Lesezeichen ein-/ausblenden

Lenze

Handbuch

Global Drive

Drive PLC Developer Studio

Einführung in die
IEC 61131-3 Programmierung

Wichtiger Hinweis:

Die Software wird dem Benutzer in der vorliegenden Form zur Verfügung gestellt. Alle Risiken hinsichtlich der Qualität und der durch ihren Einsatz ermittelten Ergebnisse verbleiben beim Benutzer. Entsprechende Sicherheitsvorkehrungen gegen eventuelle Fehlbedienungen sind vom Benutzer vorzusehen.

Wir übernehmen keine Verantwortung für direkt oder indirekt entstandene Schäden, z. B. Gewinnverluste, Auftragsverluste oder geschäftliche Beeinträchtigungen jeglicher Art.

© 2003 Lenze Drive Systems GmbH

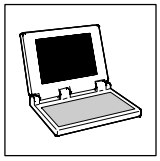
Ohne besondere schriftliche Genehmigung von Lenze Drive Systems GmbH darf kein Teil dieser Dokumentation vervielfältigt oder Dritten zugänglich gemacht werden.

Wir haben alle Angaben in dieser Dokumentation mit größter Sorgfalt zusammengestellt und auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Trotzdem können wir Abweichungen nicht ganz ausschließen. Wir übernehmen keine juristische Verantwortung oder Haftung für Schäden, die dadurch eventuell entstehen.Notwendige Korrekturen werden wir in die nachfolgenden Auflagen einarbeiten.

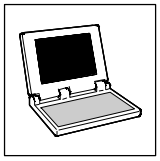
Alle in dieser Dokumentation aufgeführten Markennamen sind Warenzeichen ihrer jeweiligen Besitzer.

Version 2.0 01/2003 TD05

Einführung in die IEC 61131-3 Programmierung



Inhalt



1 Vorwort und Allgemeines

Dieses Handbuch gibt Ihnen einen kurzen Überblick über die Norm IEC 61131-3.

Mit der Norm IEC 61131-3 wurde eine Basis für eine einheitliche SPS-Programmierung geschaffen, die es dem Anwender ermöglichen soll,

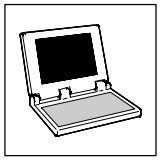
- ausgetestete und standardisierte Software-Bausteine wiederzuverwenden,
- Softwareengineering-Methoden für die Erstellung dieser Bausteine einzusetzen,
- Problemlösungsansätze ganzheitlich zu betrachten,
- komplexe Aufgaben in überschaubare Module zu abstrahieren,
- eindeutige Schnittstellen zu definieren,
- Programme durch einen einheitlichen Sprachumfang leichter auf andere Systeme portieren zu können.

Die im Drive PLC Developer Studio realisierten Programmiersprachen sind konform zu den Anforderungen der Norm IEC 61131-3.

Neben den deutschsprachigen Begriffen werden in dieser Einführung oftmals auch die internationalen Begriffe (*kursiv in Klammern*) mit aufgeführt.

1.1 Weitere Informationen zur IEC 61131-3-Programmierung

- finden Sie im Handbuch zum **Drive PLC Developer Studio**
- finden Sie im Internet auf der Homepage der **PLCopen**: www.plcopen.org



2 Das Softwaremodell

Im Softwaremodell der IEC 61131-3 werden die Begriffe Konfiguration, Ressource, Task, Programm, Funktionsblock und Funktion und deren Zusammenhang beschrieben.

Prinzipiell geht die Norm bei der Definition der Begriffe von einer maximal leistungsfähigen SPS mit folgenden Eigenschaften aus:

- Multiprozessor-fähig
- Multitasking-fähig
- unbegrenzte Anzahl analoger und digitaler Ein- und Ausgänge
- Kommunikationsfähigkeit zu anderen SPS bzw. PC

2.1 Ressourcen innerhalb einer Konfiguration

Die oberste Ebene im Softwaremodell bildet die **Konfiguration** (*Configuration*), die die Struktur des Gerätes definiert. Dieses Gerät kann z. B. eine SPS mit mehreren vernetzten Zentraleinheiten (CPUs) sein.

Eine Konfiguration beinhaltet eine oder mehrere **Ressourcen** (*Resources*), die eine Steuerungseinheit (CPU) darstellen.

Die der Ressource zugehörigen Programme werden durch **Tasks** gesteuert, die eine ablauffähige Programmeinheit darstellen.

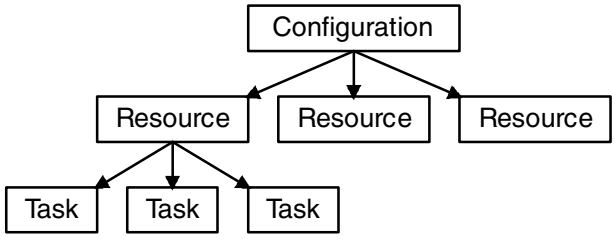


Abb. 1 Eine Konfiguration mit mehrerer Ressourcen, die wiederum unabhängige Tasks beinhalten können.

2.1.1 Tasks

Tasks können periodisch oder aufgrund eines Ereignisses abgearbeitet werden, zudem werden sie mit einer Priorität versehen, durch die die Zuteilung von CPU-Zeiten innerhalb der Ressource erfolgt.

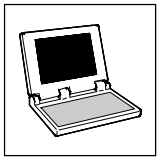
Es werden folgende Arten von Tasks unterschieden:

- Zyklische Task
- Zeitgesteuerte Task (*INTERVAL-Task*)
- Ereignisgesteuerte Task (*EVENT-Task*)
- Interrupt-Task

Die Deklarierung einer Task besteht aus dem Namen der Task, einem Eintrag für die Priorität, die die Task haben soll, und einem Eintrag für die Bedingung, unter der die Task ausgeführt werden soll.

Die Bedingung kann entweder ein Zeitintervall sein, nach dem die Task ausgeführt werden soll, ein Ereignis (steigende Flanke am digitalen Eingang bzw. **`FALSE`**/**`TRUE`**-Wechsel einer globalen Variable) oder ein Interrupt.

Zu jeder Task können Sie nun eine Folge von Programmen angeben, die von der Task aufgerufen werden sollen. Diese Programme werden in der angefügten Reihenfolge abgearbeitet.



Für die Ausführung von Tasks gelten folgende Regeln:

- Es wird die Task ausgeführt, deren Bedingung zutrifft, das heißt, wenn z. B. die angegebene Intervallzeit abgelaufen ist oder bei Ereignissteuerung die angewählte Variable von **`FALSE`** auf **`TRUE`** wechselt.
- Haben mehrere Tasks eine gültige Bedingung, dann wird die Task mit der höchsten Priorität ausgeführt.
- Mehrere Tasks mit gleicher Priorität sind nicht möglich.
  (Ausnahme: Priorität 0 = Task gesperrt)
- Wenn während der Abarbeitung einer Task die Bedingung für eine Task höherer Priorität zutrifft, so wird die Task mit der niedrigeren Priorität unterbrochen und erst nach Beendigung der höher prioren Task wieder fortgesetzt.

