A Standard-Funktionen

Dieser Anhang stellt die in Kap. 2 exemplarisch beschriebenen SPS-Standard-Funktionen vollständig zusammen. Für jede Standard-Funktion der IEC 61131-3 werden hier ihre:

- Grafische Deklaration,
- (Semantische) Beschreibung,
- Spezifizierung einiger Funktionen in Strukturiertem Text (ST)

angegeben.

Standard-Funktionen (Std.-FUN) besitzen Eingangsvariablen (Formalparameter) sowie einen Funktionswert (der Rückgabewert der Funktion). Einige Eingangsvariablen sind namenlos.

Um ihr Funktionsverhalten zu beschreiben, gelten folgende Vereinbarungen:

- eine einzelne Eingangsvariable ohne Namen wird mit „IN“ bezeichnet,
- mehrere Eingangsvariablen ohne Namen werden mit „IN1, IN2, ... INn“ durchnumeriert,
- der Funktionswert wird mit „F“ bezeichnet.

Zur Beschreibung werden u.a. Allgemeine Datentypen (wie ANY oder ANY_BIT) verwendet, deren Bedeutung in Abschn. 3.4.3 erläutert wird und die in Tab. 3.9 zusammengefasst werden. Die Bezeichnung ANY steht dabei als Abkürzung für einen beliebigen der Datentypen: ANY_BIT, ANY_NUM, ANY_STRING, ANY_DATE oder TIME.

Viele Standard-Funktionen besitzen einen textuellen Namen sowie eine alternative Darstellung mit einem Symbol (z.B. ADD und „+“). In den Abbildungen und Tabellen werden jeweils beide Varianten angegeben.

A.1 Funktionen zur Typwandlung

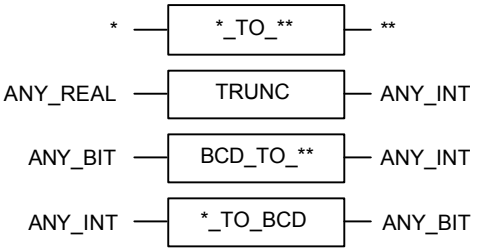


Abb. A.1. Grafische Deklarationen der Funktionen zur Typumwandlung

Diese Standard-Funktionen wandeln die Eingangsvariablen in den Datentyp ihres Funktionswerts (Typumwandlung, Konvertierung).

Name	Funktion / Beschreibung
*_TO_**	Bei der Umwandlung von REAL-Werten in INT-Werte wird zur nächsten ganzzahligen Zahl auf- bzw. abgerundet, halbe Anteile wie 0,5 oder 0,05 werden dabei aufgerundet.
TRUNC	Mit dieser Funktion werden die Stellen eines REAL-Wertes hinter dem Komma abgeschnitten, um einen ganzzahligen Wert zu bilden.
BCD	Die Ein- bzw. Ausgangswerte vom Typ ANY_BIT stellen BCD-kodierte Bitfolgen für die Datentypen BYTE, WORD, DWORD und LWORD dar. Die BCD-Kodierung wird durch die IEC 61131-3 nicht festgelegt, sie ist implementierungsabhängig.

Tab. A.1. Beschreibung der Funktionen zur Typwandlung

A.2 Numerische Funktionen

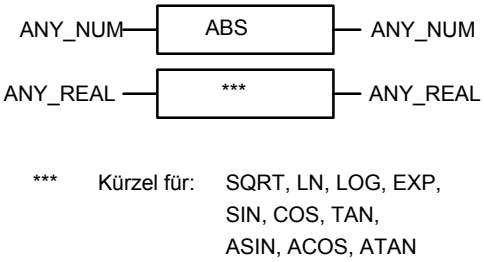


Abb. A.2. Grafische Deklarationen der Numerischen Funktionen

Name	Funktion	Beschreibung
ABS	Absolutwert	F := | IN |
SQRT	Quadratwurzel	F := $\sqrt{IN}$
LN	Natürlicher Logarithmus	F := $\log_e$ (IN)
LOG	Logarithmus Basis 10	F := $\log_{10}$ (IN)
EXP	Exponent Basis e	F := e^{IN}
SIN	Sinus, IN in Bogenmaß	F := SIN (IN)
COS	Cosinus, IN in Bogenmaß	F := COS (IN)
TAN	Tangens, IN in Bogenmaß	F := TAN (IN)
ASIN	Arcsin, Hauptwert	F := ARCSIN (IN)
ACOS	Arccos, Hauptwert	F := ARCCOS (IN)
ATAN	Arctan, Hauptwert	F := ARCTAN (IN)

Tab. A.2. Beschreibung der Numerischen Funktionen

A.3 Arithmetische Funktionen

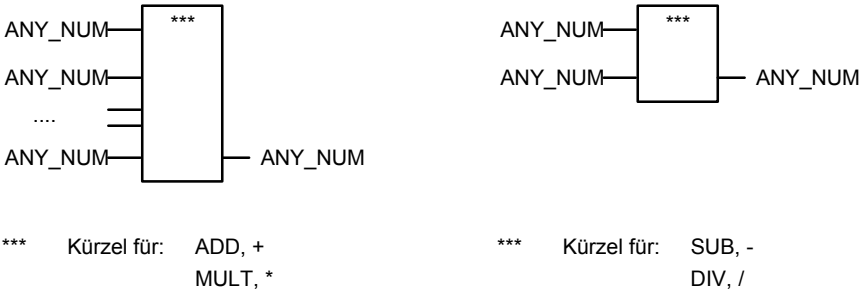


Abb. A.3. Grafische Deklarationen der Arithmetischen Funktionen ADD, MUL, SUB und DIV

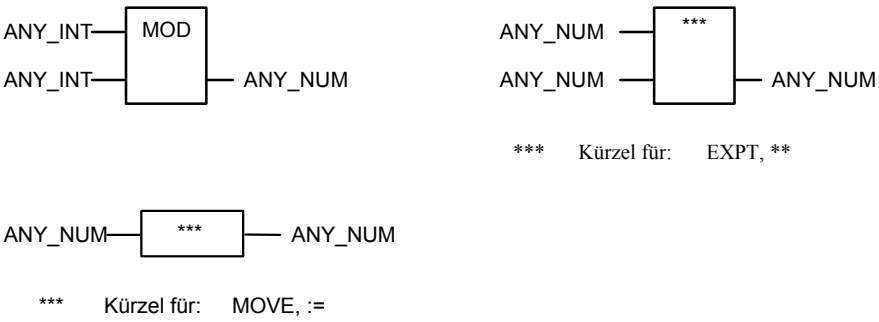


Abb. A.4. Grafische Deklarationen der Arithmetischen Std-FUN MOD, EXPT und MOVE

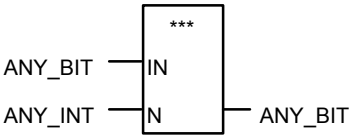
Name	Symbol	Funktion	Beschreibung
ADD	+	Addition	F := IN1 + IN2 + ... + INn
MUL	*	Multiplikation	F := IN1 * IN2 * ... * INn
SUB	-	Subtraktion	F := IN1 - IN2
DIV	/	Division	F := IN1 / IN2
MOD		Restbildung	F := IN1 - (IN1/ IN2)*IN2
EXPT	**	Exponentiation	F := $IN1^{IN2}$
MOVE	:=	Zuweisung	F := IN

Tab. A.3. Beschreibung der arithmetischen Funktionen. ADD und SUB verwenden den Typ ANYMAGNITUDE. Bei EXPT hat IN1 den Typ ANY_REAL und IN2 den Typ ANY_NUM. MOVE verwendet ANY für Ein- und Ausgang.

Bei der Division von ganzen Zahlen muss das Ergebnis wieder eine ganze Zahl sein, ggf. wird das Ergebnis in Richtung Null abgeschnitten.

Falls der Eingangsparameter IN2 Null ist, wird zur Laufzeit ein Fehler mit Fehlerursache „Division durch Null“ gemeldet, vgl. Anh. E.

A.4 Bitschiebe-Funktionen



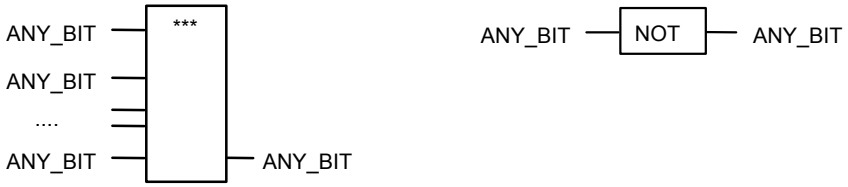
*** Kürzel für: SHL, SHR, ROL, ROR

Abb. A.5. Grafische Deklarationen der Bitschiebe-Funktionen SHL, SHR, ROR und ROL.

Name	Funktion	Beschreibung
SHL	Schieben nach links	IN um N Bits nach links schieben, von rechts mit Nullen füllen
SHR	Schieben nach rechts	IN um N Bits nach rechts schieben, von links mit Nullen füllen
ROR	Rotieren nach rechts	IN um N Bits ringförmig nach rechts schieben
ROL	Rotieren nach links	IN um N Bits ringförmig nach links schieben

Tab. A.4. Beschreibung der Bitschiebe-Funktionen. Der Eingang N muss dabei Werte größer gleich Null annehmen.

A.5 Bitweise Boolesche Funktionen



*** Kürzel für: AND, &, OR, >=1, XOR, =2k+1

Abb. A.6. Grafische Deklarationen der Bitweise Booleschen Funktionen AND, OR, XOR und NOT

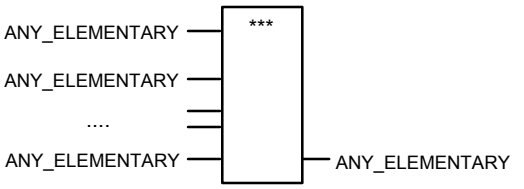
Name	Symbol	Funktion	Beschreibung
AND	&	Bitweise UND	F := IN1 & IN2 & ... & INn
OR	>=1	Bitweise ODER	F := IN1 v IN2 v ... v INn
XOR	=2k+1	Bitweise EXODER	F := IN1 XOR IN2 XOR ... XOR INn
NOT		Negation	F := ¬ IN

Tab. A.5. Beschreibung der Bitweise Booleschen Funktionen

Die Verknüpfung zwischen den Eingangsparametern erfolgt bitweise, d.h. jede Bitstelle eines Eingangs wird mit der entsprechenden Bitstelle des anderen Eingangs zur gleichen Bitstelle des Funktionswerts verknüpft.

Die grafische Darstellung einer Invertierung kann auch durch einen Kreis „o“ am booleschen Ein- oder Ausgangs einer Funktion erfolgen.

A.6 Auswahl-Funktionen für Max., Min. und Grenzwert



*** Kürzel für: MIN, MAX

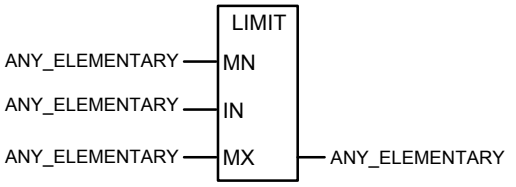


Abb. A.7. Grafische Deklarationen der Auswahlfunktionen MAX, MIN und LIMIT

Name	Funktion	Beschreibung
MAX	Maximum-Bildung	F := MAX (IN1, IN2, ... , INn)
MIN	Minimum-Bildung	F := MIN (IN1, IN2, ... , INn)
LIMIT	Begrenzung	F := MIN (MAX (IN, MN), MX)

Tab. A.6. Beschreibung der Auswahlfunktionen MAX, MIN und LIMIT

Diese drei Standard-Funktionen werden in Bsp. A.1 und Bsp. A.2 durch Deklaration in ST spezifiziert.

```
FUNCTION      MAX : ANY_ELEMENTARY  (* Maximum-Bildung *)
  VAR_INPUT   IN1, IN2, ... INn   : ANY_ELEMENTARY;        END_VAR
  VAR         Elem                : ANY_ELEMENTARY;        END_VAR
              (* ANY_ELEMENTARY steht für beliebige Datentypen*)

  IF IN1 > IN2 THEN       (* erster Vergleich *)
      Elem    := IN1;
  ELSE
      Elem    := IN2;
  END_IF;
  IF IN3 > Elem THEN      (* nächster Vergleich *)
      Elem    := IN3;
  END_IF;
  ...
  IF INn > Elem THEN      (* letzter Vergleich *)
      Elem    := INn;
  END_IF;
  MAX   :=  Elem;         (* Schreiben des Funktionswerts *)
END_FUNCTION
```

Bsp. A.1. Spezifizierung der Auswahlfunktion MAX in ST; für MIN sämtliche „>“ durch „<“ ersetzen.

Die Spezifizierung der Minimum-Funktion MIN kann man aus MAX dadurch erhalten, dass in Bsp. A.1 sämtliche Vorkommen von „>“ durch „<“ ersetzt werden.

```
FUNCTION      LIMIT : ANY_ELEMENTARY          (* Grenzwert-Bildung *)
  VAR_INPUT
    MN   :   ANY_ELEMENTARY;
    IN   :   ANY_ELEMENTARY;
    MX   :   ANY_ELEMENTARY;
  END_VAR
  MAX   :=  MIN ( MAX ( IN, MN), MX);       (* Aufrufe von MIN von MAX *)
END_FUNCTION
```

Bsp. A.2. Spezifizierung der Auswahlfunktion LIMIT in ST

A.7 Auswahl-Funktionen für Binäre Auswahl und Multiplexer

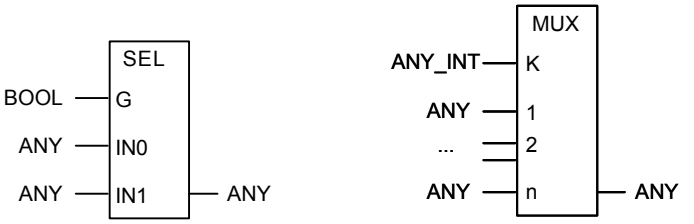


Abb. A.8. Grafische Deklarationen der Auswahlfunktionen SEL und MUX

Name	Funktion	Beschreibung
SEL	Binäre Auswahl	F := IN0, falls G = 0, IN1 sonst
MUX	Multiplexer	F := INi, falls K = i und 0<K<n-1

Tab. A.7. Beschreibung der Auswahlfunktionen SEL und MUX

Diese beiden Standard-Funktionen werden im folgenden durch Deklaration in ST spezifiziert:

```
FUNCTION      SEL : ANY          (* Binäre Auswahl *)
  VAR_INPUT
    G     :   BOOL;
    IN0   :   ANY;
    IN1   :   ANY;
  END_VAR
  IF G = 0 THEN
      SEL     :=  IN0;      (* Auswahl des oberen Eingangs *)
  ELSE
      SEL     :=  IN1;      (* Auswahl des unteren Eingangs *)
  END_IF;
END_FUNCTION
```

Bsp. A.3. Spezifizierung der Auswahlfunktion SEL in ST

```
FUNCTION      MUX : ANY          (* Multiplexer *)
  VAR_INPUT
    K     :   ANY_INT;
    IN0   :   ANY;
    IN1   :   ANY;
    ...
    INn   :   ANY;
  END_VAR
  IF (K < 0) OR (K>=n) THEN
      ... Fehlermeldung Nr....;        (* K negativ oder zu groß *)
  END_IF;
  CASE  K  OF
      0: MUX := IN0;        (* Auswahl des oberen Eingangs *)
      1: MUX := IN1;        (* Auswahl des zweiten Eingangs *)
      ...
      n: MUX := INn;        (* Auswahl des untersten Eingangs *)
  END_CASE;
END_FUNCTION
```

Bsp. A.4. Spezifizierung der Auswahlfunktion MUX in ST

A.8 Vergleichs-Funktionen

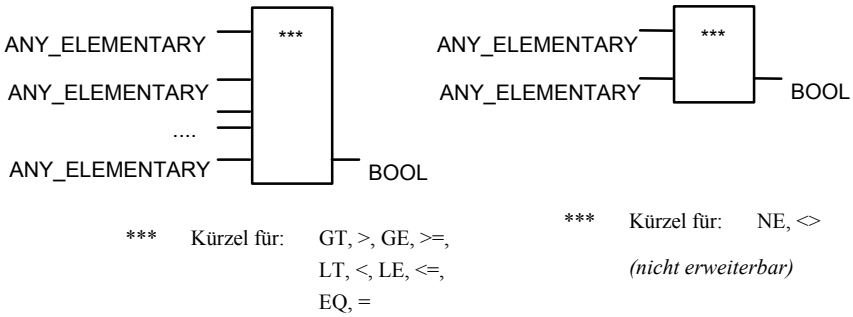


Abb. A.9. Grafische Deklarationen der Vergleichsfunktionen GT, GE, LT, LE, EQ, NE

Name	Funktion	Beschreibung
GT	Vergleich auf „> “	F := 1, falls INi > IN(i+1), 0 sonst
GE	Vergleich auf „> =“	F := 1, falls INi >= IN(i+1), 0 sonst
LT	Vergleich auf „< “	F := 1, falls INi < IN(i+1), 0 sonst
LE	Vergleich auf „< =“	F := 1, falls INi <= IN(i+1), 0 sonst
EQ	Vergleich auf „= “	F := 1, falls INi = IN(i+1), 0 sonst
NE	Vergleich auf „<>“	F := 1, falls INi <> IN(i+1), 0 sonst

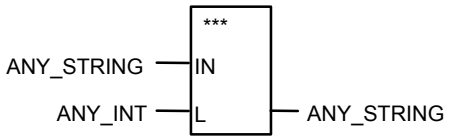
Tab. A.8. Beschreibung der Vergleichsfunktionen

Diese Standard-Funktionen werden in Bsp. A.5 stellvertretend durch Deklaration von GT in ST spezifiziert, von der die übrigen leicht abgeleitet werden können.

```
FUNCTION        GT : BOOL           (* Vergleich auf größer *)
  VAR_INPUT     IN1, IN2, ... INn   : ANY_ELEMENTARY;          END_VAR
  IF        (IN1 > IN2)
  AND       (IN2 > IN3)
  ...
  AND       (IN(n-1) > INn) THEN
        GT :=   TRUE;           (* Eingänge sind „sortiert“: monoton steigend *)
  ELSE
        GT :=   FALSE;          (* Bedingung nicht erfüllt *)
  END_IF;
END_FUNCTION
```

Bsp. A.5. Spezifizierung der Vergleichsfunktion GT in ST

A.9 Funktionen für Zeichenfolgen



*** Kürzel für: LEFT, RIGHT

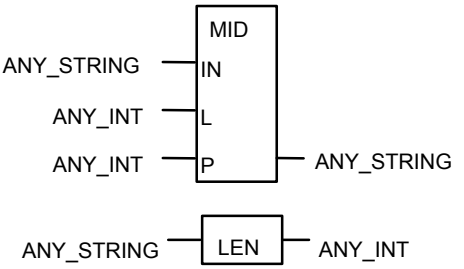


Abb. A.10. Grafische Deklarationen der Zeichenfolge-Funktionen LEFT, RIGHT, MID und LEN

