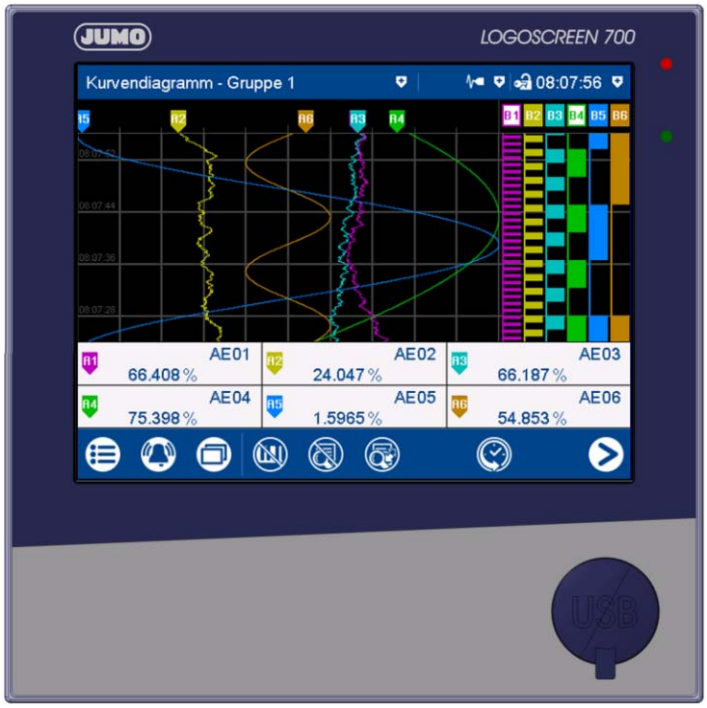
JUMO LOGOSCREEN 601/700

Bildschirmschreiber



ST-Editor-Anleitung

70653000T96Z000K000

V1.00/DE/00710349

Inhalt

Inhalt

1.1 Sicherheitshinweise

Hinweisende Zeichen

HINWEIS!

Dieses Zeichen weist auf eine **wichtige Information** über das Produkt oder dessen Handhabung oder Zusatznutzen hin.

VERWEIS!

Dieses Zeichen weist auf **weitere Informationen** in anderen Abschnitten, Kapiteln oder anderen Anleitungen hin.

1.2 Bestimmungsgemäße Verwendung

Das Gerät ist für die Verwendung in industrieller Umgebung bestimmt, wie in den technischen Daten spezifiziert. Eine andere oder darüber hinausgehende Nutzung gilt als nicht bestimmungsgemäß.

Das Gerät ist entsprechend den gültigen Normen und Richtlinien sowie den geltenden sicherheitstechnischen Regeln gebaut. Dennoch können bei unsachgemäßer Verwendung Personen- oder Sachschaden entstehen.

Um Gefahren zu vermeiden, darf das Gerät nur benutzt werden:

- für die bestimmungsgemäße Verwendung
- in sicherheitstechnisch einwandfreiem Zustand
- unter Beachtung der mitgelieferten Technischen Dokumentation

Auch wenn das Gerät sachgerecht oder bestimmungsgemäß eingesetzt wird, können von ihm applikationsbedingte Gefahren ausgehen, z. B. durch fehlende Sicherheitseinrichtungen oder falsche Einstellungen.

1.3 Qualifikation des Personals

Dieses Dokument enthält die erforderlichen Informationen für den bestimmungsgemäßen Gebrauch des darin beschriebenen Gerätes.

Es wendet sich an technisch qualifiziertes Personal, das speziell ausgebildet ist und einschlägiges Wissen auf dem Gebiet der Automatisierungstechnik besitzt.