2.2 Programmorganisationseinheiten, POEs

In der IEC1131-3 bezeichnet man Programme (*Programs*), Funktionsblöcke (*Function Blocks*) und Funktionen (*Functions*) als Programmorganisationseinheiten bzw. POEs (*Program Organization Units, POUs*).

Die Eigenschaften einer POE ermöglichen eine weitreichende Modularisierung des Anwenderprogramms und eine Wiederverwendung bereits implementierter und getesteter Software-Bausteine. Um Programmodulen den Zugriff auf POEs zu ermöglichen, wird mindestens die Deklaration der Aufrufschnittstelle benötigt. Eine POE steht nach ihrer Deklaration allen anderen POEs global zur Verfügung.

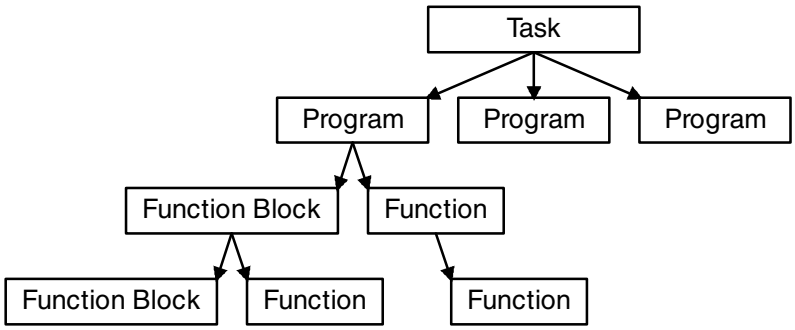
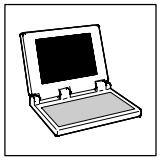


Abb. 2 Strukturierung von Programmorganisationseinheiten (POEs) in Programme, Funktionsblöcke und Funktionen

2.2.1 Programme

Durch die Zuordnung von Programmen zu einer Task werden die Laufzeiteigenschaften des Gesamtprogramms definiert, welches selbständig in einer CPU ablaufen kann. Ein Programm kann zu mehreren Tasks gehören, d. h. es werden mehrere Instanzen des Programms mit unterschiedlichen Laufzeiteigenschaften erzeugt.



2.2.2 Funktionsblöcke

Um typische SPS-Funktionalitäten zu standardisieren, führt die Norm IEC 61131-3 Standard-Funktionen und Funktionsblöcke ein. Diese "Standard-Bibliothek" bildet eine wichtige Grundlage für eine einheitliche, herstellerunabhängige Programmierung von SPS-Systemen.

Funktionsblöcke (FBs) können mit integrierten Schaltungen verglichen werden, die eine bestimmte Steuerungsfunktion beinhalten. Sie werden dazu verwendet, Ein-/Ausgänge sowie interne Variable zu setzen, dabei werden die Zustände eines Funktionsblockaufrufs von Zyklus zu Zyklus zwischengespeichert. Vom aufrufenden Programm sind nur die Ein- und Ausgangsvariablen des Funktionsblocks ansprechbar. Ein Funktionsblock kann von einem anderen Funktionsblock aufgerufen werden.

Instanzierung von Funktionsblöcken

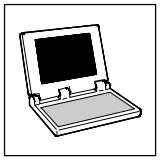
In der IEC 61131-3 ist die Instanzierung von Funktionsblöcken vorgesehen. Eine Instanz ist eine Struktur, in der beim Aufruf eines FBs alle internen sowie Ein- und Ausgabevariablen gespeichert werden.

Ein Programm, welches FB1 dreimal aufruft, besitzt drei Instanzen des FB1, eine für jeden Aufruf. Durch diese objektorientierte Vorgehensweise kann die Evaluierung des Programms aufrufgenau und seiteneffektfrei erfolgen. Zu beachten ist hierbei, daß alle Instanzen den gleichen Programmcode verwenden, d. h. Änderungen am Programmcode wirken sich in allen drei Aufrufen gleich aus.

Moderne Software-Tools wie das **Drive PLC Developer Studio** helfen mit einer automatischen Deklaration, diese Instanzierung durchzuführen: für einen Aufruf eines FB wird einfach ein Instanzname festgelegt, der die Datenstruktur dieses Aufrufs verwaltet.

Übersicht IEC 61131-3 Standard-Funktionsblöcke

Bistabile Funktionsblöcke	
SR/RS	Bistabiler Funktionsblock (dominant setzen/zurücksetzen)
SEMA	Software-Semaphor (unterbrechbar)
Flankenerkennung	
R_TRIG/F_TRIG	Detektor für eine ansteigende Flanke//fallende Flanke
Zähler	
CTU/CTD	Aufwärtszähler/Abwärtszähler
CTUD	Auf- und Abwärtszähler
Timer	
TP	Pulsgeber
TON/TOF	Timer on-delay/Timer off-delay

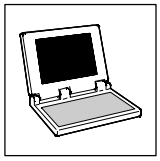


2.2.3 Funktionen

Im Gegensatz zu FBs können Funktionen ihre internen Werte nicht zwischenspeichern und dürfen daher keine globalen Variablen verwenden, auf Funktionsbausteine zugreifen und direkt adressierte Variablen deklarieren. Allen Funktionen gemeinsam ist, daß sie bei gleichen Eingangsparametern immer die gleichen Ausgangsparameter zurückliefern.

Übersicht IEC 61131-3 Standard-Funktionen

Typkonvertierungsfunktionen	
..._TO_...	Konvertierungen zwischen ganzzahligen Zahlentypen
BOOL_TO	BOOL ⇒ Typ X
TO_BOOL	Typ X ⇒ BOOL
TIME_TO / TIME_OF_DAY	TIME / TIME_OF_DAY ⇒ Typ X
DATE_TO / DT_TO	DATE / DATE_AND_TIME ⇒ Typ X
STRING_TO	STRING ⇒ Typ X
TRUNC	REAL ⇒ INT
Numerische Funktionen	
ABS	Absolutwert
SQRT	Quadratwurzel
LN	natürlicher Logarithmus
LOG	Logarithmus zur Basis 10
EXP	Exponentialfunktion
SIN	Sinusberechnung in rad
COS	Cosinusberechnung in rad
TAN	Tangensberechnung in rad
ASIN	Arcussinusberechnung in rad
ACOS	Arcuscosinusberechnung in rad
ATAN	Arcustangensberechnung in rad
EXPT	Potenzierung einer Variablen mit einer anderen
STRING-Funktionen	
LEN	gibt die Länge eines Strings aus
LEFT	liefert einen linken Anfangsstring eines Strings
RIGHT	liefert einen rechten Anfangsstring eines Strings
MID	liefert einen Teilstring eines Strings
CONCAT	Konkatenation (Aneinanderhängen) von zwei Strings
INSERT	fügt einen String ab einer bestimmten Stelle in einen anderen ein
DELETE	löscht ab einer bestimmten Stelle einen Teilstring aus einem String
REPLACE	ersetzt einen Teilstring eines Strings durch einen anderen String
FIND	sucht einen Teilstring in einem String