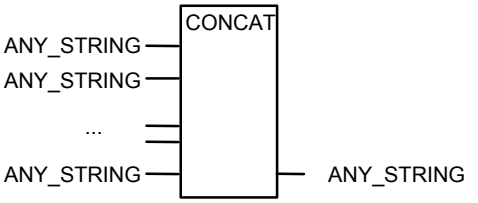


Abb. A.11. Grafische Deklarationen der Zeichenfolge-Funktion CONCAT

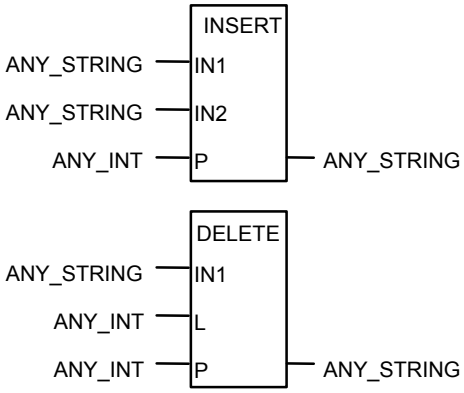


Abb. A.12. Grafische Deklarationen der Zeichenfolge-Funktionen INSERT und DELETE

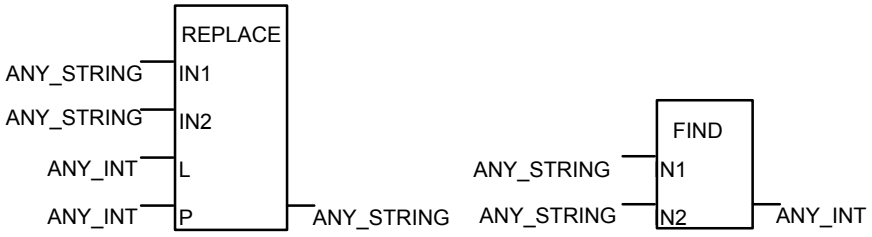


Abb. A.13. Grafische Deklarationen der Zeichenfolge-Funktionen REPLACE und FIND

Name	Funktion	Beschreibung
LEN	ermittelt die Länge einer Zeichenfolge	F := Anzahl der Zeichen in IN
LEFT	Anfangsabschnitt einer Zeichenfolge	F := Anfangsabschnitt mit L Zeichen
RIGHT	Endeabschnitt einer Zeichenfolge	F := Endeabschnitt mit L Zeichen
MID	Mittelabschnitt einer Zeichenfolge	F := Mittelabschnitt ab Position P mit L Zeichen
CONCAT	Aneinanderreihung von Zeichenfolgen	F := Gesamt-Zeichenfolge
INSERT	Einfügen einer Zeichenfolge in eine andere	F := Gesamt-Zeichenfolge mit neuem Teil ab Position P
DELETE	Löscht Abschnitt in einer Zeichenfolge	F := Rest-Zeichenfolge mit gelöschtem Teil (L Zeichen) ab Position P
REPLACE	Ersetzt einen Abschnitt in einer Zeichenfolge durch einen anderen	F := Gesamt-Zeichenfolge mit ersetztem Teil (L Zeichen) ab Position P
FIND	ermittelt die Position eines Abschnitts in einer Zeichenfolge	F := Index der gefundenen Position, sonst 0

Tab. A.9. Beschreibung der Zeichenfolge-Funktionen

Die Position eines Zeichens innerhalb einer Zeichenfolge vom Typ ANY_STRING wird beginnend mit „1“ angegeben.

Die Eingänge L und P müssen dabei Werte größer gleich Null annehmen.

A.10 Funktionen für Datentypen der Zeit

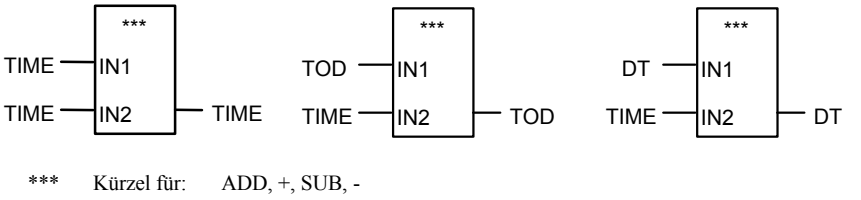


Abb. A.14. Grafische Deklarationen der gemeinsamen Zeit-Funktionen für Addition und Subtraktion

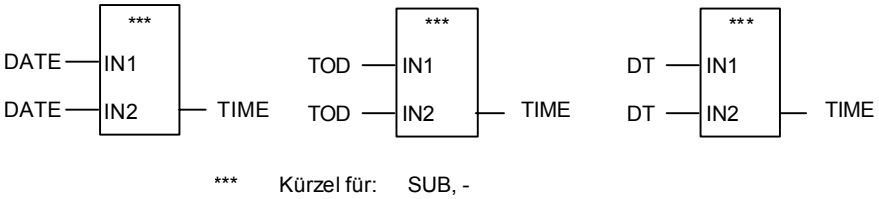


Abb. A.15. Grafische Deklarationen der zusätzlichen Zeit-Funktionen für Subtraktion

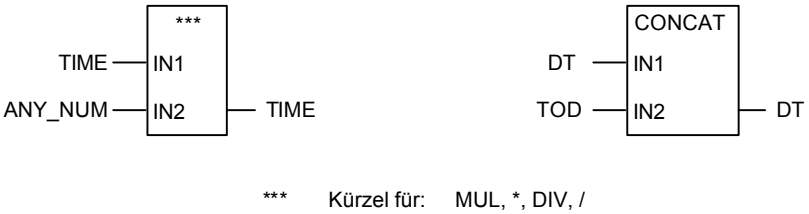


Abb. A.16. Grafische Deklarationen der Zeit-Funktionen für MUL, DIV und CONCAT

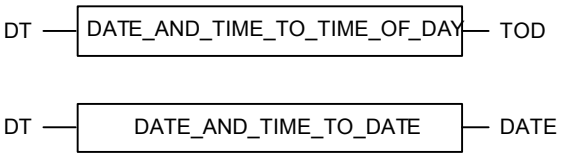


Abb. A.17. Grafische Deklarationen der Zeit-Funktionen zur Typumwandlung

Die Abkürzungen TOD und DT können gleichwertig anstelle der längeren Schlüsselworte TIME_OF_DAY bzw. DATE_AND_TIME verwendet werden.
Anstelle der Funktionsnamen ADD und SUB können auch die Varianten ADD_TIME, SUB_TIME und die Kombinationen ADD_TOD_TIME, ADD_DT_TIME, SUB_DATE_DATE, SUB_TOD_TIME, SUB_TOD_TOD, SUB_DT_TIME, MULTIME, DIVTIME, CONCAT_DATE_TOD verwendet werden.
Von dem Gebrauch der arithmetischen Bezeichnungen „+“, „-“, „/“ oder „*“ sowie ADD, SUB, DIV und MUL rät die Norm inzwischen explizit ab, um Verwechselungen und dadurch fehlerhafte Anwendung zu vermeiden.

A.11 Funktionen für Datentypen der Aufzählung

Die Standard-Funktionen SEL, MUX, EQ und NE können ebenfalls für Aufzählungs-Datentypen verwendet werden. Sie sind in diesen Fällen sinngemäß wie für ganze Zahlen anzuwenden (Werte von Aufzählungen entsprechen vom Programmiersystem „codierten“ Konstanten).

B Standard-Funktionsbausteine

Dieser Anhang stellt die in Kap. 2 exemplarisch beschriebenen Standard-Funktionsbausteine vollständig zusammen. Für jeden Standard-FB der IEC 61131-3 werden hier seine:

- Grafische Deklaration,
- (Semantische) Beschreibung,
- Spezifizierung einiger FBs in Strukturiertem Text (ST)

angegeben.

In diesem Buch werden die Ein- und Ausgänge der Std.-FBs mit den in der IEC 61131-3 festegelegten Namen versehen. Es soll jedoch darauf hingewiesen werden, dass aufgrund der Aufrufmöglichkeit der FBs als „AWL-Operatoren“ Konflikte entstehen können. Diese treten auf für die AWL-Operatoren (vgl. Abschn. 4.1.) LD, R und S, die zur Überprüfung der korrekten Programmsyntax von den FB-Eingängen LD, R und S unterschieden werden müssen. Diese drei Formaloperanden könnten beispielsweise auch (je nach Programmiersystem) mit LOAD, RESET und SET benannt werden, um Verwechselungen zu vermeiden.

B.1 Bistabile Elemente (Flip-Flops)

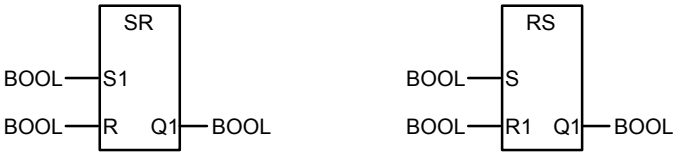


Abb. B.1. Grafische Deklarationen der Funktionsbausteine SR und RS

```
FUNCTION_BLOCK      SR          (* FlipFlop mit vorrangigem Setzen *)
  VAR_INPUT
    S1    :    BOOL;
    R     :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q1    :    BOOL;
  END_VAR
  Q1      :=   S1 OR ( NOT R AND Q1);
END_FUNCTION_BLOCK

FUNCTION_BLOCK      RS          (* FlipFlop mit vorrangigem Rücksetzen *)
  VAR_INPUT
    S     :    BOOL;
    R1    :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q1    :    BOOL;
  END_VAR
  Q1      :=   NOT R1 AND ( S OR Q1);
END_FUNCTION_BLOCK
```

Bsp. B.1. Spezifizierung der Funktionsbausteine SR und RS in ST

Diese beiden Flip-Flops realisieren Vorrangiges Setzen und Rücksetzen.

B.2 Flankenerkennung

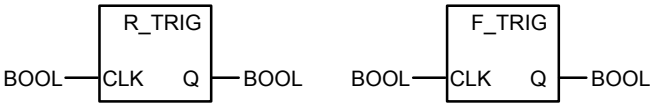


Abb. B.2. Grafische Deklarationen der Funktionsbausteine R_TRIG und F_TRIG

```
FUNCTION_BLOCK      R_TRIG     (* steigende Flanke *)
  VAR_INPUT
    CLK  :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q    :    BOOL;
  END_VAR
  VAR RETAIN                              (* beispielhafter Gebrauch der Pufferung *)
    MEM :     BOOL := 0;                  (* Flankenmerker initialisieren *)
  END_VAR
  Q       :=  CLK AND NOT MEM;            (* steigende Flanke erkennen *)
  MEM     :=  CLK;                        (* Flankenmerker rücksetzen *)
END_FUNCTION_BLOCK

FUNCTION_BLOCK      F_TRIG     (* fallende Flanke *)
  VAR_INPUT
    CLK  :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q    :    BOOL;
  END_VAR
  VAR RETAIN
    MEM :     BOOL := 1;                  (* Flankenmerker initialisieren *)
  END_VAR
  Q       :=  NOT CLK AND NOT MEM;        (* fallende Flanke erkennen *)
  MEM     :=  NOT CLK;                    (* Flankenmerker rücksetzen *)
END_FUNCTION_BLOCK
```

Bsp. B.2. Spezifizierung der Funktionsbausteine R_TRIG und F_TRIG in ST

Bei den FBs R_TRIG und F_TRIG ist zu beachten, dass sie bereits beim **ersten** Aufruf eine „Flanke“ erkennen, wenn der Eingang bei R_TRIG auf TRUE und bei F_TRIG auf FALSE liegt.

B.3 Zähler

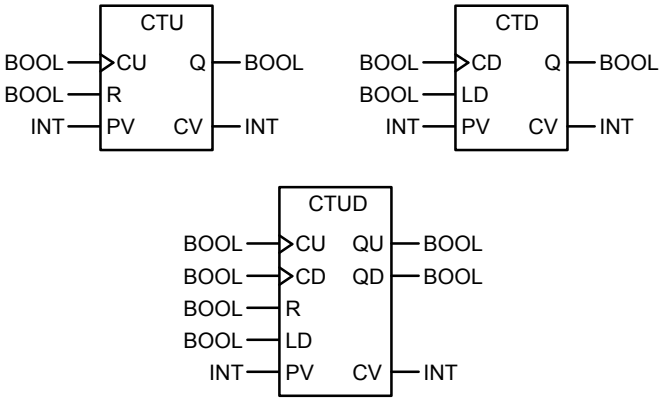


Abb. B.3. Grafische Deklarationen der Funktionsbausteine CTU, CTD und CTUD

```
FUNCTION_BLOCK        CTU      (* Vorwärts-Zähler *)
  VAR_INPUT
    CU   :   BOOL   R_EDGE;    (* CU mit steigender Flanke *)
    R    :   BOOL;
    PV   :   INT;
  END_VAR
  VAR_OUTPUT
    Q    :   BOOL;
    CV   :   INT;
  END_VAR
  IF  R  THEN                  (* Zähler rücksetzen *)
        CV     :=  0;
  ELSIF  CU AND ( CV < PV)  THEN
        CV     :=  CV + 1;     (* hochzählen *)
  ENDIF;
  Q    :=    (CV >= PV);       (* Grenzwert erreicht *)
END_FUNCTION_BLOCK
```

Bsp. B.3. Spezifizierung der Funktionsbausteine CTU und CTD in ST (wird fortgesetzt)

Die Zähler CTU und CTD gibt es auch in den Varianten für DINT, LINT, UDINT, ULINT und werden dann entsprechend mit CTU_DINT usw. bis CTD_ULINT bezeichnet.

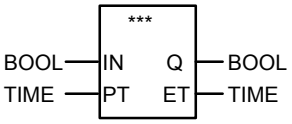
```
FUNCTION_BLOCK        CTD      (* Rückwärts-Zähler *)
  VAR_INPUT
    CD   :   BOOL    R_EDGE;   (* CD mit steigender Flanke *)
    LD   :   BOOL;
    PV   :   INT;
  END_VAR
  VAR_OUTPUT
    Q    :   BOOL;
    CV   :   INT;
  END_VAR
  IF  LD  THEN                  (* Zähler rücksetzen *)
        CV     :=  PV;
  ELSIF  CD AND ( CV > PV)  THEN
        CV     :=  CV - 1;      (* runterzählen *)
  ENDIF;
  Q    :=    (CV <= 0);        (* Null erreicht *)
END_FUNCTION_BLOCK
```

Bsp. B.3. (Fortsetzung)

```
FUNCTION_BLOCK        CTUD     (* Vorwärts/Rückwärts-Zähler *)
  VAR_INPUT
    CU   :   BOOL    R_EDGE;   (* CU mit steigender Flanke *)
    CD   :   BOOL    R_EDGE;   (* CD mit steigender Flanke *)
    R    :   BOOL;
    LD   :   BOOL;
    PV   :   INT;
  END_VAR
  VAR_OUTPUT
    QU   :   BOOL;
    QD   :   BOOL;
    CV   :   INT;
  END_VAR
  IF  R  THEN                   (* Zähler vorrangig rücksetzen *)
        CV   :=    0;
  ELSIF  LD  THEN
        CV   :=    PV;          (* auf Zählwert setzen *)
  ELSE
    IF NOT (CU AND CD) THEN
      IF  CU AND ( CV < PV)  THEN
        CV   :=    CV + 1;      (* hochzählen *)
      ELSIF  CD AND ( CV > PV)  THEN
        CV   :=    CV - 1;      (* runterzählen *)
      ENDIF;
    ENDIF;
  ENDIF
  QU   :=    (CV >= PV);        (* Grenzwert erreicht *)
  QD   :=    (CV <= 0);         (* Null erreicht *)
END_FUNCTION_BLOCK
```

Bsp. B.4. Spezifizierung des Funktionsbausteins CTUD in ST

B.4 Zeitgeber (Zeiten)



*** Kürzel für: TON, T---0, TOF, 0---T, TP

Abb. B.4. Grafische Deklarationen der Funktionsbausteine TON, TOF und TP

Die Spezifizierung der Zeitgeber-Elemente TP, TON und TOF wird im Folgenden anhand von Zeitdiagrammen vorgenommen.

Dieses Zeitverhalten setzt jeweils voraus, dass die Zykluszeit des periodischen SPS-Programms, in dem die Zeit verwendet wird, vernachlässigbar gering zur Zeitdauer PT ist, wenn die Zeit im Zyklus nur einmal aufgerufen wird.

In den Diagrammen wird das an den Ausgängen Q und ET zu beobachtende Verhalten der Zeit in Abhängigkeit vom Eingang IN dargestellt. Die Zeitachse läuft dabei von links nach rechts und ist mit „t“ beschriftet. Die booleschen Variablen IN und Q wechseln zwischen „0“ und „1“ und für ET wird der Verlauf des anwachsenden Zeitwerts dargestellt.

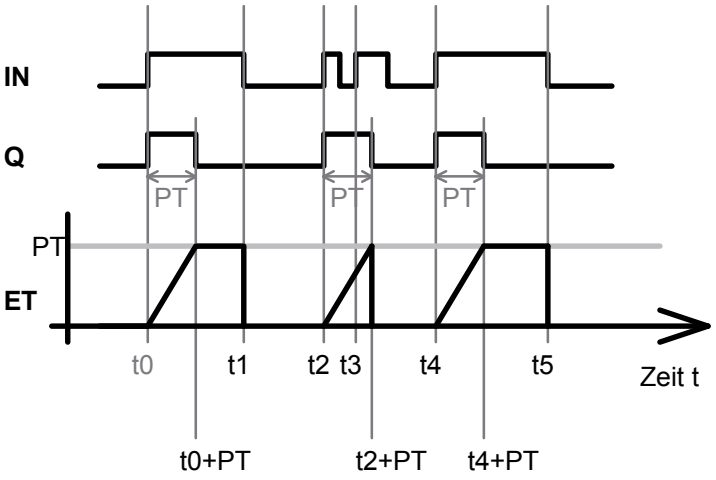


Abb. B.5. Zeitverhalten für Zeitimpuls TP abhängig vom Eingang IN

Der Std-FB „TP“ stellt einen Impulsgeber dar, der bei steigender Flanke am IN-Eingang am Ausgang Q einen konstant langen Impuls liefert. Am Ausgang ET kann jederzeit die bis dahin aufgelaufene Zeit abgefragt werden.

Wie man in Abb. B.5 erkennt, sind Zeiten vom Typ TP nicht „re-triggerbar“. Falls die Abstände zwischen den Eingangsimpulsen auf IN kürzer sind als die voreingestellte Zeitdauer, bleibt die Impulsdauer dennoch konstant; siehe Zeitraum [t2; t2+PT]. Die Zeitdauer beginnt also **nicht** mit jeder steigenden Flanke an IN.