Die Kenntnis und das technisch einwandfreie Umsetzen der in der mitgelieferten Technischen Dokumentation enthaltenen Sicherheitshinweise und Warnungen sind Voraussetzungen für die gefahrlose Montage, Installation und Inbetriebnahme sowie für die Sicherheit während des Betriebes des beschriebenen Gerätes. Nur qualifiziertes Personal verfügt über das erforderliche Fachwissen, um die in diesem Dokument verwendeten Sicherheitshinweise und Warnungen im konkreten Einzelfall richtig zu interpretieren und in die Tat umzusetzen.

1.4 Inhalt dieses Dokuments

HINWEIS!

Dieses Dokument gilt für Bildschirmschreiber der Typen 706521 und 706530.

Dieses Dokument beschreibt die Anwendung des ST-Editors, mit dem der Anwender eine eigene Applikation in der SPS-Programmiersprache „Strukturierter Text“ (ST) für das Gerät erstellen kann. Das Dokument richtet sich an Anwender, die einschlägige Programmierkenntnisse besitzen.

Die SPS-Programmiersprache „Strukturierter Text“ (ST) wird in der Norm DIN IEC 61131-3 beschrieben. Ausführliche Informationen zur Programmierung sind dieser Norm zu entnehmen. Das ST-Modul des betreffenden Gerätes unterstützt nur eine Teilmenge der in der Norm beschriebenen Programmiersprache.

1 Einleitung

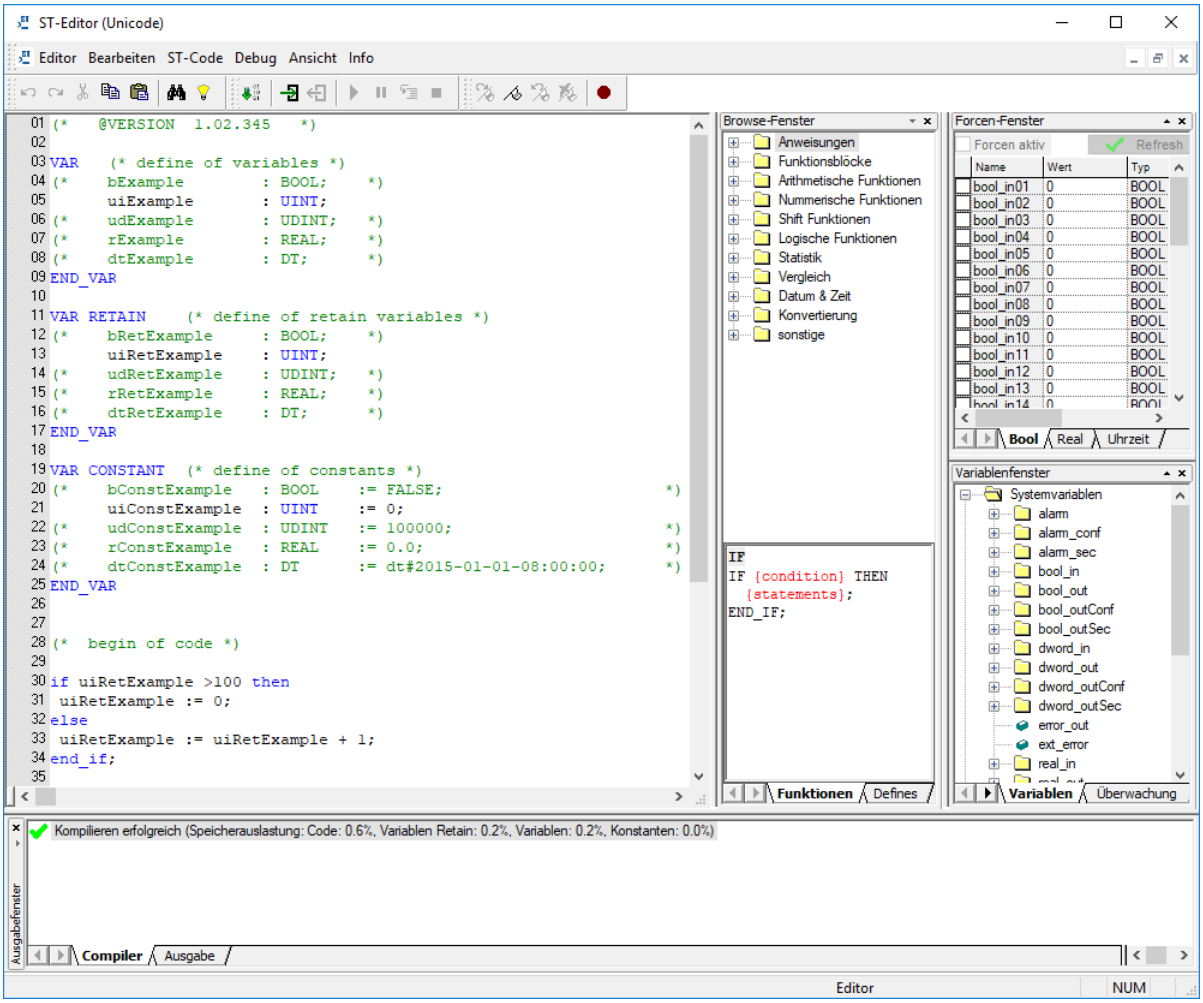
Ergänzend zu diesem Dokument ist die Betriebsanleitung des betreffenden Gerätes zu beachten:

- Typ 706521:
  Dokument 70652100T90Z...K...
- Typ 706530:
  Dokument 70653000T90Z...K...

Der ST-Editor ist Bestandteil des Setup-Programms und wird von diesem aus durch Anklicken der entsprechenden Schaltfläche im Fenster „ST-Code“ gestartet (siehe Betriebsanleitung).

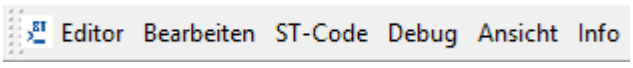
Die fertige Applikation wird als ST-Code zum Gerät übertragen und im integrierten ST-Modul ständig abgearbeitet.

Übersicht



Die Programmoberfläche besteht aus mehreren Leisten und Fenstern, die in den folgenden Abschnitten kurz beschrieben werden.

Menüleiste



Die einzelnen Menüs enthalten Funktionen zum Bearbeiten und Übersetzen eines Programms, zur Fehlersuche, zum Aus- und Einblenden von Leisten und Fenstern der Programmoberfläche sowie Versionsinformationen zum ST-Editor.

Bei aktiviertem Anzeigeschutz (im Menü „Editor“) wird vor dem erneuten Starten des ST-Editors das vom Anwender vergebene Passwort abgefragt. Ist das Passwort nicht bekannt, besteht die Möglichkeit, den ST-Editor mit den werkseitigen Einstellungen (Code-Beispiel) zu starten. Zuvor erstellter eigener Quellcode wird dadurch aber gelöscht!

In der Statusleiste werden weitere Informationen zu den Funktionen der Menüleiste eingeblendet, wenn der Mauszeiger auf die einzelne Funktion im jeweiligen Menü zeigt.

2 Bedienung

Symbolleiste

Einige Funktionen der Menüleiste stehen auch in der Symbolleiste zur Verfügung und können hier durch einfachen Mausklick ausgewählt werden. Die Bedeutung der Symbole wird durch eine Tooltip-Funktion stichwortartig beschrieben (mit dem Mauszeiger auf das jeweilige Symbol zeigen). Zusätzlich werden dann in der Statusleiste weitere Informationen zu der betreffenden Funktion eingeblendet.