Übersicht IEC 61131-3 Standard-Operatoren

Arithmetische Operatoren	
ADD	Addition
MUL	Multiplikation
SUB	Subtraktion
DIV	Division
MOD	Restbildung
EXP	Exponentiation
MOVE	Zuweisung
Bit-Shift-Operatoren	
SHL	Schieben nach links
SHR	Schieben nach rechts
ROR	Rotieren nach rechts
ROL	Rotieren nach links
Bit-String-Operatoren	
AND	Bitweises AND von Bit -Operanden
OR	Bitweises OR von Bit-Operanden
XOR	Bitweises XOR von Bit -Operanden
NOT	Bitweises NOT eines Bit Operanden
Auswahloperatoren	
SEL	Binäre Selektion
MAX	Maximum-Bildung
MIN	Minimum-Bildung
LIMIT	Begrenzung
MUX	Multiplexer
Vergleichsoperatoren	
GT	Größer als
LT	Kleiner als
LE	Kleiner oder gleich
GE	Größer oder gleich
EQ	Gleichheit
NE	Ungleichheit

2.3 Wiederanlaufverhalten der Steuerung

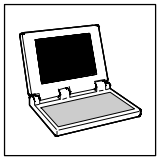
Ebenfalls im Softwaremodell der IEC 61131-3 definiert ist das Wiederanlaufverhalten der Steuerung.

Kaltstart

Bei einem Kaltstart wird das Programm neu geladen. Alle Variablen werden auf Ihren Initialwert gesetzt. Entweder wird ein Standard-Initialwert (z. B. 0 bzw. **FALSE**) oder der in der Variablen-Deklaration (optional) angegebene Initialwert gesetzt. Alle Tasks der Ressource werden gestartet.

Warmstart

Bei einem Warmstart (Wiederanlauf) werden die Variablen nicht auf Ihren Initialwert gesetzt, sondern es werden die vor der Unterbrechung im Speicher vorhandenen Werte übernommen.



3 Das Kommunikationsmodell

Das Kommunikationsmodell der IEC 61131-3 beschreibt den Datenaustausch der Elemente einer Konfiguration über

- Zugriffspfade
- Globale Variable
- Aufrufparameter
- Kommunikationsbausteine (IEC 61131-5)

Durch diese eindeutig definierten Schnittstellen soll insbesondere die Modularisierung und dadurch Wiederverwendbarkeit von Programmteilen gefördert werden.

3.1 Zugriffspfade

Definierte Zugriffspfade dienen den Elementen einer Konfiguration zur Kommunikation untereinander und mit weiteren SPS-Systemen.

3.2 Globale Variable

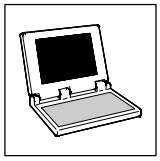
Globale Variable dienen der einfachen Kommunikation von Programmen untereinander und können innerhalb einer Konfiguration, einer Ressource und eines Programms deklariert und verwendet werden.

3.3 Aufrufparameter

Innerhalb von Programmen erfolgt der Datenaustausch über Aufrufparameter, also Ein-/Ausgangsvariablen. Durch Aufrufparameter werden eindeutige Schnittstellen zur Übergabe von Werten definiert.

3.4 Kommunikationsbausteine

Kommunikationsbausteine stellen Kommunikationsdienste zur Verfügung, die im Teil 5 der IEC 61131 definiert sind, der sich zur Zeit aber noch in Bearbeitung befindet.



4 Allgemeine Sprachelemente

Zu den allgemeinen Sprachelementen der IEC 61131-3 gehören Bezeichner, Schlüsselworte, Kommentare, Literale, Datentypen sowie Variable, die in den folgenden Unterkapiteln erläutert werden.

4.1 Bezeichner

Über Bezeichner werden Variablen, Funktionen, Programme usw. angesprochen, sie stellen also den Namen des entsprechenden Elements dar und fördern bei sinnvoller Auswahl die Lesbarkeit von Programmen.

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

Ein Bezeichner darf

- keine Leerstellen und Umlaute enthalten,
- nicht doppelt deklariert werden,
- und nicht identisch mit Schlüsselwörtern sein.(Kapitel 4.2)

Tip!

Die Konventionen, die für die Bezeichner der Variablen von Lenze Systembausteinen, Funktionsblöcken und Funktionen verwendet werden, sind im Anhang aufgeführt. (20)

4.2 Schlüsselwörter

Schlüsselwörter sind eindeutige Kombinationen von Zeichen, die als individuelle Syntax-Elemente angewendet werden.

- Schlüsselwörter dürfen nicht als Bezeichner verwendet werden.

Beispiele für Schlüsselwörter der IEC 61131-3

ABS, **SIN**, **BOOL**, **FALSE**, **TRUE**, **FOR**, **NEXT**, **IF**, **THEN**, **VAR**, **GLOBAL**, **DATE**, **TIME**, **FUNCTION**

4.3 Kommentare

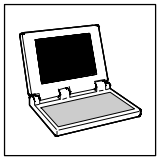
Kommentare zu Programmteilen fördern dessen Nachvollziehbarkeit und dienen somit als wichtiges Dokumentationsmittel. Kommentare sind in allen Texteditoren und dort an beliebiger Stelle erlaubt und müssen in die speziellen Zeichenfolgen (* und *) eingeschlossen werden. Auch können zu jedem Netzwerk Kommentare eingegeben werden, um dessen Funktionalität zu dokumentieren.

4.4 Literale

Ein Literal in der IEC 61131-3 ist entweder eine Zeichenfolge, eine Zahl oder eine Zeitangabe.

Zeichenfolgen

Zeichenfolge-Literale sind Sequenzen von 0 oder mehr Zeichen, die durch einfache Anführungszeichen eingeschlossen sind (z. B. `'Zeichenfolge'`).



Zahlen

Es gibt zwei Arten von numerischen Literalen, ganzzahlige Literale (Integer) und reelle Literale (Real).

Ganzzahlige Literale können mit einer Basis definiert werden, dezimale Zahlen können desweiteren Vorzeichen (+ oder -) enthalten. Bei reellen Literalen ist auch die Angabe mit Exponent möglich.

	Kennzeichnung	Beispiel
Ganzzahlige Literale (Integer)		
dezimal		10
binär	2#	2#1010
oktal	8#	8#12
hexadezimal	16#	16#A
Reelle Literale (Real)		
ohne Exponent		-12.50
mit Exponent	E	15.7E4

Zeiten

Es gibt zwei Arten von zeitlichen Literalen, Zeitdauer und Tageszeit/Datum.