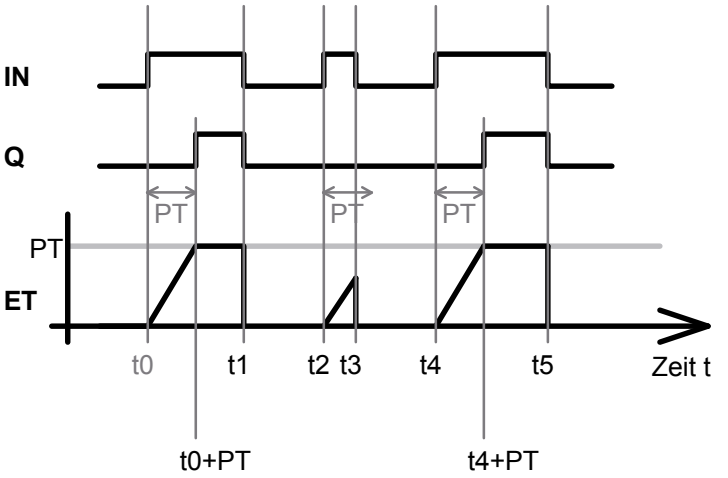


Abb. B.6. Zeitverhalten für Einschalt-Verzögerung TON abhängig vom Eingang IN

Die Einschaltverzögerung TON liefert an Q zeitverzögert den Eingangswert IN, wenn für IN eine steigende Flanke erkannt wird. Ist der Eingang IN jedoch nur für einen kurzen Impuls (kleiner PT) auf „1“, wird die Zeit für diese Flanke nicht mehr gestartet.

Am Ausgang ET kann die aufgelaufene Zeit abgelesen werden.

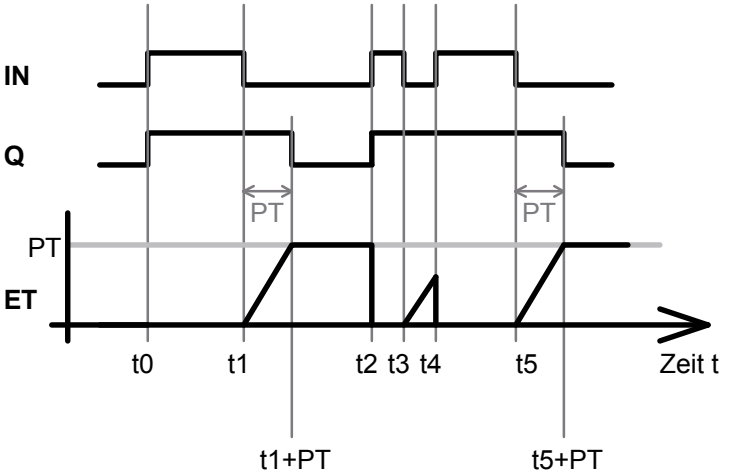


Abb. B.7. Zeitverhalten für Ausschalt-Verzögerung TOF abhängig vom Eingang IN

Die Ausschaltverzögerung ist die inverse Funktion zu TON, d.h. sie verzögert in derselben Weise eine fallende Flanke wie TON eine steigende.

Wie sich die Zeit verhält, wenn PT während der Zeitoperation verändert wird, ist implementierungsabhängig.

C AWL-Beispiele

Dieser Anhang beinhalte für jeden POE-Typ jeweils ein ausführliches Beispiel zur SPS-Programmierung nach IEC 61131-3, um die Erläuterungen in Kap. 2 und Kap. 4 zu unterstützen.

Diese Beispiele sind auf der diesem Buch beiliegenden CD zu finden.

C.1 Beispiel für FUNCTION

Die Funktion ByteExtr extrahiert das obere oder untere Byte eines Eingangsworts und gibt es als Funktionswert zurück.

```
FUNCTION ByteExtr : BYTE     (* Extraktion Byte aus Wort *)
                             (* Beginn Deklarationsteil *)
VAR_INPUT                    (* Eingangsvariablen *)
  Wort   :   WORD;           (* Wort besteht aus oberem + unterem Byte *)
  Oben   :   BOOL;           (* TRUE: oberes Byte, sonst unteres nehmen *)
END_VAR
                             (* Beginn Anweisungsteil *)
LD       Oben                (* Oberes oder unteres Byte extrahieren? *)
EQ       FALSE               (* unteres? *)
JMPCN    ObByte              (* Sprung bei Extraktion des oberen Bytes *)
LD       Wort                (* Wort laden *)
WORD_TO_BYTE                 (* Umwandlung fuer Typvertraeglichkeit *)
ST       ByteExtr            (* Zuweisung an den Funktionswert *)
RET                          (* nix zu tun *)
ObByte:                      (* Sprungmarke *)
LD       Wort                (* Wort laden *)
SHR      8                   (* oberes Byte 8 Bits nach rechts schieben *)
WORD_TO_BYTE                 (* Umwandlung fuer Typvertraeglichkeit *)
ST       ByteExtr
RET                          (* Ruecksprung mit Funktionswert in AE *)
                             (* FUN-Ende *)
END_FUNCTION
```

Bsp. C.1. Beispiel für die Deklaration einer Funktion in AWL

Die Funktion ByteExtr in Bsp. C.1 besitzt den Eingangsparameter „Wort“ vom Typ WORD und den booleschen Eingang Oben. Der Rückgabewert der Funktion ist vom Typ BYTE. Diese Funktion benötigt keine lokalen Variablen, weshalb der Deklarationsteil VAR ... VAR_END fehlt.

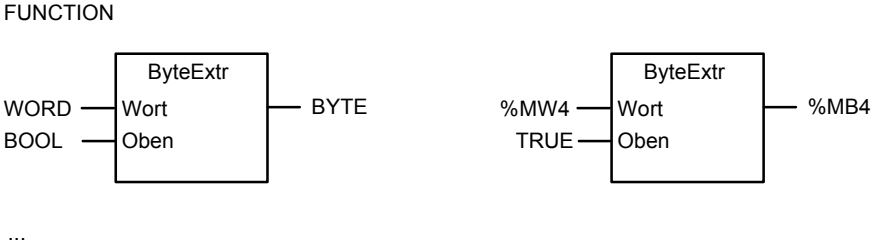
Der Rückgabewert steht im aktuellen Ergebnis (AE), wenn die Funktion mit RET zum Aufrufer zurückkehrt. Das AE ist an dieser Stelle (Sprungmarke Ende:) vom Datentyp WORD, denn Wort wurde zuvor geladen, der Funktionswert nach der Typkonvertierung vom Typ BYTE.

Die IEC 61131-3 fordert an Stellen wie dieser immer eine strenge „Typverträglichkeit“, welche das Programmiersystems sicher zu stellen hat. Daher wird in Bsp. C.1 eine Standard-Funktion zur Typumwandlung aufgerufen (WORD_TO_BYTE).

Bsp. C.2 zeigt den Anweisungsteil von ByteExtr in der Programmiersprache ST.

```
FUNCTION  ByteExtr : BYTE     (* Extraktion Byte aus Wort *)
VAR_INPUT ... END_VAR         (* wie oben *)
IF Oben THEN
  ByteExtr := WORD_TO_BYTE (SHR (Wort, 8) );
ELSE
  ByteExtr := WORD_TO_BYTE (Wort);
END_IF;
END_FUNCTION
```

Bsp. C.2. Anweisungsteil von Bsp. C.1 in ST



Bsp. C.3. Grafischer Deklarationsteil der Funktionsdeklaration zu Bsp. C.1 (links) mit einem Aufruf-Beispiel (rechts)

Bsp. C.3 zeigt den Deklarationsteil und einen Beispiel-Aufruf der Funktion ByteExtr in grafischer Darstellung. Der Aufruf ersetzt das untere Byte des Merkerworts 4 durch sein oberes Byte.

C.2 Beispiel für FUNCTION_BLOCK

Der Funktionsbaustein DivMRest berechnet das Divisionsergebnis zweier ganzer Zahlen und liefert sowohl das Divisionsergebnis als auch den Divisionsrest zurück. In einem Ausgangsflag wird „Division durch Null“ angezeigt.

```
FUNCTION_BLOCK DivMRest     (* Division mit Rest *)
                            (* Beginn Deklarationsteil *)
VAR_INPUT                   (* Eingangsparameter *)
  Dividend :   INT;         (* zu teilende ganze Zahl *)
  Divisor  :   INT;         (* ganzzahliger Teiler *)
END_VAR
VAR_OUTPUT RETAIN           (* batteriegepufferte Ausgangsparameter *)
  Quotient :   INT;         (* Ergebnis der Division *)
  DivRest  :   INT;         (* Divisionsrest *)
  DivFehler:   BOOL;        (* Flag für Division durch Null *)
END_VAR
                            (* Beginn Anweisungsteil *)
LD      0                   (* Null laden *)
EQ      Divisor             (* Ist Divisor Null? *)
JMPC    Fehler              (* Fehlerfall abfangen *)
LD      Dividend            (* Dividenden laden, Divisor ungleich Null *)
DIV     Divisor             (* Division durchfuehren *)
ST      Quotient            (* ganzzahliges Divisionsergebnis speichern *)
MUL     Divisor             (* Divisionsergebnis mit Divisor multiplizieren *)
ST      DivRest             (* Zwischenergebnis merken *)
LD      Dividend            (* Dividenden laden *)
SUB     DivRest             (* Zwischenergebnis davon abziehen *)
ST      DivRest             (* ergibt den ganzzahligen „Rest“ der Division *)
LD      FALSE               (* logisch „0“ für Fehlerflag: ruecksetzen *)
ST      DivFehler           (* Fehlerflag ruecksetzen *)
JMP     Ende                (* fertig, Sprung auf FB-Ende *)
Fehler:                     (* Behandlung des Fehlers „Division durch Null“ *)
LD      0                   (* Null, da Ausgaenge im Fehlerfall ungueltig sind *)
ST      Quotient            (* Resultat ruecksetzen *)
ST      DivRest             (* Rest ruecksetzen *)
LD      TRUE                (* logisch „1“ fuer Fehlerflag: setzen *)
ST      DivFehler           (* Fehlerflag setzen *)
Ende:
RET
                            (* FB-Ende *)
END_FUNCTION_BLOCK
```

Bsp. C.4. Beispiel für die Deklaration eines Funktionsbausteins in AWL

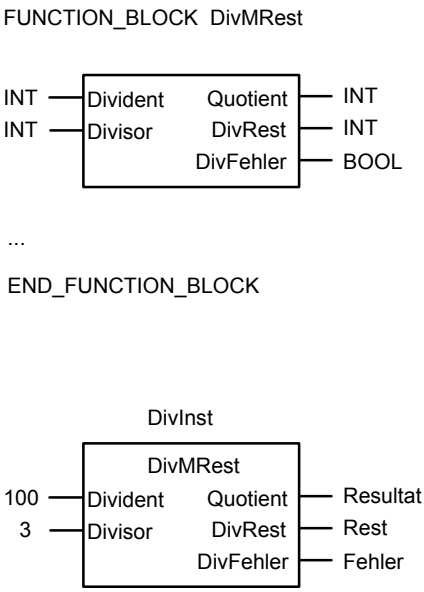
Der FB DivMRest in Bsp. C.4 berechnet die ganzzahlige Division mit Rest der beiden Eingangsparameter Dividend und Divisor. Bei Division durch Null wird der Fehler-Ausgang DivFehler gesetzt und die beiden anderen Ausgänge werden definiert auf Null gesetzt, da sie ungültig sind. Die Ausgänge sind batteriegepuffert, d.h. sie bleiben innerhalb der FB-Instanz erhalten, von der aus DivMRest aufgerufen wurde.

Dieses Beispiel kann auch als Funktion formuliert werden, da keine statische Information zwischen den Aufrufen erhalten bleiben muss.

Bsp. C.5 zeigt den Anweisungsteil von DivMRest in der Programmiersprache ST.

```
FUNCTION_BLOCK DivMRest       (* Division mit Rest *)
VAR_INPUT ... END_VAR         (* wie oben *)
VAR_OUTPUT RETAIN ... END_VAR
IF Divisor = 0 THEN
  Quotient := 0;
  DivRest := 0;
  DivFehler := TRUE;
ELSE
  Quotient := Dividend / Divisor;
  DivRest := Dividend - (Quotient * Divisor);
  DivFehler := FALSE;
END_IF;
END_FUNCTION_BLOCK
```

Bsp. C.5. Anweisungsteil von Bsp. C.4 in ST



Bsp. C.6. Grafischer Deklarationsteil zu Bsp. C.4 (oben) und ein Aufruf-Beispiel der FB-Instanz DivInst (unten)

Bsp. C.6 zeigt den Deklarationsteil und einen Beispiel-Aufruf der Funktion DivMRest in grafischer Darstellung. Der Aufruf setzt die Instanziierung dieses FB voraus (DivInst).

Die Ausgangsvariablen dieser Instanz besitzen nach Ausführung des FB die Werte: DivInst.Quotient = 33, DivInst.DivRest = 1 und DivInst.DivFehler = FALSE. Diese Rückgabewerte werden jeweils den Variablen Resultat, Rest und Fehler zugewiesen.

C.3 Beispiel für PROGRAM

Das Programm HptProg in Bsp. C.7 ist kein abgeschlossenes Programmierbeispiel, sondern zeigt Möglichkeiten zur Realisierung von Aufgaben und Benutzung von Variablen durch den POE-Typ PROGRAM.

HptProg startet zunächst eine Echtzeituhr DatUhr, die Datum und Uhrzeit mitführt (Hersteller-FB RTC: Realtime Clock). Anhand dieser Zeit wird festgestellt, wie lange eine Unterbrechung dieses Programm gedauert hat (ZeitDiff). Voraussetzung dafür ist, dass das SPS-System die Unterbrechung feststellen kann (Ress_Laeuft) und eine Hardware-Uhr von der E/A-Peripherie abfragen kann (AktDatum).

```
PROGRAM     HptProg    (* Beispiel für ein Hauptprogramm *)
VAR_INPUT                                (* Eingangsvariablen *)
  T_Start        :  BOOL := FALSE;       (* Eingang Startbedingung *)
END_VAR
VAR_OUTPUT                               (* Ausgangsvariablen *)
  T_Stoerung     :  BOOL := FALSE;       (* Ausgang „Stoerung“ *)
END_VAR
VAR_GLOBAL RETAIN                        (* globaler gepufferter Datenbereich *)
  Ress_Laeuft  AT %MX255.5 : BOOL;       (* Laeuft-Flag der Ressource/SPS-CPU *)
  DatUhr         :  RTC;                 (* Programm-Uhr: Datum mit Uhrzeit *)
  AktDatum AT %MD2 : DT;                 (* Hardware-Uhr: Akt. Datum mit Uhrzeit *)
END_VAR
VAR_GLOBAL                               (* globaler Datenbereich *)
  NotAus  AT %IX255.0 : BOOL;            (* Kontakt NOT-AUS *)
  ProgRun        :  BOOL := FALSE;       (* „laeuft“-Merker *)
  Fehler         :  BOOL;                (* Fehler-Merker *)
  F_Code         :  UDINT := 0;          (* Fehler-Code, 32 Bit ohne Vorzeichen *)
END_VAR
VAR                                      (* lokale Variablen *)
  AT %IX250.2 :  BOOL;                   (* direkt dargestellte Variable *)
  FehlerBeh      :  ZFehBeh;             (* FB-Instanz *)
  Flanke         :  R_TRIG;              (* Flankenerkennung *)
  ZeitDiff       :  TIME := t#0s;        (* Zeit-Differenz *)
END_VAR
```

Bsp. C.7. Beispiel für die Deklaration eines Hauptprogramms in AWL. Der FB ZFehBeh („Zentrale Fehler-Behandlung“) muss bereits vorhanden sein (wird fortgesetzt).

```
                                    (* Beginn Anweisungsteil *)
  LD    FALSE
  ST    Fehler                      (* Fehler ruecksetzen *)
  ...                     (* Dauer von Stromausfall bzw. Unterbrechung ermitteln *)
                          (* Uhr „DatUhr“ ist batteriegepuffert *)
  LD    t#0s                        (* Null Sekunden *)
  ST    ZeitDiff                    (* ruecksetzen *)
  LD    DatUhr.Q                    (* Zeit gueltig = Uhr laeuft *)
  JMPC  Weiter                      (* gueltig - nichts zu tun *)
  LD    AktDatum                    (* aktuelle Zeit abfragen *)
  SUB   DatUhr.CDT                  (* letzte Zeit vor Stromausfall *)
  ST    ZeitDiff                    (* Zeitdauer Stromausfall *)
Weiter:
  ...
  LD    Ress_Laeuft                 (* CPU laeuft *)
  ST    DatUhr.IN                   (* Uhr laeuft, falls CPU laeuft *)
  LD    AktDatum                    (* Anfangswert Datum/Uhrzeit *)
  ST    DatUhr.PDT                  (* Anfangswert Uhr wird geladen, *)
                                    (* falls steigende Flanke an IN *)
  CAL   DatUhr                      (* Starte Echtzeit-Uhr)
  ...
  ...
  LD    ZeitDiff                    (* Unterbrechungsdauer *)
  LT    t#50m                       (* weniger als 50 Sekunden? *)
  JMPC  MachtNichts                 (* Anlage noch warm genug... *)
  NOT
  S     Fehler                      (* Fehler setzen *)
  LD    16#000300F2
  ST    F_Code                      (* Fehlerursache merken *)
MachtNichts:
  LD    NotAus                      (* Not-Aus gedrueckt? *)
  ST    Flanke.CLK                  (* Eingang Flankenerkennung *)
  CAL   Flanke                      (* Vergleich Eingang mit Flankenmerker *)
  LDN   Flanke.Q                    (* Flanke an NotAus erkannt? *)
  AND   T_Start                     (* UND Start-Flag gesetzt *)
  ANDN  Fehler                      (* UND kein neuer Fehler *)
  AND   FehlerBeh.Quitt             (* Fehlerursache behoben *)
  ST    ProgRun                     (* Globale „laeuft“-Bedingung *)

  ....                    (* ...eigentliche Anweisungen, Aufrufe von FUN/FBs... *)

  LD    Fehler                      (* Fehler aufgetreten? *)
  AND   %IX250.2
  R     ProgRun                     (* Globale Startbedingung ruecksetzen *)
  LD    ProgRun
  JMPNC Ende                        (* bei Fehler: starte Fehlerbehandlung *)
  CALC  FehlerBeh (Code := F_Code)  (* FB mit zentraler Fehlerbehandlung *)
  LD    FehlerBeh.Quitt             (* Flag: Fehler wurde quittiert *)
Ende:
  LD    ProgRun                     (* Programm laeuft nicht *)
  ST    T_Stoerung                  (* Ausgangsparameter setzen *)
  RET                               (* Rueecksprung bzw. Ende *)
  END_PROGRAM
```

Bsp. C.7. (Fortsetzung)

In diesem Programm wird weiterhin eine Flankenerkennung verwendet, um festzustellen, ob die NotAus-Taste betätigt wurde.

Im globalen Datenbereich wird die Variable ProgRun deklariert, die auch allen unterhalb von HptProg aufgerufenen FBs zur Verfügung steht (dort als VAR_EXTERNAL). Mit ihr wird die Startbedingung verknüpft unter der Bedingung, dass kein NotAus gegeben wurde.

