativen Speicherorts sind nicht zulässig.

Die Komponenten der Struktur dürfen bei ihrer Speicherbelegung nicht überlappen, es sei denn, das Schlüsselwort `OVERLAP` ist in der Deklaration angegeben.

Das Überlappen von Folgen liegt nicht im Anwendungsbereich dieser Norm.

ANMERKUNG Das Zählen von Bit-Offsets beginnt mit 0 beim ganz rechten Bit. Das Zählen von Byte-Offsets beginnt am Anfang der Struktur mit dem Byte-Offset 0.

BEISPIEL Relativer Speicherort und Überlappen in einer Struktur.

```

TYPE
Com1_data: STRUCT
  head      AT %B0:    INT;      // Am Speicherort 0
  length.    AT %B2:    USINT := 26; // Am Speicherort 2
  flag1      AT %X3.0:  BOOL;     // Am Speicherort 3.0
  end        AT %B25:   BYTE;     // Bei 25, eine Lücke lassen
END_STRUCT;

Com2_data: STRUCT OVERLAP
  head      AT %B0:    INT;      // Am Speicherort 0
  length     AT %B2:    USINT;     // Am Speicherort 2
  flag2      AT %X3.3:  BOOL;     // Am Speicherort 3.3
  data1      AT %B5:    BYTE;     // Am Speicherort 5, überlappt
  data2      AT %B5:    REAL;     // Am Speicherort 5 bis 8
  end        AT %B19:   BYTE;     // Bei 19, eine Lücke lassen
END_STRUCT;

Com_data: STRUCT OVERLAP          // C1 und C2 überlappen sich
  C1 AT %B0:  Com1_data;
  C2 AT %B0:  Com2_data;

END_STRUCT;
END_TYPE

```

6.4.4.7.2 Initialisierung

Überlappende Strukturen können nicht explizit initialisiert werden.

6.4.4.8 Direkt dargestellte Komponenten einer Struktur – partiell festgelegt mit „*“

Die Angabe eines Sternchen „*“ (en: asterisk) in Tabelle 11 kann verwendet werden, um noch nicht voll festgelegte Speicherorte für direkt dargestellte Komponenten einer Struktur zu kennzeichnen.

BEISPIEL Zuweisung der Komponenten einer Struktur an Eingänge und Ausgänge, für die noch keine Speicherorte festgelegt sind.

```
TYPE
    HW_COMP: STRUCT;
    IN AT %I*: BOOL;
    VAL AT %I*: DWORD;
    OUT AT %Q*: BOOL;
    OUT_VAR AT %Q*: WORD;
    ITNL_VAR: REAL; // ohne Speicherort
END_STRUCT;
END_TYPE
```

Falls eine direkt dargestellte Komponente einer Struktur in einer Speicherort-Zuweisung im Deklarationsteil eines Programms, eines Funktionsbaustein-Typs oder einer Klasse verwendet wird, muss ein Stern „*“ an Stelle des Präfix für die Größe verwendet werden und eine oder mehrere vorzeichenlose ganze Zahlen in der Aneinanderreihung, um anzuzeigen, dass die direkte Darstellung noch nicht vollständig festgelegt ist.

Die Verwendung dieser Eigenschaft erfordert, dass der Speicherort der so deklarierten strukturierten Variable vollständig im VAR_CONFIG...END_VAR-Konstrukt der Konfiguration für jede Instanz des einschließenden Typs festgelegt sein muss.

Variablen dieses Typs dürfen nicht in einer VAR_INPUT, VAR_IN_OUT- oder VAR_TEMP-Sektion verwendet werden.

Es ist ein Fehler, wenn eine der vollständigen Festlegungen im VAR_CONFIG...END_VAR-Konstrukt für eine unvollständige Adressfestlegung fehlen, die durch die Angabe des Sterns „*“ in einer Instanz von Programmen oder Funktionsbaustein-Typen ausgedrückt wird, die solche unvollständigen Festlegungen enthalten.

6.4.4.9 Direkt abgeleiteter Datentyp

6.4.4.9.1 Allgemeines

Ein anwenderdefinierter Datentyp darf direkt von einem elementaren Datentyp oder einem vorher vom Anwender definierten Datentyp abgeleitet werden.

Dies kann zur Festlegung von neuen, typspezifischen Anfangswerten verwendet werden.

BEISPIEL Direkt abgeleiteter Datentyp.

```
TYPE
    myInt1123: INT := 123;
    myNewArrayType: ANALOG_16_INPUT_DATA := [8(-1023), 8(1023)];
    Com3_data: Com2_data := (head := 3, length :=40);
END_TYPE

R1: REAL := 1.0;
R2: R1;
```

6.4.4.9.2 Initialisierung

Der voreingestellte Anfangswert ist der Anfangswert des Datentyps, von dem der neue Typ abgeleitet wurde.

Der Anwender kann den Datentyp mit einem anwenderdefinierten Datentyp initialisieren. Diese Initialisierung hat Vorrang.