	Kennzeichnung	Beispiel
Zeitdauer	T# oder TIME#	T#10ms
Tageszeit/Datum		
Datum	D# oder #DATE	D#1999-08-29
Tageszeit	TOD# oder #TIME_OF_DAY	TOD#15:36:30
Datum und Tageszeit	DT# oder #DATE_AND_TIME	DT#1999-08-29-15:36:30

4.5 Datentypen

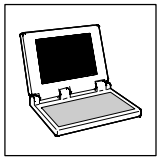
In der IEC 61131-3 sind verschiedene Standard-Datentypen definiert, aus denen abgeleitete und benutzerdefinierte Datentypen zusammengestellt werden können. Jedem Bezeichner wird ein Datentyp zugeordnet, der festlegt, wieviel Speicherplatz reserviert wird und welche Werte dem Speicherinhalt entsprechen.

Standard-Datentypen:

- `BOOL` (Wahrheitswerte `TRUE`/`FALSE`)
- `BYTE`, `WORD`, `DWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT` (Ganzzahlige Datentypen)
- `REAL` (Gleitpunktdatentyp)
- `STRING` (Zeichenkette)
- `TIME`, `TIME_OF_DAY`, `DATE`, `DATE_AND_TIME` (Zeitdatentypen)

Definierte Datentypen:

- `ARRAY` (ein-, zwei-, dreidimensionales Feld)
- `POINTER` (beinhaltet Adresse von Variable/Funktionsblock zur Laufzeit des Programms)
- Aufzählungstyp (*Enumerated*, besteht aus einer Menge von Stringkonstanten)
- `STRUCT` (Struktur)
- Referenz (erzeugt alternativen Namen für Variable/Konstante/Funktionsblock)



4.6 Variable

In der IEC 61131-3 sind fünf unterschiedliche Variablenklassen definiert:

- Globale Variablen
- Lokale Variablen
- Eingabevariablen
- Ausgabevariablen
- Ein- und Ausgabevariablen

Während Lokale Variable keine Verbindung nach außen haben, also nur innerhalb des Programmteils auf sie zugegriffen werden kann, sind Globale Variable von allen POEs aus ansprechbar.

Eingabe, Ausgabe- sowie Ein-/Ausgabevariablen werden bezogen auf ein Programm, eine Funktion oder einen Funktionsblock. Innerhalb der zugeordneten POE können sie lesend und schreibend verändert werden, außerhalb nur in der definierten Weise (Eingabe, Ausgabe bzw. Ein- und Ausgabe).

Die Deklaration der Variablen erfolgt im Quelltext zwischen den Schlüsselwörtern **VAR** und **END_VAR**. Grundsätzlich wird nach einem Kaltstart jede Variable initialisiert. Der Grundwert (Default) beträgt üblicherweise 0 oder **FALSE**. Eine anwenderspezifische Initialisierung mit einem anderen Wert ist mit dem Zeichen ":=" innerhalb der Deklaration möglich.

Variablen-Attribute:

Bei der Deklaration einer Variable können zusätzlich folgende Attribute verwendet werden:

- **RETAIN**: Diese Variablen behalten ihren Wert auch nach einem Stromausfall. Bei erneutem Start des Programms wird mit den gespeicherten Werten weitergearbeitet.
- **CONSTANT**: Variablenwerte können nicht geändert werden.
- **AT**: Variablen besitzen einen festen Platz im Speicherabbild (feste Adresse)

Beispiel: Deklaration einer Ausgabevariable mit Initialisierungswert

```
VAR_OUTPUT
  par_out1 : INT := 10;     (* Output-Parameter 1 mit Startwert 10 *)
END_VAR
```

Fest adressierte Variablen

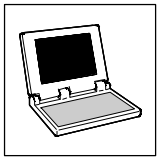
Bei der Deklaration können Variablen einem physikalischen Speicherort (SPS-Adresse) mit dem Schlüsselwort **AT** zugeordnet werden.

Der Aufbau der Adreßangabe erfolgt mittels spezieller Zeichenreihen. Die Zeichenreihe beginnt mit einem Prozentzeichen "%", gefolgt von einem Bereichspräfix und einem Präfix (Datentyp) für die Größe, und endet mit einer Zahlenfolge, die den direkten Speicherort angibt.

Bereichspräfixe: **I** (Eingang), **Q** (Ausgang), **M** (Merker, interner Speicherbereich)

Präfix für die Größe: **X** (Einzelbit), **B** (Byte, 8 Bits), **W** (Wort, 16 Bits), **D** (Doppelwort, 32 Bits)

Beispiele:		
	`%QX1.0.2`	Ausgangsbit 2
	`%IW1.0.1`	Eingangsbit 1
	`%MB7`	Merkerbyte 7
	`%MW1`	Merkerwort 1
	`%MD3`	Merkerdoppelwort 3
	`%MX1.2`	drittes Merkerbit im Merkerwort 1



5 Programmiersprachen

In der IEC 61131-3 sind folgende fünf Programmiersprachen definiert:

- AWL: Anweisungsliste (*Instruction List, IL*)
- ST: Strukturierter Text (*Structured Text*)
- AS: Ablaufsprache (*Sequential Function Chart, SFC*)
- FUP: Funktionsplan (*Function Block Diagram, FBD*)
- KOP: Kontaktplan (*Ladder Diagram, LD*)

Jede Sprache für sich hat spezielle Anwendungsfälle bzw. Ausprägungen, die zur Lösung von bestimmten Problemfällen besonders geeignet sind.

5.1 Anweisungsliste (AWL)

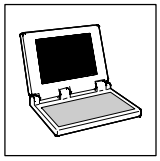
(Instruction List, IL)

Die Anweisungsliste ist mit der Sprache Assembler vergleichbar und besteht aus einer Folge von Anweisungen.

- Jede Anweisung beginnt in einer neuen Zeile, beinhaltet einen Operator und, je nach Art der Operation, einen oder mehrere durch Kommata abgetrennte Operanden.
- Vor einer Anweisung kann sich eine Identifikator-Marke befinden, gefolgt von einem Doppelpunkt (:).
- Ein Kommentar kann zusätzlich eingetragen werden.
- Leere Zeilen können zwischen Anweisungen eingefügt werden.

Beispiel:

```
LD    17
ST    lint            (* Kommentar *)
GE    5
JMPC  next
LD    idword
EQ    istruct.sdword
STN   test
next:
```



5.2 Strukturierter Text (ST)

(Structured Text)

Der Strukturierte Text besteht aus einer Reihe von Anweisungen, die wie in Hochsprachen bedingt (**IF..THEN..ELSE**) oder in Schleifen (**WHILE..DO**) ausgeführt werden können.

Neben der leistungsfähigen Schleifenprogrammierung und der Möglichkeit konditionierter Befehle können auch mathematische Funktionen hervorragend abgebildet werden, zudem ist Strukturierter Text eine sehr gut lesbare und verständliche Programmiersprache.