Die FB-Instanz FehlerBeh kann die Bearbeitung eines aufgetretenen Fehlers mit Fehlercode F_Code übernehmen. Wenn der Fehler behoben und quittiert wurde, wird der entsprechende Ausgang Quitt auf TRUE gesetzt.

```
RESOURCE ZentralEinh_1 ON CPU_001
  TASK Periodisch (INTERVAL := time#13ms, PRIORITY := 1);
  PROGRAM Applik WITH Periodisch : HptProg (  T_Start := %I250.0,
                                              T_Stoerung => %Q0.5);
END_RESOURCE
```

Bsp. C.8. Ressource-Definition mit Laufzeitprogramm Applik zu Bsp. C.7

Das Programm HptProg wird mit den Eigenschaften der periodischen Task Periodisch versehen und dadurch zum Laufzeitprogramm Applik der Ressource (SPS-CPU) ZentralEinh_1. Dieses Laufzeitprogramm läuft mit höchster Priorität (1) als zyklische Task mit einer maximalen Zykluszeit von 13 ms ab.

HptProg wird mit dem Wert des Peripheriebits %I250.0 für die Eingangsvariable T_Start aufgerufen und setzt mit T_Stoerung das Ausgangsbit %Q0.5.

D Standard-Datentypen

Dieser Anhang fasst sämtliche Elementaren Datentypen zusammen und charakterisiert tabellenartig ihre Eigenschaften. Ihre Verwendung wird in Kap. 3 erläutert.

Die IEC 61131-3 definiert fünf Gruppen von Elementaren Datentypen, zu denen in Klammern der Allgemeine Datentyp angegeben wird (siehe Tab. 3.9):

- Bitfolge (ANY_BIT),
- Ganzzahl mit und ohne Vorzeichen (ANY_INT),
- Gleitpunkt (ANY_REAL),
- Datum, Zeitpunkt (ANY_DATE),
- Zeichenfolge, Zeitdauer (ANYSTRING, TIME).

Zu jedem dieser Datentyp-Gruppen wird im folgenden eine Tabelle angegeben, die seine Eigenschaften charakterisiert:

- Name (Schlüsselwort),
- Stichworte (Kurzbeschreibung),
- Anzahl der Bits (Datenbreite),
- Werte-Bereich (unter Verwendung der IEC-Literale),
- Anfangswerte „initial“ (Standardwerte).

Die Datenbreite der Datentypen in Tab. D.5 und Tab. D.6 ist ebenso wie der zulässige Wertebereich implementierungsabhängig.

Datentyp	Stichworte	Bits	Bereich	initial
BOOL	boolesch	1	[0,1]	0
BYTE	Bitfolge 8	8	[0,...,16#FF]	0
WORD	Bitfolge 16	16	[0,...,16#FFFF]	0
DWORD	Bitfolge 32	32	[0,...,16#FFFF FFFF]	0
LWORD	Bitfolge 64	64	[0,...,16#FFFF FFFF FFFF FFFF]	0

Tab. D.1. Datentypen „Binär und Bitfolge“

Datentyp	Stichworte	Bits	Bereich	initial
SINT	kurze Ganzzahl	8	[-128,...,+127]	0
INT	Ganzzahl	16	[-32768,...,+32767]	0
DINT	Doppelte Ganzzahl	32	$[-2^{31},...,+2^{31}-1]$	0
LINT	Lange Ganzzahl	64	$[-2^{63},...,+2^{63}-1]$	0

Tab. D.2. Datentypen „Ganzzahl mit Vorzeichen“

Datentyp	Stichworte	Bits	Bereich	initial
USINT	kurze Ganzzahl	8	[0,...,+255]	0
UINT	Ganzzahl	16	[0,...,+65535]	0
UDINT	Doppelte Ganzzahl	32	$[0,...,+2^{32}-1]$	0
ULINT	Lange Ganzzahl	64	$[0,...,+2^{64}-1]$	0

Tab. D.3. Datentypen „Ganzzahl ohne Vorzeichen“

Datentyp	Stichworte	Bits	Bereich	initial
REAL	Gleitpunktzahl	32	s. IEC 60559	0.0
LREAL	lange Gleitpunktzahl	64	s. IEC 60559	0.0

Tab. D.4. Datentypen „Gleitpunktzahl“

Datentyp	Stichworte	initial
DATE	Datum	d#0001-01-01
TOD	Uhrzeit	tod#00:00:00
DT	Datum mit Uhrzeit	dt#0001-01-01-00:00:00

Tab. D.5. Datentypen „Datum und Zeiten“

Anstelle der Bezeichnung TOD kann gleichwertig auch das Schlüsselwort TIME_OF_DAY, anstelle von DT auch DATE_AND_TIME benutzt werden.

Datentyp	Stichworte	initial
TIME	Zeitdauer	t#0s
STRING	Zeichenfolge (Einzelbytes)	''
WSTRING	Zeichenfolge (Doppelbytes)	""

Tab. D.6. Datentypen „Zeitdauer und Zeichenfolge“. Die Anfangswerte für STRING und WSTRING sind die „leeren“ Zeichenfolgen.

E Fehlerursachen

Die IEC 61131-3 fordert vom Hersteller eine Liste, in der er seine Fehlerreaktion zu den nachfolgenden Fehlersituationen beschreibt; siehe dazu auch Abschn. 7.9. Die Meldung erfolgt auf vier verschiedene Arten:

1) keinerlei Systemreaktion (%),
2) als Warnung während der Programmerstellung (PeW),
3) als Fehlermeldung während der Programmerstellung (PeF),
4) Fehlermeldung und Fehlerreaktion während der Laufzeit (LzF).

Nr.	Fehlerursache	Zeitpunkt
1	Geschachtelter Kommentar	PeF
2	Wert einer Aufzählungstyp ist mehrdeutig	PeF
3	Wert einer Variablen übersteigt den festgelegten Bereich.	LzF
4	Adressauflösung in der Konfiguration nicht vollständig („*“ Angabe)	PeF
5	Schreibzugriff auf eine CONSTANT deklarierte Variable	PeF
6	Variable ist als VAR_GLOBAL CONSTATNT deklariert und wird als VAR_EXTERNAL ohne CONSTATNT referenziert.	PeF
7	Referenzierung einer Direkt dargestellten oder Externen Variablen in einer Funktion	PeF
8	Unzulässige Zuweisung an eine VAR_IN_OUT Variable (z.B. einer Konstanten)	PeF
9	Die Zuweisung an eine VAR_IN_OUT Variable ist nicht eindeutig	PeF
10	Typ-Verletzung bei Konvertierung.	PeF
11	Numerisches Ergebnis einer Standard-Funktion überschreitet den Wertebereich eines Datentyps; Division durch Null einer Standard-Funktion.	LzF PeF, LzF
12	Bitschiebefunktion mit negativer Shiftanzahl	LzF
13	Unterschiedliche Eingangsdatentypen bei einer Auswahlfunktion (Std.-FUN); Selektor (K) ist außerhalb des Bereichs der MUX-Funktion.	PeF LzF

Tab. E.1. Fehlerursachen; der Hersteller liefert eine Tabelle mit, in der die Systemreaktion auf oben beschriebene Fehlerfälle vermerkt ist. Der angegebene Zeitpunkt ist die gemäß IEC 61131-3 geeignete Stelle (wird fortgesetzt).

14	Zeichenfolge- Funktionen: Ungültige Angabe einer Zeichenposition; Ergebnis überschreitet die maximale Zeichenkettenlänge (INSERT/CONCAT Ergebnis ist zu lang). Ein ANY_INT Eingang ist negativ.	 PeW, LzF PeW, LzF LzF
15	Ergebnis überschreitet den Bereich des Datentyps „Zeit“.	LzF
16	Die als Eingangsparameter übergebene FB-Instanz besitzt keine Parameterwerte.	PeF
17	Ein VAR_IN_OUT Parameter besitzt keinen Wert.	PeF
18	Kein oder mehr als ein Anfangsschritt in einem AS-Netzwerk; Anwenderprogramm versucht, den Schritt-Merker oder -Zeit zu ändern.	PeF PeF
19	Seiteneffekte bei der Auswertung von Transitionsbedingungen.	PeF
20	Fehler innerhalb des Aktionskontrollblocks.	PeW, LzF
21	Auswahl einer AS Auswahlverzweigung besitzt mehrere erfüllte Transitionen, ohne dass eine Priorisierung vorhanden ist.	PeW, PeF
22	Ein unsicheres oder nicht erreichbares AS- Netzwerk existiert.	PeW
23	Typ-Konflikt bei VAR-ACCESS Variablen.	PeF
24	Anforderungen einer Task bzgl. Prozessor-Ressourcen zu hoch; Ausführungsende wurde nicht erreicht.	PeF PeW, LzF
25	Das numerische Ergebnis einer Operation überschreitet den Datentypbereich (AWL).	PeW, LzF
26	Das Aktuelle Ergebnis und der Operandentyp stimmen nicht überein (AWL).	PeF
27	Division durch NULL (ST); Ungültiger Datentyp einer Operation (ST). Das numerische Ergebnis einer Operation überschreitet den Datentypbereich (ST).	PeW,PeF,LzF PeF LzF
28	Rücksprung einer Funktion ohne Funktionswert (ST).	PeF
29	Iterationsschleife terminiert nicht (ST).	PeW,PeF,LzF
30	Doppelter Bezeichner für Marke und Elementname (KOP / FBS).	PeF
31	Nicht initialisierte Rückkopplungsvariable	PeF

Tab. E.1. (Fortsetzung)

Diese Tabelle ist als Anhaltspunkt zu sehen. Es gibt weitere Fehlermöglichkeiten, die in der IEC-Tabelle nicht aufgeführt sind. Aus diesem Grund sollte sie von jedem Hersteller in geeigneter Form erweitert werden.

F Implementierungsabhängige Parameter

Tab. F.1 gibt die implementierungsabhängigen Parameter wieder, die durch die IEC 61131-3 benannt werden; siehe auch Abschn. 7.10.

Nr.	Implementierungsabhängige Parameter
1	Maximale Zeichenanzahl von Bezeichnern.
2	Maximale Länge eines Kommentars (ohne führende/schließende Klammerzeichen).
3	Syntax und Semantik von Pragmas
4	Syntax und Semantik für ", wenn keine Doppelbytestrings damit deklariert werden
5	Wertebereich und Genauigkeit von Variablen des Datentyps TIME (z.B. 0<=TIME_VAR<1000 Tage), DATE, TIME_OF_DAY und DATE_AND_TIME Genauigkeit der Sekundenangaben bei Datentypen TIME_OF_DAY und DATE_AND_TIME.
6	Maximale - Anzahl von Elementen in einem Aufzählungs- Datentyp, - Anzahl von Elementen in einem Feld (wie viele Datenelemente), - Feldgröße (wie viele Bytes kann ein Array max. besitzen), - Anzahl von Elementen in einer Struktur, - Strukturgröße (wie viele Bytes kann eine Struktur max. besitzen), - Anzahl von Variablen pro Deklaration, die (durch Komma getrennt) mit einem Datentyp deklariert werden können, - Anzahl der Schachtelungstiefe von Strukturen.
7	Maximale Standardlänge von String- und WString Variablen; Maximale Länge von String- und WString Variablen.
8	Anzahl der Hierarchie-Ebenen bei direkten Variablen (%QX1.1.1.2...); Abbildung der Namen (%IX....) auf die reale Hardware.

Tab. F.1. Implementierungsabhängige Parameter, die jeder Hersteller eines IEC 61131-3 Systems beschreiben muss. Die Gliederung in einzelne Gruppen bezieht sich auf einzelne Abschnitte in der Norm. Werden keine konkreten Zahlen gefordert, kann der Hersteller eine informelle Beschreibung beifügen (wird fortgesetzt).

9	Initialisierungswert der Eingänge (%IX...) beim Systemstart.
10	Maximale Anzahl von Variablen pro Deklarationsblock
11	Verhalten eines AT- Bezeichners als Teil einer FB-Instanz- Deklaration
12	Verhalten beim Warmstart, wenn die Variable weder mit RETAIN oder NON_RETAIN deklariert wurde
13	Information zur Bestimmung der Ausführungszeiten von POEs (es sind keine näheren Angaben in der IEC 61131-3 vermerkt).
14	Werte der Baustein- Ausgänge, wenn der Baustein mit ENO=FALSE beendet wurde (Zuweisung ja/ Nein)
15	Max. Anzahl möglicher Anwenderfunktionen (soweit Begrenzung vorhanden).
16	Max. Anzahl von Eingängen bei erweiterbaren Funktionen.
17	Genauigkeit der einzelnen Typ-Umwandlungsfunktionen (REAL_TO_INT,...) sowie implementierte Fehlerbehandlung
18	Genauigkeit der numerischen Funktionen
19	Ergebnis von Typkonvertierungen zwischen Zeit- Datentypen und den restlichen Datentypen
20	Maximale Anzahl von Anwender-Funktionsbausteinen und möglichen Instanzen (soweit Begrenzung vorhanden).
21	Zuweisungsverhalten von Baustein Eingängen, wenn EN= FALSE (Zuweisung ja/ Nein)
22	Wertebereich des Parameters PV (Endewert bei Standard-Zählerbausteinen).
23	Reaktion des Systems auf eine Änderung des PT-Eingangs (Endewert bei Standard-Zeitbausteinen) während einer laufenden Zeitoperation .
24	Maximale Länge von Programmen (ausführbarer Code).
25	Zeitauflösung der Variablen Schritt-Zeit.
26	Maximale Anzahl von Schritten pro AS-Netzwerk.
27	Maximale Anzahl von Transitionen pro AS-Netzwerk und pro Schritt.
28	Maximale Anzahl von Aktionsblocks pro Schritt.
29	Funktionsweise der Aktions-Steuerung (ACTION-CONTROL Baustein soweit vorhanden) und Zugriffsmöglichkeit auf Werte des Q- und A-Ausgangs
30	Minimale Transitionsschaltzeit (verursacht durch Berechnungszeit der SPS),
31	Maximale Anzahl von Vorgänger und Nachfolger Schritten (bei Simultan- / Alternativ Verzweigungen).
32	Beschreibung der Ressource-Bibliothek. Jede Verarbeitungseinheit (z.B. Prozessortyp) erhält eine Beschreibung aller Funktionen (Std-FBs, Std.-Funktionen, Datentypen, ...), die von dieser Ressource verarbeitet werden können.
33	Wirkung der Verwendung des READ_WRITE Zugriffs auf FB Ausgänge
34	Maximale Anzahl von Tasks / Ressource; Task Intervall Auflösung (Zeitbereich möglicher Aufrufe für periodische Tasks); Art der Task-Steuerung (mit / ohne Pre-emptive Scheduling).

Tab. F.1. (Fortsetzung)

35	Maximallänge von ST Ausdrücken (Anzahl von Operanden, Operatoren).
36	Maximallängen von ST Anweisungen (IF; CASE; FOR; ...), Einschränkungen.
37	Maximale Anzahl von möglichen CASE Alternativen.
38	Wert der Laufvariablen von FOR-Schleifen nach Verlassen der Schleife.
39	Einschränkungen bzgl. der Netzwerktopologie bei KOP /FBS.
40	Auswertungsreihenfolge von Rückkopplungsschleifen.

Tab. F.1. (Fortsetzung)

Darüber hinaus gibt es eine Vielzahl weiterer Parameter, die bei der Implementierung eines Programmiersystems nach IEC 61131-3 zu beachten sind.

Aus dieser Menge stellt die Tab. F.2 eine subjektive Auswahl zusammen.

Nr.	Implementierungsabhängige Parameter
1	Umfang der syntaktischen und semantischen Überprüfung, die von den textuellen und grafischen Editoren angeboten wird (bereits während der Eingabe).
2	Beschreibung der Ausführungsreihenfolge von Netzwerken.
3	Freie Platzierung von grafischen Elementen bei Editoren mit Linien-Aktualisierung („Gummiband“) oder automatische Platzierung nach syntaktischen Regeln.
4	Deklaration und Verwendung Direkt dargestellter Variablen (DdV): - Die Deklaration erfolgt immer mit symbolischem Namen; im Anweisungsteil darf nur das Symbol *oder* beide (Symbol; direkte Adresse) verwendet werden oder - DdV können auch ohne symbolische Namen deklariert werden (Verwendung nur der direkten Adresse) oder - DdV werden vom Programmiersystem implizit deklariert.
5	Reihenfolge der VAR_*...VAR_END Blöcke, sowie Mehrfachverwendung gleichnamiger Blöcke.
6	Realisierte Funktions-Aufrufmöglichkeiten (mit / ohne Formal-Parameter) in ST.
7	Umfang der Realisierung der EN/ENO Parameter in KOP und FBS, sowie mögliche Auswirkungen auf AWL/ST-Quellen.
8	Möglichkeit der Übergabe von benutzerdefinierten Strukturen (TYPE) als Funktions- und FB-Parameter (derzeit nicht in der Norm festgelegt).
9	Bekanntmachung von Anwender definierten Datentypen (TYPE...END_TYPE): global- und POE-weit (derzeit nicht in der Norm festgelegt).
10	Umfang der Datenbereichsüberprüfung bei der Programm-Erstellung und während der Laufzeit.
11	Grafische Darstellung der Bestimmungszeichen/Attribute bei der Variablen-deklaration.

Tab. F.2. Weitere implementierungsabhängige Parameter (nicht Teil der IEC 61131-3) (wird fortgesetzt)

12	Umfang der Datenbereichsüberprüfung bei der Programm-Erstellung und während der Laufzeit.
13	Einschränkung in der Verwendung komplexer Datentypen (Funktionstyp auch für Typ „String“ oder anwenderdefinierte Typen zugelassen, ...?).
14	Algorithmus zur Auswertung von KOP/FBS-Netzwerken (siehe Abschn. 4.3.4 und 7.3.1).
15	Hilfen zur Nachvollziehbarkeit der Querübersetzung (soweit vorhanden).
	...

Tab. F.2. (Fortsetzung)

Die IEC 61131-3 erlaubt Funktionalitäten, die über den Standard hinausgehen. Sie müssen allerdings beschrieben sein. Einige davon wurden für Tab. F.3 zusammengestellt.

Nr.	Erweiterungen
1	Ausdehnung der Wertebereichsüberprüfung auf mehrere Datentypen (z.B. ANY_NUM) als nur Integer (ANY_INT).
2	FB-Instanzen als Feld-Variable zulassen.
3	FB-Instanzen in Strukturen erlauben.
4	Überladen auch für benutzerdefinierte Funktionen, FBs, und Programme.
5	Zeitpunkt der Berechnung der Schrittweite bei FOR-Anweisungen.
6	Möglichkeit der Verwendung von Präprozessor-Anweisungen für Literale, Makros, bedingte Übersetzung, Include-Anweisungen (zum Einlesen von Dateien mit FB-Schnittstelleninformation, mit der Liste der verwendeten Direkt darstellbaren Variablen oder EXTERNAL-Deklarationen, ...).
7	Verwendung unterschiedlicher Speichermodelle in der SPS (Small; Compact; ...).
	...

Tab. F.3. Mögliche Erweiterungen (nicht Teil der IEC 61131-3)

G Beispiel einer AWL-Syntax

Viele der in diesem Buch aufgeführten Beispiele sind in Anweisungsliste (AWL) formuliert. Diese Programmiersprache ist weit verbreitet und wird von den meisten Programmiersystemen unterstützt. Durch die Möglichkeit, Datentypen verwenden zu können, die meist nur in Hochsprachen zu finden waren wie Felder oder Strukturen, ergeben sich mit der IEC 61131-3 gegenüber herkömmlichem AWL neue Möglichkeiten.