Symbol	Bedeutung	Beschreibung
	Rückgängig (Strg+Z)	Letzte Aktion rückgängig machen
	Wiederherstellen	Zuvor rückgängig gemachte Aktion wiederherstellen
	Ausschneiden (Strg+X)	Markierte Daten entfernen und in Zwischenablage übertragen
	Kopieren (Strg+C)	Markierte Daten kopieren und in Zwischenablage übertragen
	Einfügen (Strg+V)	Daten aus der Zwischenablage einfügen
	Suchen (Strg+F)	Eingegebenen Text suchen
	Funktionstext (F4)	Funktionstext aus dem Browse-Fenster einfügen Siehe auch Abschnitt „Browse-Fenster“, Seite 11.
	Übersetzen (F7)	ST-Code übersetzen
	Debugging starten	Debugging starten (Kaltstart) Nach dem Starten wird ein bereits im Gerät vorhandenes Programm beendet. Stattdessen wird das aktuelle Programm in das Gerät geladen. Siehe auch Abschnitt „Debugging“, Seite 13.
	Debugging beenden	Debugging beenden Nach dem Beenden bleibt das aktuelle Programm im Gerät erhalten und wird ausgeführt. Siehe auch Abschnitt „ST-Editor beenden“, Seite 13.
	Starten/Weiter (F5)	Programm starten oder nach Unterbrechung weiter ausführen
	Programm unterbrechen	Programm unterbrechen
	Einzelschritt (F11)	Programm per Einzelschritt ausführen
	Beenden	Programm beenden
	Nächstes Lesezeichen (F2)	Einfügemarke zum nächsten Lesezeichen verschieben
	Lesezeichen umschalten (Strg+F2)	Lesezeichen für die aktuelle Zeile umschalten
	Vorheriges Lesezeichen (Shift+F2)	Einfügemarke zum vorherigen Lesezeichen verschieben
	Lesezeichen löschen (Shift+Strg+F2)	Alle Lesezeichen löschen

Symbol	Bedeutung	Beschreibung
	Haltepunkt umschalten (F9)	Haltepunkt für die aktuelle Zeile umschalten

Bearbeitungsfenster

```
01 (*   @VERSION  1.02.345   *)
02
03 VAR   (* define of variables *)
04 (*    bExample        : BOOL;    *)
05       uiExample       : UINT;
06 (*    udExample       : UDINT;   *)
07 (*    rExample        : REAL;    *)
08 (*    dtExample       : DT;      *)
09 END_VAR
10
11
12 VAR CONSTANT  (* define of constants *)
13 (*    bConstExample   : BOOL    := FALSE;                    *)
14       uiConstExample  : UINT    := 0;
15 (*    udConstExample  : UDINT   := 100000;                   *)
16 (*    rConstExample   : REAL    := 0.0;                      *)
17 (*    dtConstExample  : DT      := dt#2015-01-01-08:00:00;   *)
18 END_VAR
19
20
21 (*  begin of code *)
22
23 if uiExample >100 then
24  uiExample := 0;
25 else
26  uiExample := uiExample + 1;
27 end_if;
28
```

Im Bearbeitungsfenster wird das Programm erstellt, indem Variablen und Konstanten deklariert werden, Programmcode erstellt wird und Kommentartexte verwendet werden.

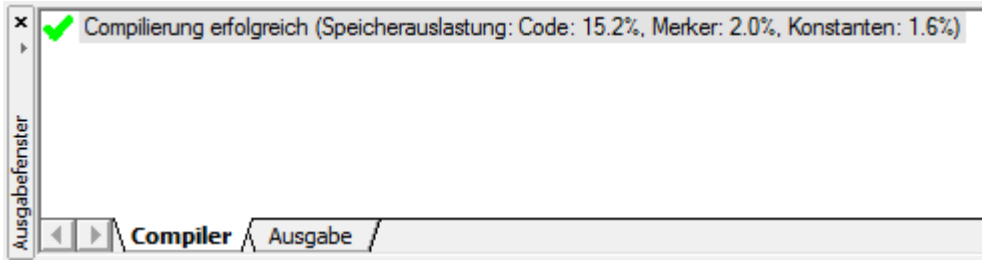
Durch Strg+Mausrad oder Strg+Shift+Pfeiltaste lässt sich die Schriftgröße ändern.