Beispiel:

```
IF value < 7 THEN
  WHILE value < 8 DO
        value := value + 1;
  END_WHILE;
END_IF;
```

Anweisungen (Überblick)

Anweisungsart	Beispiel

Zuweisung durch Zuweisungsoperator	`A:=B;`
	`CV:=CV + 1;`
	`C:=SIN(X);`

Aufruf eines Funktionsblocks, Benutzung der FB-Ausgabe	`CMD_TMR(IN:=%IX5, PT:=300);` `A:=CMD_TMR.Q`
RETURN	**RETURN**;

IF-Bedingung	`D:=B*B;`
	IF D<0.0 **THEN**
	  C:=A;
	ELSIF D=0.0 **THEN**
	  C:=B;
	ELSE
	  C:=D;
	END_IF;

CASE-Auswahl	**CASE** INT1 **OF**
	  1: BOOL1:=TRUE;
	  2: BOOL2:=TRUE;
	ELSE
	  BOOL1:=FALSE;
	  BOOL2:=FALSE;
	END_CASE;

FOR-Schleife	`J:=101;`
	FOR I:=1 **TO** 100 BY 2 **DO**
	  IF ARR[I]=70 THEN
	  J:=I;
	  EXIT;
	  END_IF;
	END_FOR;

WHILE-Schleife	`J:=1;`
	WHILE J<=100 AND ARR[J]<>70 **DO**
	  J:=J+2;
	END_WHILE;

REPEAT-Schleife	`J:=-1;`
	REPEAT
	  J:=J+2;
	UNTIL J= 101 OR ARR[J]=70
	END_REPEAT;

EXIT	**EXIT**;
Leere Anweisung	**;**

5.3 Ablaufsprache (AS)

(Sequential Function Chart, SFC)

Die Ablaufsprache ist eine graphisch orientierte Sprache, die es ermöglicht, die zeitliche Abfolge verschiedener Aktionen innerhalb eines Programms zu beschreiben.

Ein in Ablaufsprache geschriebener Baustein besteht aus einer Folge von Schritten, die über gerichtete Verbindungen (Transitionen) miteinander verbunden sind.

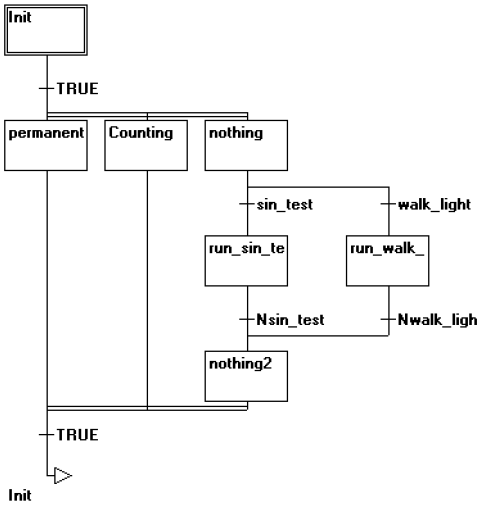


Abb. 3 Beispiel für ein Netzwerk in der Ablaufsprache

Die grafische Darstellung von Transitionen und Schritten erinnert an ein Flußdiagramm, ist sehr gut lesbar und eignet sich hervorragend für die Programmierung übergeordneter Zustandsabläufe.

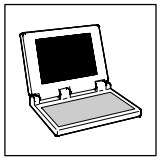
Schritte

Ein Schritt besteht aus einem Flag und einer oder mehreren zugewiesenen Aktionen oder boolschen Variablen.

- Die Aktionen von Schritten liegen getrennt von den Schritten vor und können innerhalb ihres Bausteins mehrfach verwendet werden.
- Über sogenannte Bestimmungszeichen (Qualifier) werden die Aktionen und boolschen Variablen aktiviert und deaktiviert, zum Teil mit zeitlichen Verzögerungen.
- Da eine Aktion immer noch aktiv sein kann, auch wenn bereits der nächste Schritt abgearbeitet wird, z. B. durch das Bestimmungszeichen S (Set), kann man Nebenläufigkeiten erreichen.

Aktionen

Eine Aktion kann eine Folge von Instruktionen in AWL oder in ST, eine Menge von Netzwerken in FUP oder in KOP oder wieder eine Ablaufstruktur (AS) enthalten.



Transitionen

Zwischen den Schritten liegen sogenannte Transitionen. Der einer Transition folgende Schritt wird aktiv, wenn die Transitionsbedingung **TRUE** ist.

Folgende Transitionsbedingungen sind möglich:

- eine Boolsche Variable
- eine Boolsche Adresse
- eine Boolsche Konstante (**TRUE**)
- eine Folge von Instruktionen mit einem Boolschen Ergebnis in ST-Syntax (`(i<=100) AND b`)
- eine Folge von Instruktionen ausprogrammiert in einer beliebigen Sprache

Alternativverzweigungen

Zwei oder mehr Zweige in AS können als Alternativverzweigungen definiert werden.

- Jeder Alternativzweig muß mit einer Transition beginnen und enden.
- Alternativverzweigungen können Parallelverzweigungen und weitere Alternativverzweigungen beinhalten.
- Eine Alternativverzweigung beginnt an einer horizontalen Linie (Alternativanfang) und endet an einer horizontalen Linie (Alternativende) oder mit einem Sprung.
- Wenn der Schritt, der der Alternativanfangslinie vorangeht, aktiv ist, dann wird die erste Transition jeder Alternativverzweigung von links nach rechts ausgewertet. Die erste Transition von links, deren Transitionsbedingung den Wert **TRUE** hat, wird geöffnet und die nachfolgenden Schritte werden aktiviert.

Parallelverzweigungen

Zwei oder mehr Verzweigungen in AS können als Parallelverzweigungen definiert werden.

- Jeder Parallelzweig muß mit einem Schritt beginnen und enden.
- Parallelverzweigungen können Alternativverzweigungen oder weitere Parallelverzweigungen beinhalten.
- Eine Parallelverzweigung beginnt bei einer doppelten Linie (Parallelanfang) und endet bei einer doppelten Linie (Parallelende) oder mit einem Sprung.
- Wenn der der Parallelanfangs-Linie vorangehende Schritt aktiv ist, und die Transitionsbedingung nach diesem Schritt den Wert **TRUE** hat, dann werden die ersten Schritte aller Parallelverzweigungen aktiv. Diese Zweige werden nun alle parallel zueinander abgearbeitet.
- Der Schritt nach der Parallelende-Linie wird aktiv, wenn alle vorangehenden Schritte aktiv sind, und die Transitionsbedingung vor diesem Schritt den Wert **TRUE** liefert.

Sprünge

Ein Sprung ist eine Verbindung zu dem Schritt, dessen Name unter dem Sprungsymbol angegeben ist.

Sprünge werden benötigt, weil es nicht erlaubt ist, nach oben führende oder sich überkreuzende Verbindungen zu schaffen.