Die nachfolgend beschriebene AWL-Syntax wird mit Hilfe von *Syntaxgraphen* in vereinfachter Form wiedergegeben.

Syntaxgraphen erläutern die zulässige Verwendung von Begrenzungszeichen, Schlüsselworten, Literalen und Bezeichnern auf anschauliche Weise. Sie können zur Entwicklung von Compilern leicht in eine textuelle Form gebracht werden und definieren so den formalen Aufbau einer Programmiersprache (*Syntax* einer Sprache). Der Anwender kann sich ihrer bedienen, um nachzuschlagen, wie die von ihm zu programmierenden Sprachkonstrukte aufzubauen sind.

Die Syntaxbeschreibung dieses Kapitels geht über die IEC 61131-3 hinaus, da zusätzlich zu der reinen Syntaxbeschreibung semantische Bedingungen (Folgerichtiger Gebrauch des Aktuellen Ergebnisses, Verwendung von Funktionsparametern,...) mit eingearbeitet sind; die IEC 61131-3 bietet hierfür lediglich eine informelle Beschreibung.

Die Regeln sind im Abschn. G.1 zu finden. Die Verwendung dieser Graphen wird im Abschn. G.2 anhand eines Beispiels erklärt.

G.1 Syntaxgraphen für AWL

Gibt es für einen Knoten des Syntaxgraphen eine weitere Untergliederung (Untergraphen), ist der Name dieses Knoten kursiv geschrieben. Schlüsselwörter oder terminale Symbole, die nicht weiter ersetzt werden, besitzen Normalschrift. Die Verwendung von Kommentaren wurde aus Gründen der Übersichtlichkeit weggelassen. Kommentar ist seit der Ausgabe 2 der Norm an jeder Stelle erlaubt, an der auch ein Leerzeichen (außer Zeichenketten) stehen kann. Ein Kommentar beginnt mit „(*“,endet mit „*)“ und enthält dazwischen beliebig viele alphanumerische Zeichen **ohne** EOL. Kommentare können nicht geschachtelt werden.

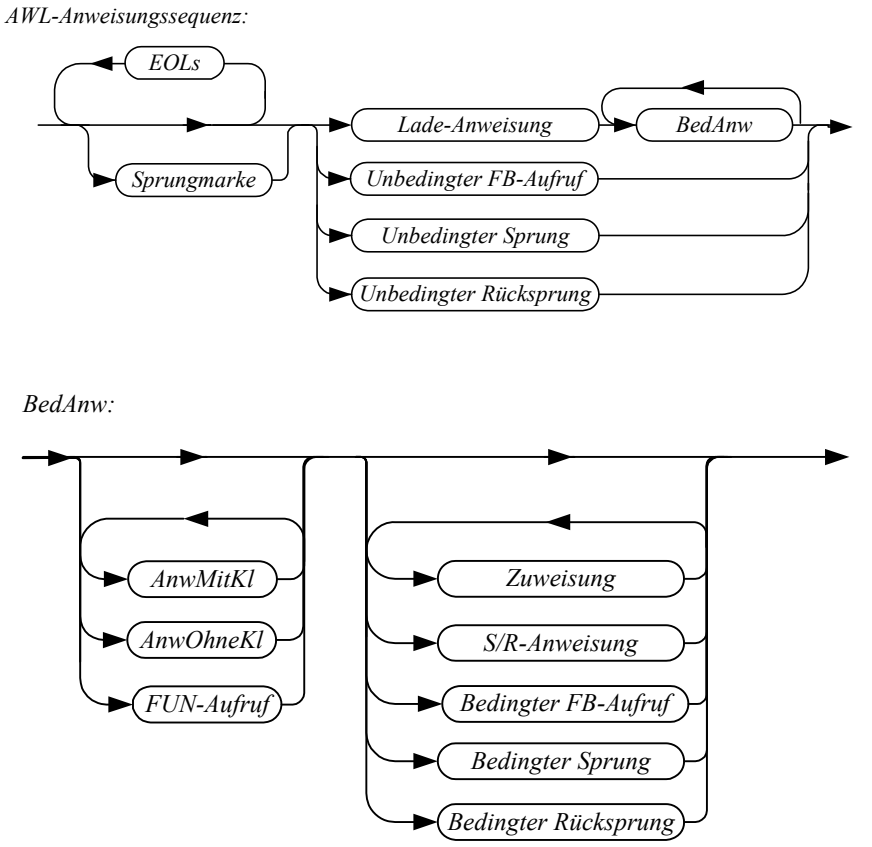


Abb. G.1. Syntaxgraphen einer AWL-Anweisungs-Sequenz mit den Unterelementen „Bedingte Anweisung“ und „Sprungmarke“ (wird fortgesetzt)

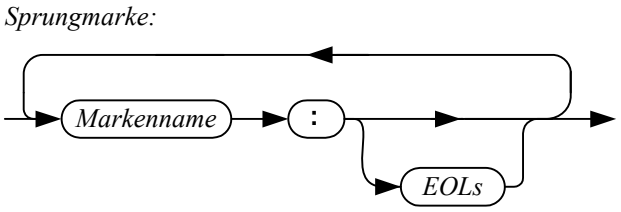


Abb. G.1. (Fortsetzung)

Eine AWL-Anweisungs-Sequenz beginnt optional mit einer Sprungmarke; es schließt sich entweder eine Lade-Anweisung, gefolgt von einer oder mehreren bedingten Anweisungen BedAnw, eine unbedingte Anweisung mit FB-Aufruf, Sprung oder Rücksprung an; siehe dazu den Syntaxgraphen der Abb. G.1.

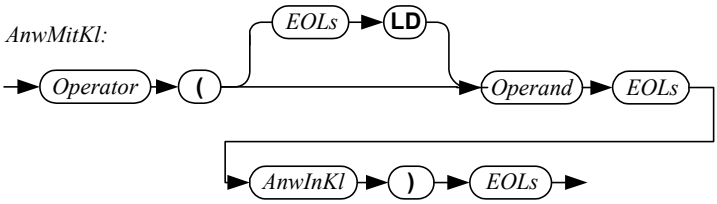
Die Bedingte Anweisung beginnt mit einer Folge von Anweisungen mit und ohne Klammern bzw. Funktionsaufrufen. Daran schließt sich eine Folge von (S/R-) Zuweisungen, bedingten Aufrufen oder Sprüngen an.

Die Sprungmarke besteht aus einer Marken-Bezeichnung, gefolgt von einem Doppelpunkt. Sie steht entweder unmittelbar vor der ersten Anweisung der Folge oder wird von EOLs (engl.: end-of-line) gefolgt.

Es folgen die Syntaxgraphen für Anweisungen, Aufrufe und Sprünge.

Abb. G.2. Syntaxgraph einer Anweisung ohne Klammer

Eine Anweisung ohne Klammer (Abb. G.2) besteht aus einem AWL-Operator mit einem Operanden und dem Zeilenabschluss durch EOLs.



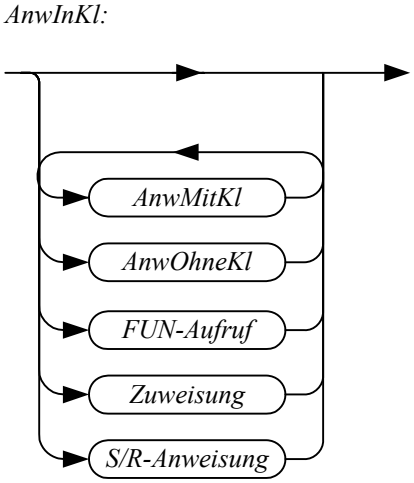
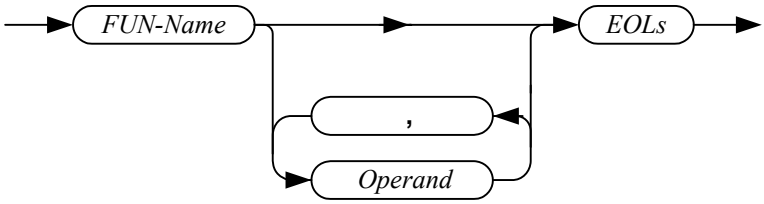


Abb. G.3. Syntaxgraphen einer Anweisung mit Klammer

Die in Abb. G.3 gezeigten Syntaxgraphen zeigen den Aufbau einer Anweisung mit Klammer. Diese Anweisungsart beginnt mit einem die Klammerung einleitenden Operator mit Operanden als erste Anweisung. Darauf folgen beliebig viele Anweisungen AnwInKl innerhalb der Klammer, die mit einer schließenden Klammer und EOLs abgeschlossen wird.

Diese inneren Anweisungen können wiederum Klammern einschließen (Schachtelung) sowie FUN-Aufrufe und Zuweisungen besitzen.

FUN-Aufruf mit Aktualparameter:



FUN-Aufruf mit Formalparameter:

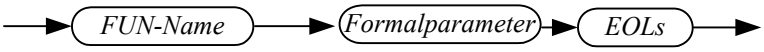
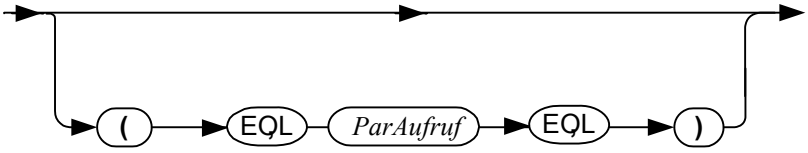


Abb. G.4. Syntaxgraph für den Aufruf einer Funktion

Formalparameter:



ParAufruf:

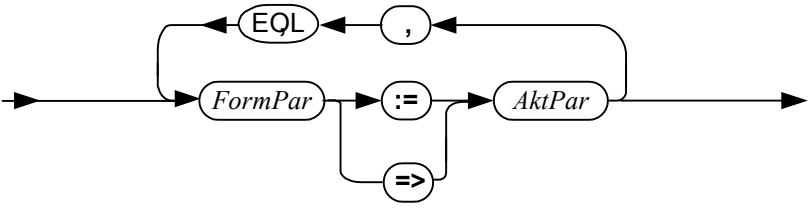
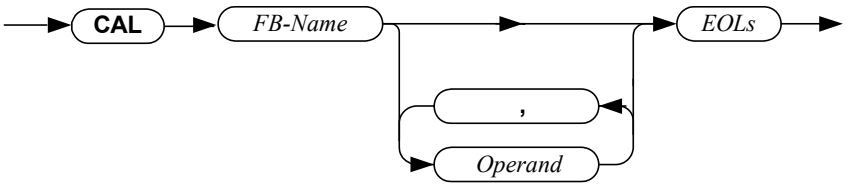


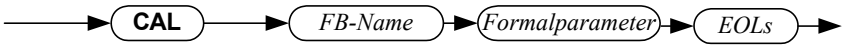
Abb. G.5. Formale Parameterübergabe

Wie Abb. G.4 zeigt, besteht der Aufruf einer Funktion aus der Angabe des Funktionsnamens sowie einer Anzahl durch Kommata getrennter Operanden als Aktualparameter dieser Funktion. Eine Funktion kann auch wie ein Funktionsbaustein mit Formalparametern aufgerufen werden.

FB-Aufruf mit Aktualparameter:



Unbedingter FB-Aufruf:



Bedingter FB-Aufruf:

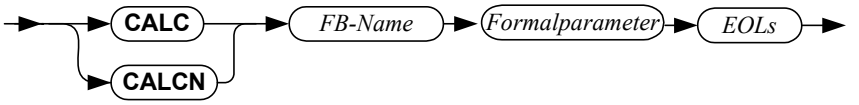


Abb. G.6. Syntaxgraphen für den Aufruf eines Funktionsbausteins

Die Syntaxgraphen für bedingten und unbedingten Aufruf eines Funktionsbausteins zeigt Abb. G.6. Der Aufruf wird beim unbedingten Aufruf mit CAL, beim bedingten mit CALC bzw. CALCN begonnen. Darauf folgen der Name der FB-Instanz sowie in Klammern die Parameter des FB.

Die Übergabe eines Aktualparameters an einen Formalparameter wird durch eine Zuweisung mit „:=“ dargestellt. Solche Zuweisungen sind für jeden Parameter erforderlich und werden durch Kommata und Zeilenwechsel getrennt.

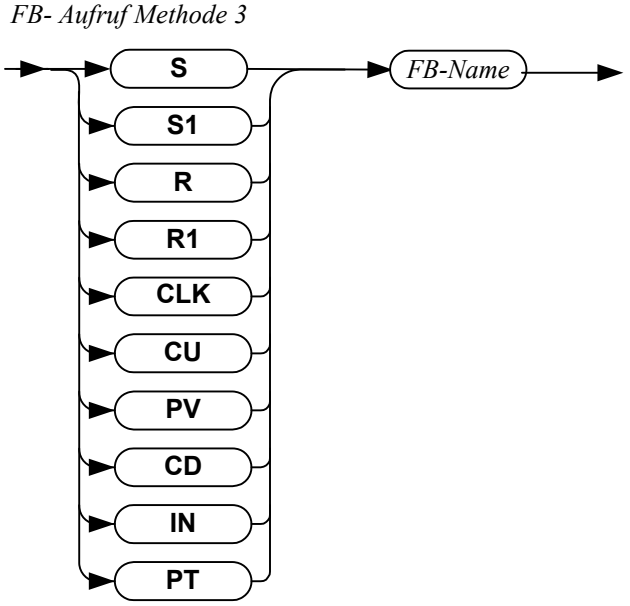


Abb. G.7. FB Aufruf mit Methode 3

Bei der in Abb. G.7 gezeigten Syntax ist darauf zu achten, dass dieser Aufruf je nach Parameter nur eine Parameterinitialisierung oder die Ausführung des FBs bewirkt.

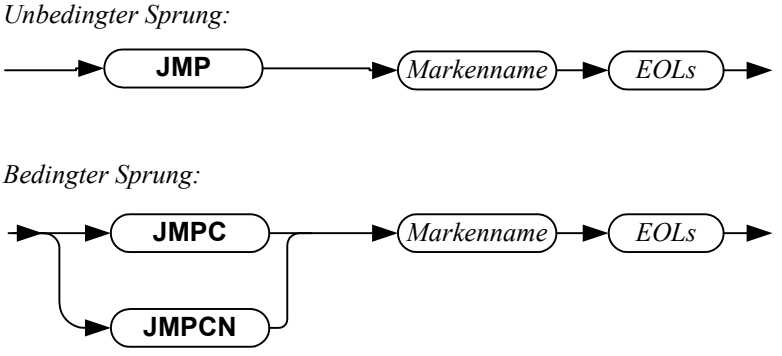
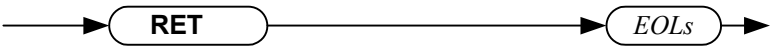


Abb. G.8. Syntaxgraphen für Bedingten und Unbedingten Sprung

Bei Sprüngen wird hinter dem Sprung-Operator JMP (unbedingt) bzw. JMPC oder JMPCN (bedingt) der Name der Sprungmarke angegeben (Abb. G.8).

Unbedingter Rücksprung:



Bedingter Rücksprung:

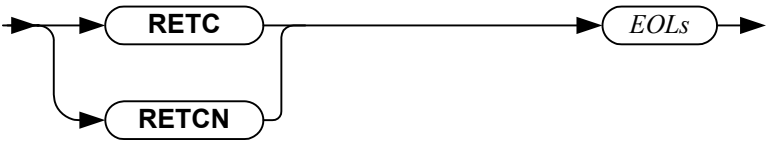


Abb. G.9. Syntaxgraphen für Bedingten und Unbedingten Rücksprung

Die in Abb. G.9 gezeigten Rücksprünge besitzen keine Operanden bzw. Parameter.

Lade-Anweisung:

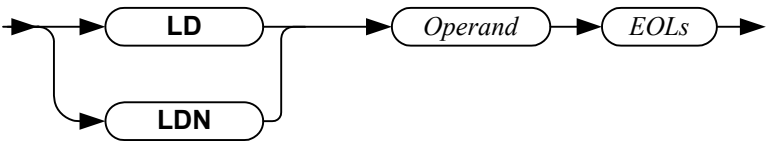


Abb. G.10. Syntaxgraph für die Lade-Anweisung

Die Lade-Anweisung der Abb. G.10 besitzt einen einzelnen (invertierbaren) Operanden. Sie kann nicht mit einer Klammer kombiniert werden oder innerhalb einer Klammerung verwendet werden.

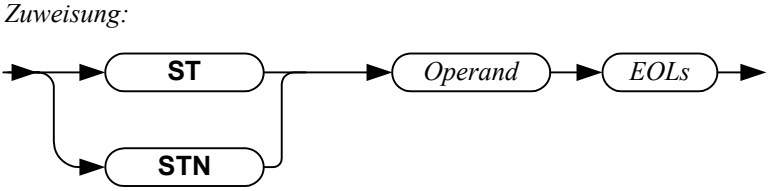


Abb. G.11. Zuweisung

Zuweisungen (Abb. G.11) bestehen aus dem Operator ST bzw. STN und der Angabe des zu speichernden Operanden.

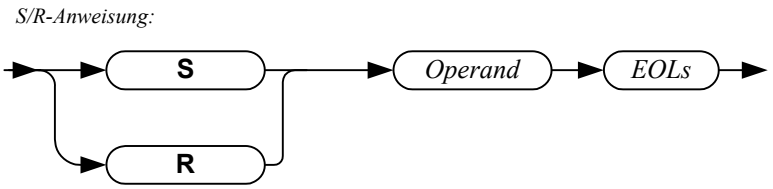


Abb. G.12. S/R-Anweisung

Eine S/R-Anweisung (Abb. G.12) besteht aus den AWL-Operatoren S oder R und einem Operanden.

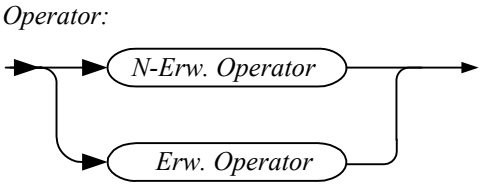


Abb. G.13. Operator als Zusammenfassung der Erweiterbaren und Nicht-Erweiterbaren Operatoren

Die in Abb. G.13 dargestellten Operatoren besitzen eine verknüpfende Funktion, dienen also nicht zum Laden oder Speichern. Es wird dabei zwischen Erweiterbaren und Nicht-Erweiterbaren Operatoren unterschieden, die im Folgenden dargestellt werden.

Erw. Operator:

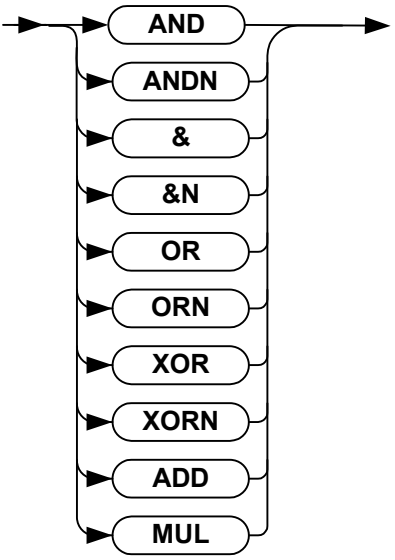


Abb. G.14. Erweiterbare Operatoren: Bitweise Operationen sowie Addition und Multiplikation

Die Erweiterbaren Operatoren zeigt Abb. G.14. Sie können mehr als zwei Eingangsparameter besitzen. Die bitweise booleschen Operatoren (Standard-Funktionen) können optional mit Invertierung verwendet werden.