Versionsbezeichnung:

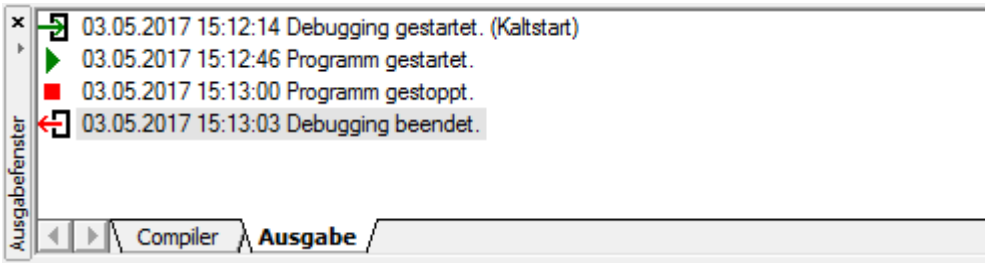
Um das Programm mit einer Versionsbezeichnung zu versehen, muss in einer Kommentarzeile das Schlüsselwort @VERSION gefolgt von der Versionsbezeichnung verwendet werden. Enthält das Programm mehrere Kommentarzeilen mit diesem Schlüsselwort, wird nur die erste Kommentarzeile ausgewertet. Nachdem der ST-Code dauerhaft zum Gerät übertragen wurde, lässt sich die Versionsbezeichnung im Gerät anzeigen (geräteabhängig, z. B. Geräteinfo > Versionen > ST-Code-Version).

Ausgabefenster

Das Ausgabefenster besteht aus zwei Teilen (Registern).

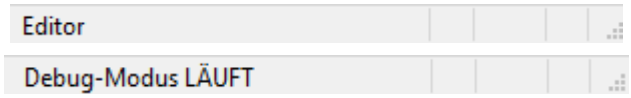


Im Register „Compiler“ werden Meldungen angezeigt, die das Übersetzen des Programmcodes betreffen.



Das Register „Ausgabe“ enthält Meldungen, die den Debug-Modus betreffen.

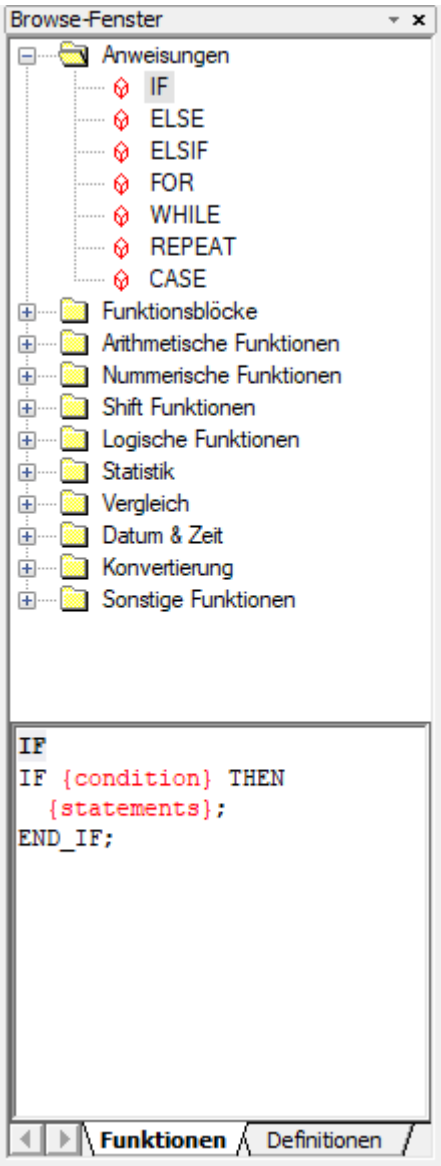
Statusleiste



Die Statusleiste befindet sich am unteren Rand der Programmoberfläche und zeigt an, ob der Editor oder der Debug-Modus aktiv ist. Im Falle des Debug-Modus‘ werden die einzelnen Zustände (STOPP, LÄUFT, PAUSE) angezeigt.