5.4 Funktionsplan (FUP)

(Function Block Diagram, FBD)

Der Funktionsplan ist eine graphisch orientierte Programmiersprache.

Er arbeitet mit einer Liste von Netzwerken, wobei jedes Netzwerk eine Struktur enthält, die jeweils einen logischen bzw. arithmetischen Ausdruck, den Aufruf eines Funktionsblocks, einen Sprung oder eine Return-Anweisung darstellt. Ausgänge von Funktionsblöcken werden mit Eingängen folgender Funktionsblöcke verbunden, Sprünge und Rücksprunge erleichtern die Programmierung.

Basierend auf definierten Funktionsblöcken lassen sich mit dem Funktionsplan beliebige Programmabläufe mit Hilfe von Verbindungselementen vollgrafisch realisieren. Zudem trägt die schematische Darstellung des Datenflusses dazu bei, Programmabläufe transparent zu halten.

Wie auch bei der **9300 Servo PLC** und der **Drive PLC** werden häufig Hardwarekomponenten mit den zugehörigen Funktionsblöcken angeboten, so daß sowohl auf Hard-, als auch auf Software-Ebene korrespondierende Module existieren.

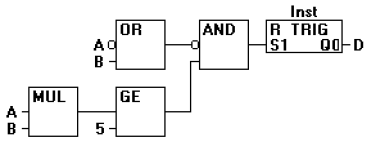


Abb. 4 Beispiel für ein Netzwerk im Funktionsplan

5.5 Kontaktplan (KOP)

(Ladder Diagramm, LD)

Der Kontaktplan ist eine graphisch orientierte Programmiersprache, die dem Prinzip einer elektrischen Schaltung angenähert ist.

Einerseits eignet sich der Kontaktplan dazu, logische Schaltwerke zu konstruieren, andererseits kann man aber auch Netzwerke wie im FUP erstellen. Daher kann der Kontaktplan sehr gut dazu benutzt werden, um den Aufruf von anderen Bausteinen zu steuern.

Der Kontaktplan besteht aus einer Folge von Netzwerken.

Ein Netzwerk wird auf der linken und rechten Seite von einer vertikalen Stromleitung begrenzt. Dazwischen befindet sich ein Schaltplan aus Kontakten, Spulen und Verbindungslinien, die von links nach rechts den Zustand "AN" oder "AUS" (**TRUE** oder **FALSE**) weitergeben:

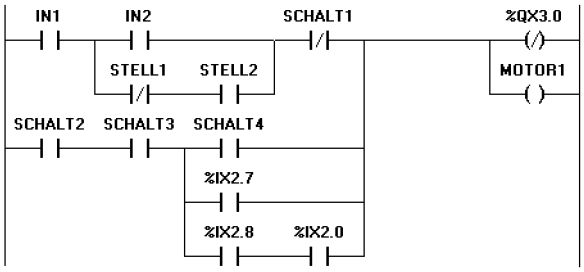
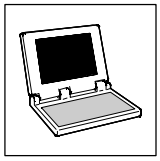


Abb. 5 Beispiel für ein Netzwerk im Kontaktplan



Kontakte

Hat die zu einem Kontakt gehörige boolsche Variable den Wert **TRUE**, dann wird der Zustand "AN" über die Verbindungslinie von links nach rechts weitergegeben, sonst erhält die rechte Verbindung den Wert "AUS".

- Kontakte können parallel geschaltet sein, dann muß einer der Parallelzweige den Wert "AN" übergeben, damit die Parallelverzweigung den Wert "AN" übergibt.
- Kontakte können in Reihe geschaltet sein, dann müssen alle Kontakte den Zustand "AN" übergeben, damit der letzte Kontakt den Zustand "AN" weitergibt.
- Ein Kontakt kann auch negiert sein, erkennbar am Schrägstrich im Kontaktsymbol. Er gibt dann den Eingangszustand weiter, wenn sein Zustand "AUS" (**FALSE**) ist.

Spulen

Auf der rechten Seite eines Netzwerks im KOP befindet sich eine beliebige Anzahl sogenannter Spulen, dargestellt durch Klammern.

- Eine Spule gibt den Wert der Verbindungen von links nach rechts weiter, und kopiert ihn in eine zugehörige boolsche Variable.
- An der Eingangslinie kann der Wert "AN" oder "AUS" anliegen (entsprechend den boolschen Werten **TRUE** bzw. **FALSE**).
- Spulen können nur parallel geschaltet werden.
- Eine Spule kann auch negiert sein, erkennbar am Schrägstrich im Spulensymbol. Sie kopiert dann den negierten Wert in die zugehörige boolsche Variable.

Set/Reset-Spulen

Eine Spule kann auch als Set oder Reset-Spule definiert sein.

Eine Set-Spule (erkennbar am "S" im Spulensymbol) kann den Zustand "AN" annehmen, daraufhin aber nicht mehr den Zustand "AUS".

- Wenn die boolsche Variable der Set-Spule einmal auf **TRUE** gesetzt wurde, dann kann sie daraufhin nicht mehr auf **FALSE** gesetzt werden.

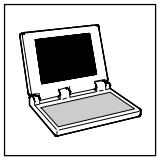
Eine Reset-Spule (erkennbar am ‚R' im Spulensymbol) kann den Zustand "AUS" annehmen, daraufhin aber nicht mehr den Zustand "AN".

- Wenn die boolsche Variable der Reset-Spule einmal auf **FALSE** gesetzt wurde, dann kann sie daraufhin nicht mehr auf **TRUE** gesetzt werden.

Funktionsblöcke im Kontaktplan

Neben Kontakten und Spulen können Sie im Kontaktplan auch Funktionsblöcke und Programme eingeben.

Funktionsblöcke bzw. Programme müssen im Netzwerk einen Eingang und einen Ausgang mit boolschen Werten haben und können an denselben Stellen verwendet werden wie Kontakte, d. h. auf der linken Seite des KOP-Netzwerks.



6 Anhang

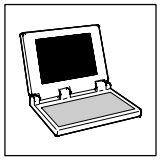
6.1 IEC-Schlüsselwörter

Schlüsselwörter sind eindeutige Kombinationen von Zeichen, die als individuelle Syntax-Elemente angewendet werden.

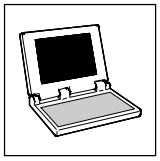
- Schlüsselwörter dürfen nicht als Bezeichner verwendet werden.
- Zu den Schlüsselwörtern im **Drive PLC Developer Studio** gehören auch die Namen der Lenze-Funktionsblöcke, die immer mit "L_" beginnen (**`L_ABS`**, **`L_ADD`**, ...).