N-Erw. Operator:

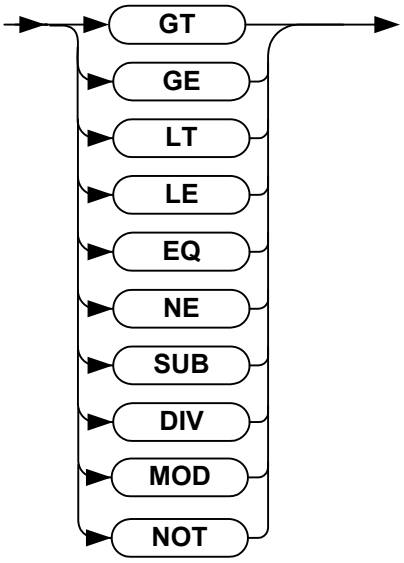


Abb. G.15. Nicht-Erweiterbare Operatoren: Vergleichen, Subtraktion, Division

Die Nicht-Erweiterbaren Operatoren in Abb. G.15 haben (einschließlich AE) genau zwei Eingangsparameter.

EOLs:

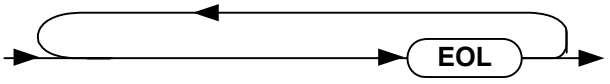


Abb. G.16. Syntaxgraph für das Zeilenende (engl.: end of line) einer AWL-Anweisung

Eine AWL-Zeile wird mit einem einzelnen EOL-Zeichen (z.B. Carriage Return / Line Feed), dem oft ein erklärender Zeilen-Kommentar vorangestellt ist, abgeschlossen (Abb. G.16). Diese Elemente können einmal oder beliebig oft hintereinander vorkommen.

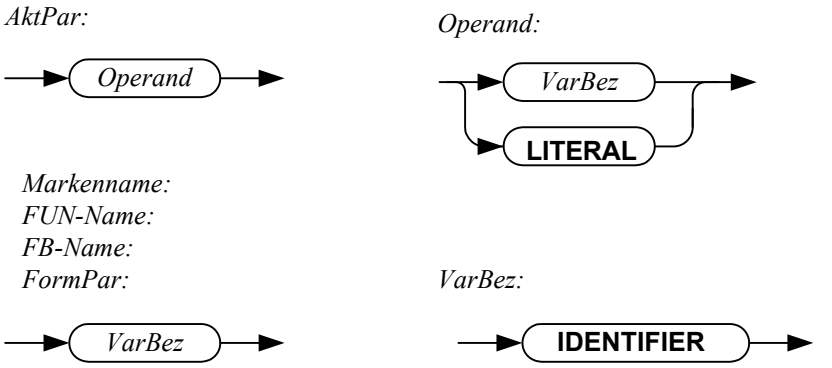


Abb. G.17. Operanden, Parameter und Bezeichner werden durch Identifier bzw. Literale gebildet.

Abb. G.17 zeigt die Abbildung von Parametern, Operand und verschiedenen Bezeichnertypen auf Bezeichner (IDENTIFIER) und Werte (LITERAL).

Zur Vereinfachung werden hier die Syntaxgraphen der Bezeichner und Literale nicht dargestellt. Ihre prinzipielle Darstellung ist in Abschn. 3.2 erläutert.

G.2 AWL-Beispiel zu Syntaxgraphen

Zu den soeben dargestellten AWL-Syntaxgraphen wird nun ein Beispiel gegeben. Dieses zeigt, wie aus Syntaxgraphen Programmbeispiele konstruiert werden und umgekehrt, wie nämlich AWL-Beispiele auf ihre Richtigkeit hin überprüft werden können.

```
                                    Beginn der AWL-Sequenz
0001        SequenzEins:            (* Sprungmarke *)
0002                                (* Einfache Verknüpfung mit Sprung *)
0003        LD      Var1            (* Lade-Anweisung *)
0004                                (* Beginn bedingte Anweisungen *)
0005        ANDN    Var2            (* Anweisung ohne Klammer *)
0006        ORN     ( Var3          (* Anweisung mit Klammer *)
0007        AND     Var4
0008        )                       (* Ende der Klammerung *)
0009        AND     Var5
0010        ST      Var6            (* Zuweisung *)
0011        S       Var7            (* S/R-Anweisung *)
0012        RETC                    (* Bedingter Rücksprung *)
0013                                (* Ende der bedingten Anweisungen *)
0014                                (* Ende der AWL-Sequenz *)
```

Bsp. G.1. AWL-Beispiel. Die Kommentare weisen auf den entsprechenden Syntaxgraphen hin. Die links stehenden Zeilennummern dienen der Referenzierung in Tab. G.1.

Um zu sehen, wie sich das AWL-Beispiel in Bsp. G.1 aus den Syntaxgraphen des Abschn. G.1 aufbauen lässt, werden in Tab. G.1 zu jeder AWL-Zeile die anzuwendenden Syntaxgraphen angegeben.

Zeile in Bsp. G.1	Syntaxgraph	Abbildung
0001-0014	AWL-Anweisungs-Sequenz	Abb. G.1
0001-0002	Sprungmarke	Abb. G.1
0003-0004	Lade-Anweisung	Abb. G.10
0005	Anweisung ohne Klammer	Abb. G.2
0005	Erweit. Operator ANDN	Abb. G.14
0006-0008	Anweisung mit Klammer	Abb. G.3
0006,0007	Erweit. Operatoren ORN, AND	Abb. G.14
0007,0009	Anweisung ohne Klammer	Abb. G.2
0007,0009	Erweit. Operator AND	Abb. G.14
0010	Zuweisung	Abb. G.11
0011	S/R-Anweisung	Abb. G.12
0012-0014	Bedingter Rücksprung	Abb. G.9

Tab. G.1. Anzuwendende Syntaxgraphen zu Zeilen der AWL-Sequenz in Bsp. G.1

Dieses Beispiel zeigt, wie sich durch mehrfache Anwendung der Syntaxgraphen ein konkretes AWL-Programm ergibt: Im Syntaxgraphen für eine AWL-Anweisungs-Sequenz (Abb. G.1) wird zunächst die Sprungmarke mit Bezeichner, Doppelpunkt und Kommentaren eingesetzt, gefolgt von der ersten (Lade-) Anweisung.

Der Teil „bedingte Anweisungen“ ergibt sich aus zwei Anweisungen, von denen die erste mit Klammer wiederum einige Anweisungen in Klammern enthält. Nach den bedingten Anweisungen wird die Sequenz mit Zuweisungen und Rücksprung abgeschlossen.

Auf diese Weise ist es möglich, aus den einzelnen Syntaxgraphen zulässige AWL-Sequenzen zu erzeugen. Ebenso können umgekehrt zu AWL-Zeilen die erforderlichen Syntaxgraphen gefunden werden, um festzustellen, ob ein Programmstück syntaktisch korrekt ist.

H Reservierte Schlüsselworte und Begrenzungszeichen

Es wird von der IEC 61131-3 ausdrücklich zugelassen, dass Schlüsselworte und Begrenzungszeichen durch Übersetzungstabellen an nationale Zeichensätze angepasst werden können.

H.1 Reservierte Schlüsselworte

In Tab. H.1 sind sämtliche Reservierten Schlüsselworte für Programmiersprachen der IEC 61131-3 in alphabetischer Reihenfolge aufgelistet. Sie dürfen **nicht** als Namen für benutzter-definierte Elemente verwendet werden. In dieser Aufzählung wurden zusätzlich zur Norm auch sämtliche Formalparameter von FUN bzw. FB sowie Datentypen mit aufgenommen, um Verwechselungen zu vermeiden.

A	ABS	ACOS
	ACTION	ADD
	AND	ANDN
	ANY	ANY_BIT
	ANY_DATE	ANY_DERIVED
	ANY_ELEMENTARY	ANY_INT
	ANY_MAGNITUDE	ANY_NUM
	ANY_REAL	ARRAY
	ASIN	AT
	ATAN	
B	BOOL	BY
	BYTE	

Tab. H.1. Reservierte Schlüsselworte der IEC 61131-3 (wird fortgesetzt)

C	CAL	CALC
	CALCN	CASE
	CD	CDT
	CLK	CONCAT
	CONFIGURATION	CONSTANT
	COS	CTD
	CTU	CTUD
	CU	CV
D	D	DATE
	DATE_AND_TIME	DELETE
	DINT	DIV
	DO	DS
	DT	DWORD
E	ELSE	ELSIF
	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	
F	FALSE	F_EDGE
	F_TRIG	FIND
	FOR	FROM
	FUNCTION	FUNCTION_BLOCK
G	GE	GT
I	IF	IN
	INITIAL_STEP	INSERT
	INT	INTERVAL

Tab. H.1. (Fortsetzung)

J	JMP	JMPC
	JMPCN	
L	L	LD
	LDN	LE
	LEFT	LEN
	LIMIT	LINT
	LN	LOG
	LREAL	LT
	LWORD	
M	MAX	MID
	MIN	MOD
	MOVE	MUL
	MUX	
N	N	NE
	NEG	NON_RETAIN
	NOT	
O	OF	ON
	OR	ORN
P	P	PRIORITY
	PROGRAM	PT
	PV	
Q	Q	Q1
	QU	QD
R	R	R1
	R_EDGE	R_TRIG
	READ_ONLY	READ_WRITE
	REAL	RELEASE
	REPEAT	REPLACE
	RESOURCE	RET
	RETAIN	RETC
	RETCN	RETURN
	RIGHT	ROL
	ROR	RS

Tab. H.1. (Fortsetzung)

S	S	S1
	SD	SEL
	SEMA	SHL
	SHR	SIN
	SINGLE	SINT
	SL	SQRT
	SR	ST
	STEP	STN
	STRING	STRUCT
	SUB	
T	T	TAN
	TASK	THEN
	TIME	TIME_OF_DAY
	TO	TOD
	TOF	TON
	TP	TRANSITION
	TRUE	TYPE
U	UDINT	UINT
	ULINT	UNTIL
	USINT	VAR
V	VAR_ACCESS	VAR_CONFIG
	VAR_EXTERNAL	VAR_GLOBAL
	VAR_INPUT	VAR_IN_OUT
	VAR_OUTPUT	VAR_TEMP
W	WHILE	WITH
	WORD	WSTRING
X	XOR	XORN

Tab. H.1. (Fortsetzung)

H.2 Begrenzungszeichen

Begrenzungszeichen stellen „Sonderzeichen“ in der Syntax der Programmiersprachen dar und besitzen je nach Verwendung unterschiedliche Bedeutung. So können runde Klammern beispielsweise zur Kennzeichnung von Anfang und Ende einer Aktualparameterliste bei einem Funktionsaufruf oder zusammen mit dem Stern zur Begrenzung von Kommentaren benutzt werden.

Sämtliche Begrenzungszeichen und deren Kombinationen werden in Tab. H.2 mit ihren möglichen Bedeutungen aufgeführt.

Zeichen zur grafischen Darstellung der Linienführung werden hier nicht berücksichtigt.

Begrenzungszeichen	**Bedeutung, Erläuterungen**
Leerzeichen	können überall eingefügt werden - außer innerhalb Schlüsselworten, Literalen, Bezeichnern, Direkt dargestellten Variablen oder Kombinationen von Begrenzungszeichen (wie „(*“ oder „*)“). Über Tabulatoren (TABs) macht die IEC 61131-3 keine Aussage, sie werden üblicherweise wie Leerzeichen behandelt.
Zeilenende	zum Abschluss einer Anweisungszeile in AWL, in ST auch innerhalb Anweisungen zulässig. Nicht zulässig innerhalb AWL-Kommentaren. (engl. EOL end of line), üblicherweise durch CR&LF (engl.: carriage return & Line feed) implementiert.
Kommentaranfang **(***	Beginn eines Kommentars (nicht schachtelbar)
Kommentarende ***)**	Ende eines Kommentars
Plus +	1. Führendes Vorzeichen eines Dezimal-Literals, auch im Exponenten eines Gleitpunkt-Literals 2. Additionsoperator in Ausdrücken
Minus -	1. Führendes Vorzeichen eines Dezimal-Literals, auch im Exponenten eines Gleitpunkt-Literals 2. Subtraktionsoperator in Ausdrücken 3. Negationsoperator in Ausdrücken 4. Jahr-Monat-Tag-Trennzeichen in Zeitliteralen
Nummernkreuz #	1. Basiszahl-Trennzeichen in Literalen 2. Zeitliteral-Trennzeichen

Tab. H.2. Begrenzungszeichen der IEC 61131-3 (wird fortgesetzt)

Begrenzungszeichen		Bedeutung, Erläuterungen
Punkt	**.**	1. Ganzzahl/Bruch-Trennzeichen 2. Trennzeichen innerhalb Hierarchischer Adressen von Direkt dargestellten und Symbolischen Variablen 3. Trennzeichen zwischen Komponenten einer Datenstruktur (beim Zugriff) 4. Trennzeichen für Komponenten einer FB-Instanz (beim Zugriff)
	e, E	führendes Zeichen für Exponenten der Gleitpunkt-Literale
Anführungszeichen	**'**	Anfang und Ende von Zeichenfolgen
Dollar-Zeichen	**$**	Anfang von Sonderzeichen innerhalb Zeichenfolgen
Präfix Zeitliterale	**t#, T#** **d#, D#** **d, D** **h, H** **m, M** **s, S** **ms, MS** **date#, DATE#** **time#, TIME#** **time_of_day#** **TIME_OF_DAY#** **tod#, TOD#** **date_and_time#** **DATE_AND_TIME#** **dt#, DT#**	Einleitende Zeichen von Zeitliteralen, Kombinationen von Klein- und Großbuchstaben sind ebenfalls zulässig.
Doppelpunkt	**:**	Trennzeichen für: 1. Uhrzeit innerhalb Zeitliterale 2. Festlegung des Datentyps bei Variablendeklaration 3. Festlegung eines Datentyp-Namens 4. Schrittnamen 5. PROGRAM...WITH... 6. Funktionsname/Datentyp 7. Zugriffspfad: Name/Typ 8. Sprungmarke vor nächster Anweisung 9. Netzwerkname (Marke) vor nächster Anweisung
Zuweisung	**:=**	1. Operator für Anfangswert-Zuweisung 2. Eingangsverbindungs-Operator (Zuweisung von Aktualparameter an Formalparameter beim POE-Aufruf) 3. Anweisungsoperator

Tab. H.2. (Fortsetzung)

Begrenzungszeichen		Bedeutung, Erläuterungen
Runde Klammerung	**(**...**)**	Anfang und Ende für: 1. Aufzählungsliste 2. Anfangswert-Liste, auch: Mehrfache Anfangswerte (mit Wiederholungszahl) 3. Bereichsangabe 4. Feldindex 5. Folgenlänge 6. Operator in AWL (Berechnungsebene) 7. Parameterliste beim POE-Aufruf 8. 6. Unterausdruck-Hierarchie
Eckige Klammern	**[**...**]**	Anfang und Ende für: 1. Feldindex (Zugriff auf ein Feldelement) 2. Zeichenfolgenlänge (bei Deklaration)
Komma	**,**	Trennzeichen für: 1. Aufzählungsliste 2. Anfangswert-Liste 3. Feldindex (mehrdimensional) 4. Variablennamen (bei Mehrfachdeklaration desselben Datentyps) 5. Parameterliste beim POE-Aufruf 6. Operandenliste in AWL 7. CASE-Werteliste
Semikolon	**;**	Abschluss von: 1. Definition eines (Daten)-Typs 2. Deklaration (z.B. Variablen) 3. ST-Anweisung
Punkt-Punkt	**..**	Trennzeichen für: 1. Bereichsangabe 2. CASE-Bereich
Prozent	**%**	Einleitendes Zeichen für Hierarchische Adressen bei Direkt dargestellten und Symbolischen Variablen
Zuweisung2	**=>**	Ausgangsverbindungs-Operator (Zuweisung von Formalparameter an Aktualparameter beim PROGRAM-Aufruf)
Vergleich	**>, <** **>=, <=,** **=, <>,**	Vergleichsoperatoren in Ausdrücken
Exponent	**	als Operator in Ausdrücken
Multiplikation	*	Multiplikations-Operator in Ausdrücken
Division	/	Divisions-Operator in Ausdrücken
Kaufmanns-UND	**&**	AND-Operator in Ausdrücken

Tab. H.2. (Fortsetzung)

I Glossar

Im Folgenden werden wichtige Begriffe und Abkürzungen erläutert, die in diesem Buch verwendet werden. Sie werden in alphabetischer Reihenfolge aufgeführt.

Begriffe, die durch die IEC 61131-3 festgelegt sind, sind entsprechend mit „IEC“ gekennzeichnet.

Abgeleiteter Datentyp	IEC	Mit Hilfe einer *Typdefinition* wird ein benutzerspezifischer *Datentyp* erzeugt, dessen Elemente aus *Elementaren* und/oder wiederum *Abgeleiteten Datentypen* bestehen.
Ablaufsprache	IEC	Ablaufsprache AS (engl.: Sequential Function Chart – SFC) ist eine Programmiersprache zur Beschreibung sequentieller und paralleler Steuerungsabläufe mit Zeit- und Ereignis-Steuerung.
AE		Abk. für *Aktuelles Ergebnis*
Aktion	IEC	Boolesche Variable oder eine Reihe von Anweisungen, die über einen *Aktionsblock* angesteuert werden können, in *AS*.
Aktionsblock	IEC	Aktivierungsbeschreibung von *Aktionen* in AS
Aktionskontrollblock	IEC	Steuereinheit für jede *Aktion* in AS, die über *Aktionsblöcke* die Eingangsbedingung für die zugeordnete Aktionsfreischaltung erhält.
Aktualparameter		Versorgung einer Eingangsvariablen (*Formalparameter*) einer *POE* mit einem aktuellen Wert.
Aktuelles Ergebnis	IEC	Zwischenergebnis in *AWL* von beliebigem *Datentyp*
Allgemeiner Datentyp	IEC	Zusammenfassung von *Elementaren Datentypen* zu Gruppen, um *Überladen von Funktionen* beschreiben zu können; wird auch „Generischer Datentyp“ genannt.
Anfangswert		Wert einer *Variable*, der bei ihrer Initialisierung vergeben wird.