Darüber hinaus werden in der Statusleiste weitere Informationen zu den Funktionen der Menüleiste und der Symbolleiste eingeblendet (mit dem Mauszeiger auf die einzelne Funktion im jeweiligen Menü bzw. auf das Symbol in der Symbolleiste zeigen).

Browse-Fenster



Im Register „Funktionen“ sind alle Anweisungen und Funktionen aufgelistet, die im Editor genutzt werden können. Im unteren Teil des Fensters wird die Verwendung der Anweisung bzw. Funktion gezeigt.

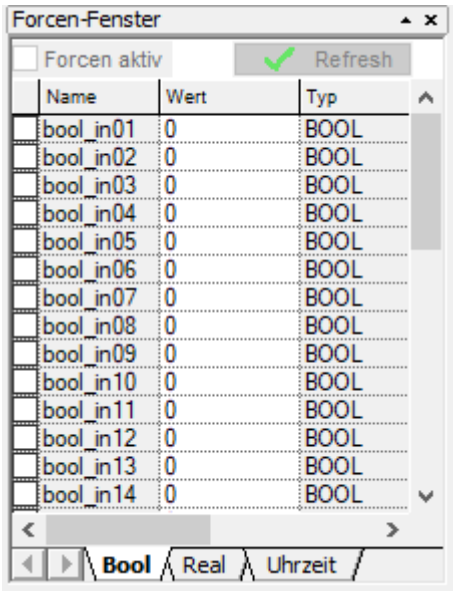
Der im unteren Teil des Fensters dargestellte Funktionstext kann per Drag & Drop, durch Klicken auf das Symbol „Funktionstext“ oder mit Funktionstaste F4 im Bearbeitungsfenster an der gewünschten Stelle in den ST-Code eingefügt werden (am Cursor oder anstelle eines markierten Textes). Nach den Einfügen wird durch wiederholtes Klicken auf das Symbol „Funktionstext“ bzw. Drücken von F4 der nächste Platzhalter im Funktionstext markiert.

HINWEIS!

Das Register „Definitionen“ ist geräteabhängig und enthält Definitionen (Namen für feste Werte), die werkseitig vorgegeben sind oder vom Anwender erstellt wurden. Sofern das ST-Modul des betreffenden Gerätes Definitionen unterstützt, werden diese in einem separaten Kapitel „Definitionen“ beschrieben.

Forcen-Fenster

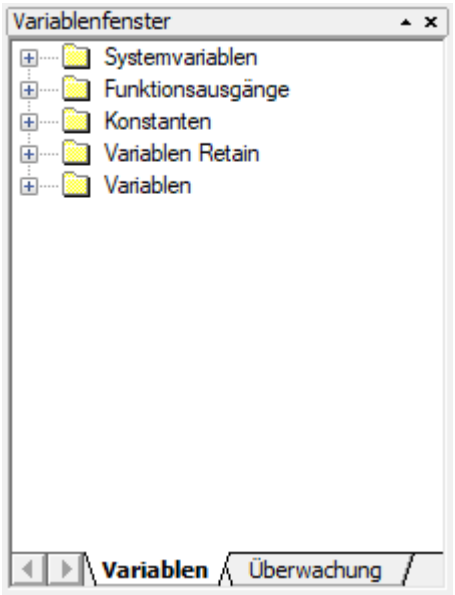
Das Forcen-Fenster besteht aus drei Teilen (Registern). Hier ist nur das Register „Bool“ dargestellt.



Im Forcen-Fenster können die Eingangsvariablen vom Typ „Bool“ und „Real“ sowie Datum und Uhrzeit vom Anwender gesetzt werden, um den Programmcode zu testen.