Reservierte Schlüsselwörter für Programmiersprachen der IEC 61131-3:

ABS	ACOS	ACTION	ADD	AND	ANDN
ANY	ANY_BIT	ANY_DATE	ANY_INT	ANY_NUM	ANY_REAL
ARRAY	ASIN	AT	ATAN		
BOOL	BY	BYTE			
CAL	CALC	CALCN	CASE	CD	CDT
CLK	CONCAT	CONFIGURATION	CONSTANT	COS	CTD
CTU	CTUD	CU	CV		
DATE	DATE_AND_TIME	DELETE	DINT	DIV	DO
DS	DT	DWORD			
ELSE	ESIF	END_ACTION	END_CASE	END_CONFIGURATION	END_FOR
END_FUNCTION	END_FUNCTION_BLOCK		END_IF	END_PROGRAM	END_REPEAT
END_RESOURCE	END_STEP	END_STRUCT	END_TRANSITION	END_TYPE	END_VAR
END_WHILE	EN	ENO	EQ	ET	EXIT
EXP	EXPT				
FALSE	F_EDGE	F_TRIG	FIND	FOR	FROM
FUNCTION	FUNCTION_BLOCK				
GE	GT				
IF	IN	INITIAL_STEP	INSERT	INT	INTERVAL
JMP	JMPC	JMPCN			
L	LD	LDN	LE	LEFT	LEN
LIMIT	LINT	LN	LOG	LREAL	LT
LWORD					
MAX	MID	MIN	MOD	MOVE	MUL
MUX					
N	NE	NEG	NOT		
OF	ON	OR	ORN		



P	PRIORITY	PROGRAM	PT	PV	
Q	Q1	QU	QD		
R	R1	R_TRIG	READ_ONLY	READ_WRITE	REAL
RELEASE	REPEAT	REPLACE	RESOURCE	RET	RETAIN
RETC	RETCN	RETURN	RIGHT	ROL	ROR
RS	RTC	R_EDGE			
S	S1	SD	SEL	SEMA	SHL
SHR	SIN	SINGLE	SINT	SL	SQRT
SR	ST	STEP	STN	STRING	STRUCT
SUB					
TAN	TASK	THEN	TIME	TIME_OF_DAY	TO
TOD	TOF	TON	TP	TRANS	TRUE
TYPE					
UDINT	UINT	ULINT	UNTIL	USINT	
VAR	VAR_ACCESS	VAR_EXTERNAL	VAR_GLOBAL	VAR_INPUT	VAR_IN_OUT
VAR_OUTPUT					
WHILE	WITH	WORD			
XOR	XORN				



6.2 Konventionen für Lenze Variablenbezeichner

Dieses Kapitel stellt Ihnen die Konventionen vor, die für die Variablenbezeichner der Lenze Systembausteine, Funktionsblöcke sowie Funktionen verwendet werden, um eine einheitliche und durchgängige Benennung zu gewährleisten und dadurch die Lesbarkeit von SPS-Programmen zu fördern.

Tip!

Die von Lenze verwendeten Konventionen basieren auf der sogenannten “Ungarischen Notation”, wodurch anhand des Bezeichners sofort auf die wichtigsten Eigenschaften (z. B. den Datentyp) der entsprechenden Variable geschlossen werden kann.

Ein Bezeichner setzt sich zusammen aus

- einer **Systembaustein-Bezeichnung** (nur bei Bezeichnern von Systembaustein-Variablen)
- einer **Variablentypangabe** (optional)
- einer **Datentypangabe**
- einem **Identifikator** (dem “eigentlichen” Namen der Variablen)
- einer **Signaltypangabe** (optional)

6.2.1 Systembaustein-Bezeichnung

(nur bei Bezeichnern von Systembaustein-Variablen)

Der Zugriff auf die Ein-/Ausgänge eines Systembausteins erfolgt direkt über entsprechende I/O-Variablen.

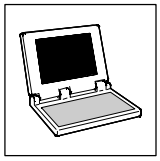
Um kenntlich zu machen, welchem Systembaustein eine solche I/O-Variable zugeordnet ist, wird dem Bezeichner der Name des entsprechenden Systembausteins gefolgt von einem Unterstrich vorgesetzt.

Beispiele für Systembaustein-Bezeichnungen:

*AIN_*

*CAN1_*

*DIGIN_*



6.2.2 Variablentypangabe

Die Variablentypangabe kann optional verwendet werden, um im Bezeichner den Variablentyp mit anzugeben:

Variablentypangabe (optional)	Bedeutung
I_	VAR_INPUT
Q_	VAR_OUTPUT
IQ_	VAR_IN_OUT
R_	VAR RETAIN
C_	VAR CONSTANT
CR_	VAR CONSTANT RETAIN
g_	VAR_GLOBAL
gR_	VAR_GLOBAL RETAIN
gC_	VAR_GLOBAL CONSTANT
gCR_	VAR_GLOBAL CONSTANT RETAIN

6.2.3 Datentypangabe

Die Datentypangabe gibt Aufschluß darüber, von welchen Datentyp die entsprechende Variable ist:

Datentypangabe	Bedeutung
b	Bool
by	Byte
n	Integer
w	Word
dn	Double Integer
dw	Double Word
s	String
f	Real (Float)
sn	Short Integer
t	Time
un	Unsigned Integer
udn	Unsigned Double Integer
usn	Unsigned Short Integer

Handelt es sich bei der Variable um ein Array oder einen Zeiger (Pointer), so wird dies zusätzlich **vor** der eigentlichen Datentypangabe mit angegeben:

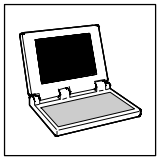
Datentypangabe (optional)	Bedeutung
a	Array (zusammengesetzter Typ), Feld
p	Zeiger (pointer)

Beispiele für Datentypangaben:

aby (Array vom Datentyp Byte)

dn (Double Integer)

pdn (Zeiger auf Double Integer)



6.2.4 Identifikator

Der Identifikator ist der "eigentliche" Name der Variable und sollte auf den Verwendungszweck bzw. die Funktion der Variable hinweisen.

- Der Identifikator beginnt immer mit einem Großbuchstaben.
- Setzt sich ein Identifikator aus mehreren "Worten" zusammen, so beginnt jedes "Wort" mit einem Großbuchstaben.
- Alle anderen Buchstaben werden klein geschrieben.

Beispiele für Identifikatoren:

JogValue

NumberOfValues

CurrentSelectedJogValue

6.2.5 Signaltypangabe

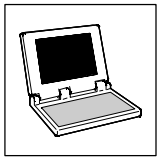
Den meisten Ein- und Ausgängen von Lenze-Funktionsblöcken/Systembausteinen kann ein bestimmter Signaltyp zugeordnet werden, wobei zwischen digitalen, analogen, Lage- sowie Drehzahlsignalen unterschieden wird.

Dem Bezeichner der entsprechenden Ein-/Ausgangsvariable wird eine Endung (angeführt mit einem Unterstrich) angefügt, die angibt, um welchen Signaltyp es sich handelt.