Anweisungsliste	IEC	Anweisungsliste (engl.: Instruction List – IL) ist eine weit verbreitete Assembler-ähnliche Programmiersprache für SPS-Systeme.
AS	IEC	Abk. für *Ablaufsprache*
Aufzählung		Besonderer *Datentyp* zur Definition textueller Sequenzen als Wertebereich. Intern zumeist als ganzzahlige Werte realisiert.
AWL	IEC	Abk. für *Anweisungsliste*
Batteriepufferung		Fähigkeit einer SPS, bestimmte Datenbereiche bei Stromausfall gegen Verlust zu sichern. Die IEC 61131-3 verwendet dazu das Schlüsselwort RETAIN.
Baustein		Programmeinheit, aus denen SPS-Programme zusammengesetzt werden. Bausteine können oft unabhängig voneinander in die SPS geladen werden. vgl. *POE*. In anderen Programmiersprachen oft auch als „Unterprogramm“ bezeichnet.
Bereichsangabe		Angabe eines zulässigen Wertebereichs für einen *Datentyp* oder eine *Variable*.
CPU		Central Processing Unit (Zentraleinheit, z.B. einer SPS)
Datenbaustein		Global zur Verfügung stehender, aktivierbarer Datenbereich. vgl. *Baustein*. In der IEC 61131-3 gibt es dafür keine direkte Entsprechung, sie werden hier durch globale, strukturierte Datenbereiche bzw. *FB-Instanz*-Datenbereiche ersetzt.
Datentyp		definiert die Bitlänge und Eigenschaften des Wertebereichs einer *Variablen.*
Deklaration	IEC	Bekanntgabe von *Variablen* und *FB-Instanzen* in einem *Deklarationsblock* unter Angabe eines Bezeichners, des *Datentyps* bzw. *FB-Typs* sowie ggf. *Anfangswerte, Bereichsangabe* und Feldeigenschaften. Die Definition bzw. die Programmierung von *POEs* wird ebenfalls als *Deklaration* bezeichnet, da hierbei ihre Schnittstellen dem *Programmiersystem* bekannt gemacht werden.
Deklarationsblock	IEC	Zusammenfassung von *Deklarationen* einer Variablenart zu Beginn der *POE*.
Direkt dargestellte Variable	IEC	*Variable* ohne weiteren Bezeichner, die einer *Hierarchischen Adresse* entspricht.
E/A-Peripherie		Die zu einem *SPS-System* gehörenden Eingangs- und Ausgangs-Module mit ihren *Hierarchischen Adressen.*
Einzelelement-Variable	IEC	*Variable*, die auf einem einzelnen *Datentyp* basiert.

Elementarer Datentyp	IEC	Ein durch die IEC 61131-3 vordefinierter Standard-*Datentyp*.
Erweiterung von Funktionen	IEC	Eine *Funktion* kann eine variable Anzahl von Eingängen besitzen.
FB	IEC	Abk. für *Funktionsbaustein*
FB-Instanz	IEC	siehe *Instanz*
FB-Typ	IEC	Name eines *Funktionsbausteins* mit Aufruf- und Rückgabeschnittstelle
FBS	IEC	Abk. für *Funktionsbausteinsprache*
Feld		Aneinanderreihung von Elementen gleichen *Datentyps*.
Flanke		Mit einer „steigenden“ Flanke wird der 0→1-Übergang einer booleschen Variable bezeichnet. Eine „fallende“ Flanke ist dementsprechend der 1→0-Übergang.
Formalparameter		Name (Bezeichner) einer Eingangsvariablen (alle *POEs*) oder Ausgangsvariablen (*Funktionsbaustein* und *Programm*).
Funktion	IEC	Eine *POE* vom Typ FUNCTION
Funktionsbaustein	IEC	Eine *POE* vom Typ FUNCTION_BLOCK
Funktionsbausteinsprache	IEC	Funktionsbausteinsprache FBS (engl.: Function Block Diagram – FBD) ist eine grafische Programmiersprache zur Beschreibung von Netzwerken mit gleichzeitig arbeitenden booleschen und arithmetischen Elementen.
Hierarchische Adresse	IEC	Physikalische Steckplatzadressen der E/A-Module eines SPS-Systems (vgl. *E/A-Peripherie*).
Indirekter FB-Aufruf		Aufruf einer *FB-Instanz*, dessen Name als VAR_IN_OUT-Parameter einer *POE* übergeben wurde.
Instanz	IEC	Strukturierter Datensatz eines FBs durch *Deklaration* eines *Funktionsbausteins* unter Angabe des *FB-Typs*.
Kaltstart	IEC	Programmstart, bei dem sämtliche Variablen und Speicherbereiche (neu) initialisiert werden (engl.: Cold Restart). Dieser auch *Neustart* genannte Vorgang kann bei bestimmten Ereignissen automatisch oder auch manuell durch den Anwender erfolgen.
Kommentar		Zwischen Klammern und Sternchen eingeschlossener Text (nicht schachtelbar!) zur Erläuterung des Programms, wird vom *Programmiersystem* nicht interpretiert.
Konfiguration	IEC	Sprachelement CONFIGURATION, das einem *SPS-System* entspricht.
KOP	IEC	Abk. für *Kontaktplan*

Kontaktplan	IEC	Kontaktplan (engl.: Ladder Diagram – LD) ist eine grafische Programmiersprache zur Beschreibung von Netzwerken mit gleichzeitig arbeitenden booleschen, elektromechanischen Elementen wie Kontakten und Spulen.
Laufzeitprogramm		Zu einer ausführbaren Einheit gebundenes Programm vom *POE*-Typ PROGRAM (durch Zuordnung einer *Task*).
Multielement-Variable	IEC	*Variable* vom Typ *Feld* oder *Struktur*, die aus mehreren *Datentypen* zusammengesetzt ist.
Neustart		siehe *Kaltstart*
Pragma		Pragmas dienen typischerweise der automatischen Vor- oder Nachbearbeitung von Programmen und werden vom *Programmiersystem* angeboten.
PC		Personal Computer. Gleichzeitig in der IEC 61131 (englische Version) verwendete Abkürzung für „Programmable Controller“.
POE	IEC	Abk. für *Programm-Organisationseinheit*
Programm	IEC	Eine *POE* vom Typ PROGRAM
Programm-Organisationseinheit	IEC	Ein *Baustein* der IEC 61131-3 des Typs *Funktion*, *Funktionsbaustein* oder *Programm*, aus dem Anwenderprogramme hierarchisch aufgebaut werden.
Programmiergerät		siehe *SPS-Programmiergerät*
Programmiersystem		siehe *SPS-Programmiersystem*
Querübersetzung		Konvertierung der Darstellung einer *POE* zwischen verschiedenen Programmiersprachen, typischerweise zwischen textuellen und grafischen Sprachen.
Rekursion		ist in der IEC 61131-3 unzulässig. Bezeichnet: a) die *Deklaration* eines *FB*s unter Verwendung des eigenen Namens bzw. *FB-Typs*, b) den gegenseitigen Aufruf von *FB*s. Rekursion wird als Fehler betrachtet und muss bei der Programmierung bzw. zur Laufzeit erkannt werden.
Ressource	IEC	Sprachelement RESOURCE, das einer Zentraleinheit des *SPS-Systems* entspricht.
Rückübersetzbarkeit		Rückgewinnung der Quelle einer *POE* aus der *SPS*.
Schritt	IEC	Zustandsknoten in einem *AS*-Programm, in dem Anweisungen der zu einem *Schritt* zugehörigen *Aktion* angestoßen werden.
Semantik		Bedeutung der Sprachelemente einer Programmiersprache sowie ihre Auslegung und Anwendung.

SPS		Speicherprogrammierbare Steuerung (engl.: „Programmable Controller“).
SPS-Programmiergerät		Einheit aus Computer, *Programmiersystem* und sonstiger Peripherie zur Programmierung der *SPS*.
SPS-Programmiersystem		Gesamtheit aller Programme, die zur Programmierung eines *SPS-Systems* notwendig sind: Erstellung und Übersetzung des Programms, Übertragen in die *SPS* sowie Programmtest- und Inbetriebnahme-Funktionen.
SPS-System		Gesamtheit aller zur Ausführung eines SPS-Programms benötigten Betsandteile der Hardware und Firmware bzw. Betriebssystems.
ST	IEC	Abk. für *Strukturierter Text*
Standard-Funktionen	IEC	Menge der durch die IEC 61131-3 fest vordefinierten *Funktionen* zur Realisierung SPS-typischer Funktionalität.
Standard-Funktionsbausteine	IEC	Menge der durch die IEC 61131-3 fest vordefinierten *Funktionsbausteine* zur Realisierung SPS-typischer Funktionalität.
Std.-FB		Abk. für *Standard-Funktionsbausteine*
Std.-FUN		Abk. für *Standard-Funktionen*
Strukturierter Text	IEC	Strukturierter Text (engl.: Structured Text) ist eine textuelle Programmiersprache zur Beschreibung von Algorithmen und Ausführungssteuerung mit den Mitteln einer modernen Hochsprache.
Symbolische Variable	IEC	*Variable* mit Bezeichner, der eine *Hierarchische Adresse* zugeordnet wird.
Syntax		Aufbau und strukturelles Zusammenwirken der Sprachelemente einer Programmiersprache.
Task	IEC	Definition von Laufzeiteigenschaften eines Programms.
Transition	IEC	Übergang von einem AS-*Schritt* zum nächsten durch Auswertung der Transitionsbedingung.
Typdefinition		Definition eines benutzerspezifischen *Datentyps* auf der Basis bereits vorhandener *Datentypen.*
Überladen von Funktionen	IEC	Eine *Funktion* stellt unter gleichem Namen Funktionsklassen mit unterschiedlichen Eingangs-*Datentypen* (jeweils vom gleichen Typ) zur Verfügung.
Variable		Bezeichnung eines Datenspeichers, der Werte annehmen kann, die durch den *Datentyp* sowie Angaben bei der Variablen-*Deklaration* festgelegt werden.

Warmstart	IEC	Programmstart an der Stelle, an der ein Spannungsausfall stattgefunden hat (engl.: Hot Restart). Dabei bleiben sämtliche gepufferten Datenbereiche des Programms erhalten und das Programm kann weiterlaufen, als hätte es die Unterbrechung nicht gegeben. Im Unterschied zum *Warmstart am Programmanfang* muss die Unterbrechungsdauer prozessabhängig innerhalb eines vorgegebenen Intervalls liegen. Dazu muss das SPS-System eine separat gesicherte Echtzeit-Uhr besitzen, um die Unterbrechungsdauer berechnen zu können.
Warmstart am Programmanfang	IEC	Programmstart wie beim *Warmstart* mit dem Unterschied, dass am Programmanfang wieder begonnen wird, wenn die Unterbrechungszeit eine maximale Zeitspanne überschritten hat. Das Anwenderprogramm kann anhand eines entsprechenden Status-Flags diese Situation erkennen, um eine spezielle Vorbesetzung seiner Daten vornehmen zu können. (engl.: Warm Restart).
Wiederanlauf		Zusammenfassender Begriff für *Warmstart* bzw. *Warmstart am Programmanfang*.
Zuordnungsliste		Liste, die zentral die Zuordnung von Symbolen zu SPS-Adressen beinhaltet.
Zyklus		Ein Durchlauf des (periodisch aufgerufenen) Anwenderprogramms.
Zykluszeit		Die Zeit, die ein Anwenderprogramm für einen *Zyklus* benötigt.

J Literaturverzeichnis

Bücher zur SPS-Programmierung nach IEC 61131-3:

[Adam-97] Adam Hans-Joachim, Adam Mathias
„Programmieren in Anweisungsliste nach IEC 1131-3“
Elektor Verlag,
ISBN 3-89576-056-0

[Aspern-00] Jens von Aspern
„SPS-Softwareentwicklung mit IEC 61131“
Hüthig Verlag 2000,
ISBN 978-3-7785-2681-1

[Becker-00] Norbert Becker
„Automatisierungstechnik“
Vogel Buchverlag 2006,
ISBN 3-8343-3017-5

[Berger-03] Hans Berger
„Automatisieren mit STEP 7 in KOP und FUP“ Speicher-programmierbare Steuerungen SIMATIC S7-300/400
Hrsg.: Siemens-Aktiengesellschaft, Berlin München,
Publicis-MCD-Verlag Erlangen 2003,
ISBN 978-3895782190

[Berger-06] Hans Berger
„Automatisieren mit STEP 7 in AWL und SCL“ Speicher-programmierbare Steuerungen SIMATIC S7-300/400
Hrsg.: Siemens-Aktiengesellschaft, Berlin München,
Publicis-MCD-Verlag Erlangen 2003,
ISBN 978-3895782800

[Lepers-07] Heinrich Lepers
„SPS Programmierung nach IEC 61131-3“, aus Reihe Reihe: Franzis PC und Elektronik
Franzis Verlag Poing, 2007,
ISBN 978-3-7723-5805-0

[Lewis-98] R. W. Lewis
„Programming industrial control systems using IEC 1131-3“, IEE Control Engineering, The Institution of Electrical Engineers, 1998,
ISBN 0-852-96950-3

[Lewis-01] R. W. Lewis
„Modelling Control Systems Using Iec 61499: Applying Function Blocks to Distributed Systems “, IEE Control Engineering, The Institution of Electrical Engineers, 2001,
ISBN 0-852-96796-9

[Neumann-98] Peter Neumann, Eberhard E. Grötsch, Christoph Lubkoll, René Simon
„SPS - Standard: IEC 1131-3, Programmieren in verteilten Automatisierungssystemen“
R. Oldenbourg Verlag, München Wien 1998,
ISBN 3-486-24153-2

[Rolle-98] I. Rolle (Hrsg.), A. Lehmann, H.-P. Otto, L. Trapp, H. Wegmann
„IEC 61131, Wozu?“,
VDE Taschenbuch, 1998,
ISBN 3-800-72309-3

[Seitz-03] Matthias Seitz
„Speicherprogrammierbare Steuerungen“,
Fachbuchverlag Leipzig 2003,
ISBN 3-446-22174-3

[Wratil-96] Peter Wratil
„Moderne Programmiertechnik für Automatisierungssysteme“, EN 61131 (IEC 1131) verstehen und anwenden
Vogel Verlag Würzburg 1996,
ISBN 3-8023-1575-8

Normen zur SPS-Programmierung:

[DIN EN 61131-1] DIN EN 61131-1 Norm , 2004-03 Speicherprogrammierbare Steuerungen - Teil 1: „Allgemeine Informationen“ (IEC 61131-1:2003); Deutsche Fassung EN 61131-1:2003
Beuth Verlag GmbH, www.beuth.de

[DIN EN 61131-2] DIN EN 61131-2; VDE 0411-500:2004-02 Norm , 2004-02 Speicherprogrammierbare Steuerungen - Teil 2: „Betriebsmittelanforderungen und Prüfungen“ (IEC 61131-2:2003); Deutsche Fassung EN 61131-2:2003
Beuth Verlag GmbH, www.beuth.de

In Vorbereitung:
DIN EN 61131-2; VDE 0411-500:2007-01 Norm-Entwurf , 2007-01 Speicherprogrammierbare Steuerungen - Teil 2: „Betriebsmittelanforderungen und Prüfungen“ (IEC 65B/582/CDV:2006); Deutsche Fassung prEN 61131-2:2006
Beuth Verlag GmbH, www.beuth.de

Berichtigung zum Teil 2:
DIN EN 61131-2 Berichtigung 1; VDE 0411-500 Berichtigung 1:2004-10 Norm , 2004-10 „Berichtigungen zu DIN EN 61131-2“ (VDE 0411 Teil 500):2004-02
Beuth Verlag GmbH, www.beuth.de

[DIN EN 61131-3] DIN EN 61131-3 Norm , 2003-12, Ed. 2 Speicherprogrammierbare Steuerungen - Teil 3: „Programmiersprachen (IEC 61131-3:2003)“; Deutsche Fassung EN 61131-3:2003
Beuth Verlag GmbH, www.beuth.de

[IEC 61131-4 TR] DIN EN 61131-4 Technical Report , 2004-07 „User guidelines“
IEC, www.iec.ch

[DIN EN 61131-5] DIN EN 61131-5 Norm , 2001-11 “Speicherprogrammierbare Steuerungen - Teil 5: Kommunikation (IEC 61131-5:2000)“; Deutsche Fassung EN 61131-5:2001
Beuth Verlag GmbH, www.beuth.de

[DIN EN 61131-7] DIN EN 61131-7 Norm , 2001-11
„Speicherprogrammierbare Steuerungen - Teil 7: Fuzzy-Control-Programmierung (IEC 61131-7:2000)“; Deutsche Fassung EN 61131-7:2000
Beuth Verlag GmbH, www.beuth.de

[DIN EN 61131-8 TR] DIN EN 61131-3 Beiblatt 1 Norm , 2005-04
„Speicherprogrammierbare Steuerungen - Leitlinien für die Anwendung und Implementierung von Programmiersprachen für Speicherprogrammierbare Steuerungen“; entspricht DIN EN 61131-8 TR Ed. 2
Beuth Verlag GmbH, www.beuth.de

[IEC 61499-1] DIN EN 61499-1 Norm , 2006-06
„Funktionsbausteine für industrielle Leitsysteme - Teil 1: Architektur (IEC 61499-1:2005);“ Deutsche Fassung EN 61499-1:2005
Beuth Verlag GmbH, www.beuth.de

[IEC 61499-2] DIN EN 61499-2 Norm , 2006-06
„Funktionsbausteine für industrielle Leitsysteme - Teil 2: Anforderungen an Software-Werkzeuge (IEC 61499-2:2005);“ Deutsche Fassung EN 61499-2:2005
Beuth Verlag GmbH, www.beuth.de

[IEC 61499-4] DIN EN 61499-4 Norm , 2007-05
„Funktionsbausteine - Teil 4: Regeln für normgerechte Profile (IEC 61499-4:2005)“; Deutsche Fassung EN 61499-4:2006
Beuth Verlag GmbH, www.beuth.de

[IEC 61499-08] Normungskomitee der IEC: „TC65: INDUSTRIAL PROCESS MEASUREMENT AND CONTROL SC65B – Devices & process analysis”
http://www.holobloc.com/stds/iec/tc65wg6/minutes/fla07_wg15_minutes.pdf

[IEC IEC 65B WG7] Normungskomitee der IEC: „WG 7: Programmable control systems for discontinuous industrial-processes”,
http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=e&wwwprog=dirwg.p&progdb=db1&ctnum=436

[OPC Foundation] Nutzervereinigung OPC: Openness, Productivity, Collaboration (vormals: OLE for Process Control) http://www.opcfoundation.org

[VDI 3696/2-95] VDI/VDE-Richtlinien, VDI/VDE 3696 Blatt 2 „Herstellerneutrale Konfigurierung von Prozeßleitsystemen- Standard-Funktionsbausteine“ 10/95. VDI/VDE-Handbuch Regelungstechnik.

[VDI 3696/3-95] VDI/VDE-Richtlinien, VDI/VDE 3696 Blatt 3, „Herstellerneutrale Konfigurierung von Prozeßleitsystemen- Syntax des Funktionsbaustein-Textes“ 10/95. VDI/VDE-Handbuch Regelungstechnik.