Variablenfenster

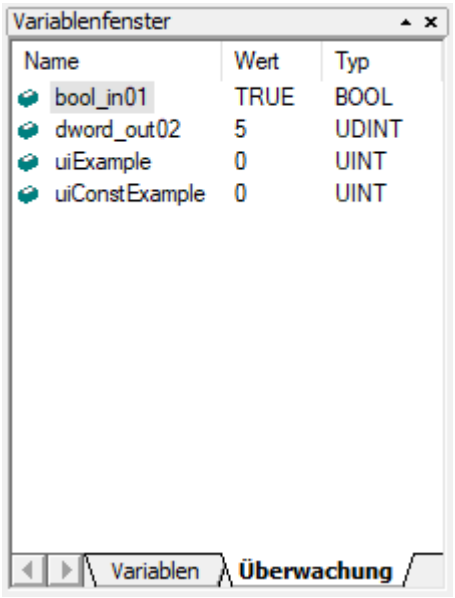
Das Variablenfenster besteht aus zwei Teilen (Registern).



Im Register „Variablen“ werden alle Systemvariablen und Funktionsausgänge (Ausgänge der Funktionsblöcke) sowie die vom Anwender deklarierten Konstanten und Variablen (Merker) aufgelistet. Diese können per „Drag and Drop“ aus dem Variablenfenster kopiert und im Bearbeitungsfenster an der gewünschten Stelle in den ST-Code eingefügt werden.

HINWEIS!

Variablen, die über Netz-Aus gespeichert werden (Retain-Variablen), werden nicht von allen Gerätetypen unterstützt.



Im Register „Überwachung“ können Variablen und Konstanten dargestellt und während des Debuggens beobachtet werden. Dazu müssen diese per „Drag and Drop“ aus dem Bearbeitungsfenster oder dem Register „Variablen“ kopiert und im Register „Überwachung“ eingefügt werden.

Das Einfügen und auch das Entfernen ist nur möglich, wenn das Programm nicht läuft. Dann stehen auch über das Kontextmenü (rechte Maustaste) verschiedene Funktionen zur Verfügung. Mit der Funktion „Neu...“ kann eine neue Variable hinzugefügt werden, indem ein Variablenname eingegeben wird (gegebenenfalls Variablendeklaration erforderlich).

Der Anwender kann für Systemvariablen Alias-Namen vergeben.

⇨ Kapitel 4.4 „Alias-Namen“, Seite 23

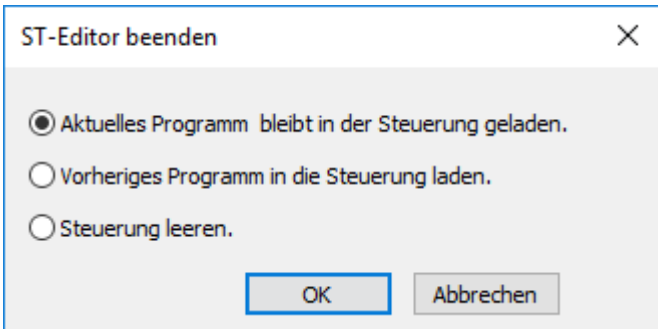
Debugging

Mit der Funktion *Debug* > *Debugging starten* (Menüleiste) stehen folgende Varianten zur Verfügung:

- Kaltstart: Ein bereits im Gerät laufendes Programm wird beendet, und das aktuelle Programm aus dem ST-Editor wird in das Gerät geladen. Alle Variablen (Retain- und Nicht-Retain-Variablen) werden auf ihren Default-Wert zurückgesetzt.
- Warmstart: Die Retain-Variablen werden nicht zurückgesetzt.
- Servicestart: Das aktuell im Gerät laufende Programm wird im ST-Editor verwendet und kann analysiert werden. Dabei werden keine Variablen zurückgesetzt. Voraussetzung für den Servicestart ist, dass der im ST-Editor kompilierte Code mit dem Code im Gerät übereinstimmt.