Signaltyp	Endung		Speicherplatz	Normierung (externe Größe ≡ interne Größe)	Bisherige Kennung
analog	_a	(analog)	16 Bit	100 % ≡ 16384	○
digital	_b	(binary)			□
Winkeldifferenz oder Drehzahl	_v	(velocity)	16 Bit	15000 rpm ≡ 16384	△
Winkel oder Lage	_p	(position)	32 Bit	1 Motorumdrehung ≡ 65535	▲

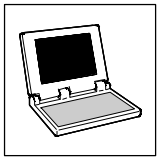
6.2.6 Beispiele für Variablenbezeichner

Variablenbezeichner	zugehöriger Systembaustein	Variablentyp	Datentyp	Signaltyp	Verwendungszweck/Funktion
g_anFixSetSpeedValue_a	-	VAR_GLOBAL	Array (Integer)	analog	Array für fixe Sollwerte
CAN2_nOutW1_a	CAN2_IO	-	Integer	analog	Ausgangswort 1 vom CAN2_OUT
AOUT1_nOut_a	AOUT1	-	Integer	analog	Auszugebendes Analogsignal
bQSP_b	-	-	Bool	binary (TRUE/FALSE)	Auslösen von Quickstop
byFunction	-	-	Byte	-	Funktionsauswahl
dnIn1_p	-	-	Double Integer	position	Winkeleingangssignal 1
nVp	-	-	Integer	-	Verstärkung



6.3 Glossar

Ablaufsprache	Ablaufsprache AS (*Sequential Function Chart - SFC*) ist eine Programmiersprache zur Beschreibung sequentieller und paralleler Steuerungsabläufe mit Zeit- und Ereignis-Steuerung.
Aktion	Boolsche Variable oder eine Reihe von Anweisungen, die über einen Aktionsblock angesteuert werden können, in AS.
Aktionsblock	Aktivierungsbeschreibung von Aktionen in AS.
Aktuelles Ergebnis	Zwischenergebnis in AWL von beliebigem Datentyp.
Anweisungsliste	Anweisungsliste (*Instruction List - IL*) ist eine weit verbreitete Assembler-ähnliche Programmiersprache für SPS-Systeme.
AS	Abk. für Ablaufsprache.
AWL	Abk. für Anweisungsliste.
Baustein	(Unter-) Programmeinheit, aus denen SPS-Programme zusammengesetzt werden. Bausteine können oft unabhängig voneinander in die SPS geladen werden. Vgl. POE.
CPU	Zentraleinheit (*Central Processing Unit*), z. B. einer SPS.
Deklaration	Bekanntgabe von Variablen und FB-Instanzen in einem Deklarationsblock unter Angabe eines Bezeichners, des Datentyps bzw. FB-Typs sowie ggf. Anfangswerte, Bereichsangabe und Feldeigenschaften.
Deklarationsblock	Zusammenfassung von Deklarationen einer Variablenart zu Beginn der POE.
Elementarer Datentyp	Ein durch die IEC 61131-3 vordefinierter Standard-Datentyp.
Erweiterung von Funktionen	Eine Funktion kann eine variable Anzahl von Eingängen besitzen.
FB	Abk. für Funktionsblock (*Function Block*), oftmals wird hierfür auch der Begriff “Funktionsbaustein” verwendet.
FB-Instanz	siehe Instanz
FB-Typ	Name eines Funktionsblocks mit Aufruf- und Rückgabeschnittstelle.
FBD	*Function Block Diagram*, siehe Funktionsplan
Funktion	Eine POE vom Typ `Function`
Funktionsbaustein	siehe Funktionsblock
Funktionsblock	Eine POE vom Typ `Function_Block`
Funktionsplan	Der Funktionsplan (*Function Block Diagram*) besteht aus einer Liste von Netzwerken, in denen sich beliebige Programmabläufe mit Hilfe von Verbindungselementen vollgrafisch realisieren lassen.
FUP	siehe Funktionsplan
IL	*Instruction List*, siehe Anweisungsliste
Indirekter FB-Aufruf	Aufruf einer FB-Instanz, dessen Name als `VAR_IN_OUT`-Parameter einer POE übergeben wurde.
Instanz	Strukturierter Datensatz eines FBs durch Deklaration eines Funktionsblocks unter Angabe des FB-Typs.
KOP	Abk. für Kontaktplan.
Konfiguration	Die Konfiguration (*Configuration*) definiert die Struktur der SPS, sie bildet die oberste Ebene im Softwaremodell der IEC 61131-3.
Kontaktplan	Kontaktplan (*Ladder Diagram - LD*) ist eine Programmiersprache zur Beschreibung von Netzwerken mit gleichzeitig arbeitenden boolschen, elektromechanischen Elementen wie Kontakten und Spulen.
LD	*Ladder Diagram*, siehe Kontaktplan
POE	Abk. für Programmorganisationseinheit (*Program Organization Unit - POU*)
POU	*Program Organization Unit*, siehe Programmorganisationseinheit
Programmorganisationseinheit	Ein Baustein der IEC 61131-3 des Typs Funktion, Funktionsblock oder Programm, aus dem Anwenderprogramme hierarchisch aufgebaut werden.
Ressource	Eine Ressource (*Resource*) stellt innerhalb einer Konfiguration eine Zentraleinheit (CPU) dar.
Schritt	Zustandsknoten in einem AS-Programm, in dem Anweisungen der zu einem Schritt zugehörigen Aktion angestoßen werden.
SFC	*Sequential Function Chart*, siehe Ablaufsprache
SPS	Speicherprogrammierbare Steuerung (*Programmable Controller*).



Einführung in die IEC 61131-3 Programmierung

Anhang

ST	Abk. für Strukturierter Text.
Standard-Funktionen	Menge der durch die IEC 61131-3 fest vordefinierten Funktionen zur Realisierung SPS-typischer Funktionalität.
Standard-Funktionsbausteine	siehe Standard-Funktionsblöcke
Standard-Funktionsblöcke	Menge der durch die IEC 61131-3 fest vordefinierten Funktionsblöcke (*Function Blocks*) zur Realisierung SPS-typischer Funktionalität.
Strukturierter text	Strukturierter Text (*Structured Text*) ist eine Programmiersprache zur Beschreibung von Algorithmen und Ausführungssteuerung mit den Mitteln einer modernen Hochsprache.
Task	Definition von Laufzeiteigenschaften eines Programms.
Transition	Übergang von einem AS-Schritt zum nächsten durch Auswertung der Transitionsbedingung.
Typdefinition	Definition eines benutzerspezifischen Datentyps auf der Basis bereits vorhandener Datentypen.
Variable	Bezeichnung eines Datenspeichers, der Werte annehmen kann, die durch den Datentyp sowie Angaben bei der Variablen-Deklaration festgelegt werden.
Zyklus	Ein Durchlauf des (periodisch aufgerufenen) Anwenderprogramms.
Zykluszeit	Die Zeit, die ein Anwenderprogramm für einen Zyklus benötigt.