PLCopen Kontakte

[PLCopen Europe] Eelco van der Wal
Molenstraat 34
4201 CX Gorinchem, The Netherlands

Tel: +31-183-660261
Fax: +31-183-664821
Email: evdwal@plcopen.org
http://ww.plcopen.org

[PLCopen North America]
Bill Lydon
10308 W. Cascade Dr.
Franklin, WI 53132
USA

Tel: +1 414-427-5853
Email: blydon@plcopen-na.org
http://www.plcopen-na.org/

[PLCopen Japan] Shigeo Kawashima
Mitsui Sumitomo Bank Ningyo-cho
5-7, Nihonbashi Ohdemma-cho, Chuo-lu
Tokyo, 103-0011
Japan

Tel: +81-5847-8058
Fax: +81-5847-8180
Email: kawashima-shigeo@fujielectric.co.jp
http://www.plcopen-japan.jp

[PLCopen China] No. 1, JiaoChangKuo
DeShengMenWai
Beijing, 100011
China

Tel: +86-(010) 6202-9216
Fax: +86-(010) 6202-7873
http://www.plcopen.org.cn

K Index

—A—

Ablaufkette 176
Ablaufsprache siehe AS
Ablaufsteuerung 206
Ableitung von Datentypen 82
ACCESS 251
AE 105, 381
Akkumulator siehe AE
Aktion 189, 191, 381
– Anweisung 191
– boolesch 189, 191
Aktionsblock 189, 191, 381
Aktionskontrollblock 381
Aktionskontrolle 199
Aktionsmerker 200
Aktionsname 193
Aktiv-Attribut 175
Aktivierungsmerker 203
Aktualparameter 36, 60, 217, 381
– AWL 112
– FBS 144
– KOP 159
– ST 126
AktuellesErgebnis siehe AE
Alternativ-Kettenauswahl 177
Anfangsschritt 183, 207
Anfangswert 381
– bei Programmstart 96
– für Anwenderdaten 298
Anfangswerte 89
Anlagen-Dokumentation 270
Anlaufverhalten 278
Anweisung 103
– AWL 103, 104
– ST 119, 120
Anweisungsliste siehe AWL
Anweisungsteil 194
Anwendungs-Modell 308
ANY
– Allgemeiner Datentyp 90
– Datentyp 91
Arithmetische Funktionen 322
AS 174, 382
– Netzwerk 175
– Struktur 174, 190
Attribut Bestimmungszeichen 98
Attribute 98
Aufrufparameter 257
Aufzählung 382
Aufzählungstyp 85
Ausdruck (ST) 103, 121
– Abarbeitung 123
Ausführungssteuerung
– FBS 143
– KOP 158
Ausgangsparameter
– PROG 251
Austauschformat 18
Auswahl-Funktionen 325, 327
AWL 104, 382
– Aufruf von FB 114
– Aufruf von Funktionen 112
AWL-Beispiele 343
AWL-Syntax 359

—B—

Batteriepufferung 382
Baustein 22, 382
– bei IEC 61499 307
– Bibliothek 289
Bausteinarten 30
Bausteine in verteilten Systemen 288
Begrenzungszeichen 70, 373
Beispiele
– AS (Dino-Park) 208

– AWL (Bergbahn) 116
– FBS (Stereo-Rekorder) 148
– KOP (Bergbahn) 163
– ST (Stereo-Rekorder) 134
Belegungsliste 245, 270
Bereichsangabe 382
Bergbahn
– Beispiel in AWL 116
– Beispiel in KOP 163
Bestimmungszeichen
– Attribut 98
– in AS 189, 193, 198
Bezeichner 70, 74
Bistabile Elemente (Flip-Flops) 336
Bitschiebe-Funktionen 323
Bitweise Boolesche Funktionen 324

—C—

CASE 129
CD
– Inhalt 315
CONFIGURATION 53, 246
CONSTANT 98, 99, 100
CPU 382

—D—

Datenbaustein 30, 285, 382
Datenfluss 305, 309
Datenstruktur 87, 96
Datentyp 79, 81, 382
– Ableitung 82, 88, 381
– Allgemein 90, 381
– Anfangswert 84
– ANY 90
– Aufzählung 84
– Bereichsangabe 84
– Datenstruktur 84
– Elementar 81, 383
– Feldgrenzen 84
– zusätzliche Eigenschaften 83
Deklaration 382
Deklarationsblock 382
Diagnose 292
DIN EN 61131-3 14
Dino-Park
– Beispiel in AS 208
Direkt dargestellte Variable 93, 94, 257, 382

—E—

E/A-Peripherie 77, 93, 382
Einfache Kette 177
Eingangsparameter
– PROG 251
Einkaufsberater 317
Einzelelement-Variable 95, 382
Einzelschritt 276, 284
Elementare Datentypen 81
EN/ENO 54
– FBS 144
– KOP 160
– Ladder 170
Erweiterung von Funktionen 383
Execution Control Chart (ECC) 310
EXIT 133
Externe Variable 257

—F—

F_EDGE 98, 100
FB siehe Funktionsbaustein
FB-Aufruf
– AWL 114
– FBS 144
– indirekt 383
– KOP 159
– ST 122, 126
FB-Instanz 22, 64, 383
FBS 137, 383
– Aktionsblock 198
– Aufruf von FB 144
– Aufruf von Funktionen 144
FB-Typ 43, 383
FB-Verschaltung 288, 291
– bei IEC 61499 303, 313
Fehlerursachen 353
Feld 86, 383
Feldgrenzen 86
Flanke 383
Flankenerkennung 49, 337
– Kontakt 156
– Spule 157
FOR 131
Forcing 276, 283, 284
Formalparameter 36, 60, 383
– FBS 141, 144
– KOP 159
– ST 124, 126
FUN siehe Funktion
Funktion 31, 50, 383

– als Operator in ST 124
– FB-Aufrufe 57
– FUN-Aufrufe 57
– für Datentypen der Aufzählung 334
– für Datentypen der Zeit 333
– für Zeichenfolgen 330
– Variablenarten 50
– zur Typwandlung 320
Funktion (AWL)
– Parameter 113
Funktionsaufruf
– AWL 112
– FBS 144
– KOP 160
– ST 123, 124
Funktionsbaustein 31, 42
– Ableitung 292
– Basis-FB bei IEC 61499 312
– bei IEC 61499 305
– bei IEC 61499 310
– benutzerdefiniert bei IEC 61499 313
– indirekter Aufruf 66
– Instanzbildung 22
– Instanziierung 42
– Instanzname 92
– Kapselung 48
– Objektorientierung 48
– Seiteneffekte 48
– Variablenarten 49
– Wiederverwendbarkeit 48
– zusammengesetzt bei IEC 61499 312
Funktionsbaustein-Modell 309
Funktionsbausteinsprache siehe FBS
Funktionsplan 291
Funktionswert 50
– AWL 113
– FBS 145
– KOP 160
– ST 123, 125, 126
Fuzzy Control Language 16

—G—

Geräte
– bei IEC 61499 305
Geräte-Modell 306
Globale Variable 257
Grafikelement 138
– FBS 142
– KOP 153
Grafikobjekt
– FBS 142
– KOP 153
– KOP und FBS 138

—H—

Haltepunkt 276, 284
Hierarchische Adresse 93, 383

—I—

IEC 19
IEC 1131-3
– Gemeinsame Sprachelemente 69
– Kommunikationsmodell 256
– Softwaremodell 241
– Stärken 297
– Strukturierte SPS-Programme 300
– Variablenkonzept 80
IEC 61131
– Aufbau 14
– Geschichte 14
– Ziele 12
IEC 61131-3
– Anlaufverhalten 278
– Fehlerkonzept 293
– Grafische Sprachen 103
– Textuelle Sprachen 103
IEC 61499 303
– Überblick Aufbau 314
IF 127
Implementierungsabh. Parameter 355
Indirekte Adresse
– Ladder 172
Initialisierung
– Anfangswerte von Variablen 89
– bei fehlenden Eingangsvariablen 63
– bei Variablenarten 98
– FB-Instanz 63
Instanz 42, 383
– Datenbaustein 47
– Gedächtnis 46, 47
– RETAIN 47
– Struktur 44
Instanznamen 92

—K—

Kaltstart 97, 383
Kette 177
Kettenauswahl 178

Ketten-Schleife 179
Kettensprung 179
Klammeroperator (AWL) 108
Kommentar 272, 383
– AS 175
– AWL 105
– KOP/FBS 138
– ST 119
Kommunikation 255
Kommunikations-Bausteine 257
Kommunikationsmanager 277
Kompaktgeräte 9
Konfiguration 245, 383
– Beispiel 253
– Dokumentation 270
– Kommunikation 246
Konfigurations-Editor 303, 314
Konfigurationselemente 242
Konnektor 103
– AS 184
– KOP/FBS 139
Konnenktor
– FBS 142
Kontakt 154
Kontaktplan siehe KOP
Kontrollfluss 305, 309
KOP 137, 152, 383
– Aktionsblock 197
– Aufruf von FB 159
– Aufruf von Funktionen 160

—L—

Ladder 169
Laufzeiteigenschaften 248
– bei IEC 61499 305
Laufzeitprogramm 242, 243, 245, 384
Literal 70, 72

—M—

Makro 291
Modifizierer (AWL) 108
Multiauswahl (ST) 128
Multielement-Variable 50, 95, 384

—N—

Netzwerk 137
– Abarbeitung in FBS 146
– Abarbeitung in KOP 146, 161
– Ladder 170
Netzwerkaufbau
– FBS 140
– KOP 153
Netzwerkgrafik 138
Netzwerkkommentar 138
Netzwerkmarke 137
Neustart 97, 384
NON_RETAIN 98, 100
NON_RETAIN 99
Numerische Funktionen 321

—O—

Öffner 155
Online-Änderung 277
Operand
– AWL 103, 105
– ST 121
Operand (ST) 121
Operator (AWL) 103, 105, 108
– ANY_BIT 110
– ANY_NUM 111
– FB-Eingangsvariable 114
– JMP/CAL/RET 112
– Sprung/Aufruf 112
Operator (ST) 121, 122
– Funktion 124
Operatorgruppen 107
Organisationsbaustein 30

—P—

PC 384
PLCopen 17
– Benchmark 17
– Safety 17
– Sicherheit 17
– XML 17
– Zertifizierung 18
POE 21, 30, 384
– Anweisungsteil 23, 40
– Ausgangsparameter 47
– AWL-Beispiele 343
– AWL-Einführungsbeispiel 25
– Deklarationsteil 34, 77
– Eingangsparameter 47
– Formalparameter 37
– Grundelemente 32
– Merkmale der Schnittstelle 36
– Merkmalsübersicht 68
– Rekursiver Aufruf 58
– Sprachunabhängigkeit 269

– Übersicht 21, 30
– Variablenarten 35
– Variablen-Zugriff 38
– Wiederverwendbarkeit 32, 298
POE-Typen 30
Power Flow 280
Pragmas 76
PROG siehe Programm
PROGRAM 53, 248
Programm 31, 53, 384
Programmdokumentation 270
Programmierfehler 298
Programmiergerät 384
Programmiersprachen
– Merkmale 41
Programmiersystem 384
– Anforderungen 259
Programmorganisationseinheit siehe POE
Programmstatus 279, 280
Programmstruktur
– AS 174
– Dokumentation 270
Programmtest 284
– in Ablaufsprache 284
Programmtransfer 277
Projektverwaltung 272
– Aufgaben 275

—Q—

Q-Ausgang 200
Querübersetzbarkeit 147, 263, 384
– Einschränkungen 265
– Gütekriterien 268
– mit Zusatzinformation 267
Querverweisliste 270, 271

—R—

R_EDGE 98, 100
READ_ONLY 98, 100
READ_WRITE 98, 100
Rekursion 384
REPEAT 130
Reservierte Schlüsselworte 71, 373
RESOURCE 247
Ressource 245, 384
– bei IEC 61499 305, 306
Ressource-Modell 307
RETAIN 23, 49, 98, 99, 100
RETURN 126
Rezeptur 284
Rückdokumentation 260
Rückführung (AS) 179
Rückkopplungsvariable
– AS 191, 193
– FBS 148
– KOP 162
Rückübersetzbarkeit 260, 261, 384

—S—

Safety 16
Schließer 155
Schlüsselworte 70
Schritt 175, 182, 384
Schrittbaustein 30, 174
Schrittkette 174
Schrittmerker 183, 193
Schrittname 183
Schrittzeit 183
Seiteneffekte
– bei FB 48
– bei FUN 50
– in KOP 163
Semantik 69, 384
Service-Schnittstellen-Funktionsbausteine 307
Sicherheitsgerichtete SPS 16
Simultan-Verzweigung 178
Soft Logic Array 9
Speicher
– Ladder 171
Sprachelemente
– einfache 69
Sprachverträglichkeit 262
Sprungmarke 105
SPS 385
SPS-Adressen 94
SPS-Konfiguration 241, 299
SPS-Programmiergerät 385
SPS-Programmiersystem 259, 385
– Trend 301
SPS-Programmierwerkzeuge 10
– Anforderungen 259
SPS-System 12, 14, 385
Spule 154
ST 119, 385
– Aufruf von FB 122, 126
– Aufruf von Funktionen 123
Standard-Datentypen 351
Standard-FB

– bei IEC 61499 313
Standard-Funktionen 214, 319, 385
– Aufrufschnittstelle 221
– Beispiele 221
– Beschreibung im Anhang 319
– Erweitern 220
– Überladen 218
Standard-Funktionsbausteine 230, 385
– Aufrufschnittstelle 232
– Beispiele 232
– Beschreibung im Anhang 335
Status
– asynchron 281
– bei Änderung 281
– Datenanalyse 281
– synchron 281
Stereo-Rekorder
– Beispiel in FBS 148
– Beispiel in ST 134
Stromflußanzeige 280
Stromlaufplan 41
Strukturierter Text siehe ST
Strukturkomponente 95
Symbolische Variable 93, 94, 385
Syntax 69, 385
– AWL 359
Syntaxgraph für AWL 359
Systemkonfigurator 277
System-Modell 305

—T—

Task 242, 245, 385
– Unterbrechbarkeit 250
TASK 248
– Parameter 250
Teilanweisung 121
Test&Inbetriebnahme 276
Token siehe Aktiv-Attribut
Transition 175, 182, 184, 385
Transitionsbedingung 175, 182
– direkte Angabe 184
– Konnektor 184
Transitionsname 184
Typdefinition 77, 82, 385
– Anfangswerte 89
Typgerechter Datenzugriff 298
Typüberprüfung 79
Typverträglichkeit 344
– in AWL 106
– in ST 125

—U—

Überladen 60, 218, 232
Überladen von Funktionen 91, 385
Umverdrahtung 245, 271
Unerreichbare Netzwerke 180
Unsichere Netzwerke 180

—V—

VAR 98, 100
VAR_ACCESS 98, 100
VAR_CONFIG 98, 254
VAR_EXTERNAL 46, 49, 65, 98, 100
VAR_GLOBAL 98, 100
VAR_IN_OUT 37, 46, 49, 65, 98, 100
VAR_INPUT 37, 46, 65, 98, 100
VAR_OUTPUT 37, 46, 65, 98, 100
VAR_TEMP 46, 98
Variable 92, 385
– Attribute 98
– Deklaration 22
– statt Hardware-Adressen 77, 297
Variablenarten 39, 100
Variablenausgabe 280
Variablendeklaration 92
– Attribute 98
– Beispiel 23
– grafische Darstellung 100
– Variablenarten 98, 100
Variablenstatus 279, 280
Verbindung
– FBS 142
Verbindungen
– FBS 143
– KOP 154
Verdrahtungsliste 270
Vergleichs-Funktionen 328
Verteilte Anwendungen 289
Verzweigung (ST) 127

—W—

Warmstart 97, 386
Wertebereich 85
WHILE 130
Wiederanlauf 97, 386
Wiederverwendbarkeit 29, 32, 48

—Z—

Zähler 338
Zeitgeber (Zeiten) 340
Zugriffspfad 48, 247, 252, 257

Zuordnungsliste 95, 270, 271, 386
Zustandsdiagramm
– bei IEC 61499 310
Zuweisung (ST) 124
Zwischensprache 104
Zyklus 386
– AS-Ablaufsteuerung 207
Zykluszeit 386

Autorenbiographien

Karl-Heinz John

Geboren am 09.07.1955 in Ulm. Verheiratet, 2 Kinder.

Studium der Informatik 1975 bis 1981 an der Friedrich-Alexander-Universität Erlangen-Nürnberg mit Schwerpunkt Rechnerarchitektur; Abschluss Diplom-Informatiker. Diplomarbeit über Mikroprogrammierung. 1981 bis 1983 wissenschaftlicher Mitarbeiter am Lehrstuhl für Rechnerarchitektur und Verkehrstheorie.

Seit 1984 bei der infoteam Software GmbH (www.infoteam.de), Softwarespezialist für Automatisierung und medizinische Gerätetechnik mit Niederlassungen in Bubenreuth / Nürnberg, Stäfa / Schweiz und Beijing / China. Als Mitinhaber und Geschäftsführer Geschäftsbereich „Technik“ u.a. für die Entwicklung von IEC 61131- Programmiersystemen wie OpenPCS verantwortlich.
Mitbegründer der PLCopen (www.plcopen.org), Vice-President ASQF (www.asqf.de) (größter deutscher Fachverband Softwarequalität); ASQF-Fachgruppenleiter Automatisierung (ASQF Automation Days in Nürnberg und St. Gallen).

Michael Tiegelkamp

Geboren am 23.12.1959 in Krefeld. Geschieden, 2 Kinder.

Studium der Informatik 1982 bis 1988 an der Friedrich-Alexander-Universität Erlangen-Nürnberg mit Schwerpunkten Rechnerarchitektur und Kommunikationssysteme; Abschluss als Diplom-Informatiker. Diplomarbeit über SPS-Architekturen.

1988 bis 1994 bei der infoteam Software GmbH als Projektleiter und Leiter Marketing für SPS-Programmiersysteme verantwortlich. Bei der SIEMENS AG bis 1997 als Projektleiter verantwortlich für Software-Entwicklungen im Bereich

der Programmiergeräte-Software für SIMATIC S5/S7, zwischen 1998 und 2002 Gruppenleiter für Produktdefinition von SIMATIC Industrie Software und von 2002 bis 2004 Projektleiter für System-Roadmap & Innovation in der Automatisierungstechnik. 2004 und 2005 Gruppenleiter Produktmanagement in der Niederspannungs-Energieverteilung, 2005 bis 2007 Aufbau des Geschäftssegments Power Management. 2007 Integrationsmanager und Wechsel zur Siemens-Tochter OEZ s.r.o., Tschechien, seit 2007 dort Manager Prozesse und Projekte.

Dieses Buch wurde mit viel Engagement und großer Sorgfalt geschrieben. Sollten Sie dennoch Fehler entdecken, wären wir dankbar für Ihre Hinweise! An dieser Stelle einen herzlichen Dank an alle Leser, welche nach wie vor durch Hinweise und Anmerkungen zur heutigen Qualität dieses Buches beigetragen haben.

Auch weitere Anregungen jeder Art sind willkommen.

Die Autoren sind jederzeit erreichbar unter:

	Karl-Heinz John	*Michael Tiegelkamp*
Anschrift	Irrlrinnig 13 91301 Forchheim	Kurpfalzstr. 34 90602 Pyrbaum
e-mail	karlheinz.john@gmx.de *oder (über infoteam):* john@infoteam.de	Michael.Tiegelkamp@gmx.de *oder (über Siemens):* michael.tiegelkamp@siemens.com
Homepage	www.fen-net.de/karlheinz.john	
Tel./Fax:	+49 (9191) 14672 (Tel.)	+49 (173) 3112532 (Tel.)