ST-Editor beenden

Beim Beenden des ST-Editors mit *Editor* > *Speichern und beenden* (Menüleiste) stehen folgende Möglichkeiten zur Auswahl (sofern der Debugging-Modus aktiv war):



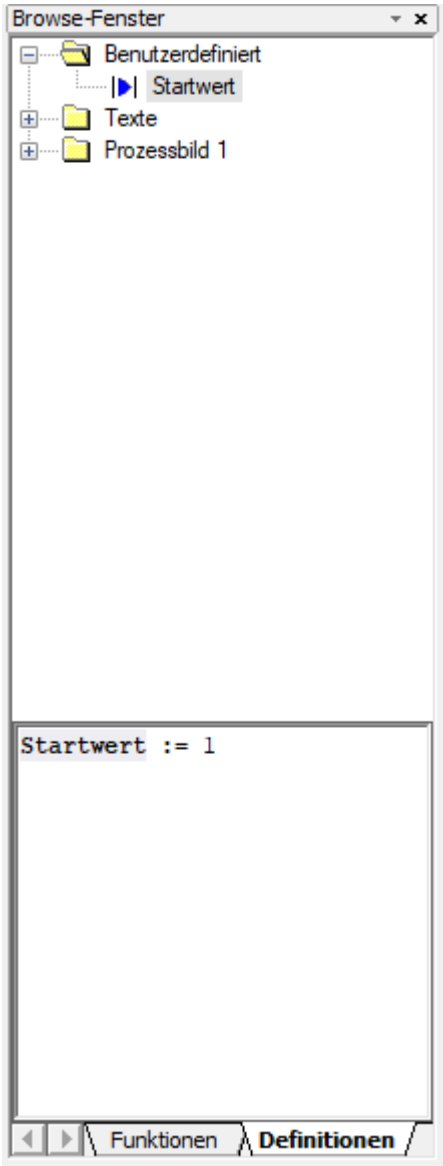
- *Aktuelles Programm bleibt in der Steuerung geladen.*
  Das zuletzt im Debugging-Modus ins Gerät geladene Programm soll verwendet werden.
- *Vorheriges Programm in die Steuerung laden.*
  Das zuletzt im Debugging-Modus ins Gerät geladene Programm soll nicht verwendet werden. Stattdessen wird das vorherige (dauerhaft ins Gerät geladene) Programm verwendet, sofern vorhanden.
- *Steuerung leeren.*
  Weder das zuletzt im Debugging-Modus ins Gerät geladene Programm, noch das vorherige Programm soll verwendet werden.

HINWEIS!

Im Debugging-Modus wird das Programm nur temporär in das Gerät geladen (keine Sicherung bei Netz-Ausfall). Damit Änderungen im Gerät dauerhaft wirksam werden, muss nach dem Beenden des ST-Editors die Setup-Datei zum Gerät übertragen werden (Fenster ST-Code mit „OK“ schließen, Datentransfer zum Gerät).

Das Browse-Fenster enthält unter dem Register „Definitionen“ die werkseitigen und die vom Anwender erstellten Definitionen (Namen für feste Werte). Werkseitig sind die Ordner „Benutzerdefiniert“ (leer) und „Texte“ vorhanden. Konfigurationsabhängig (Setup-Programm) werden weitere Ordner mit der Bezeichnung „Prozessbild x“ (x = Nummer des Prozessbilds im Setup-Programm) verwendet.

Benutzerdefinierte Definitionen



Der Ordner „Benutzerdefiniert“ enthält Definitionen, die vom Anwender im ST-Editor erstellt wurden. Um eine Definition zu erstellen, ist im Kontextmenü die Funktion „Neu“ zu verwenden (mit rechter Maustaste auf „Benutzerdefiniert“ klicken).

Durch eine benutzerdefinierte Definition kann einem beliebigen Wert (hier: 1) ein Name (hier: Startwert) zugeordnet werden, der anstelle des Wertes im ST-Code verwendet wird. Der Name sollte die Bedeutung des Wertes beschreiben, wodurch die Verständlichkeit des ST-Codes erhöht wird.

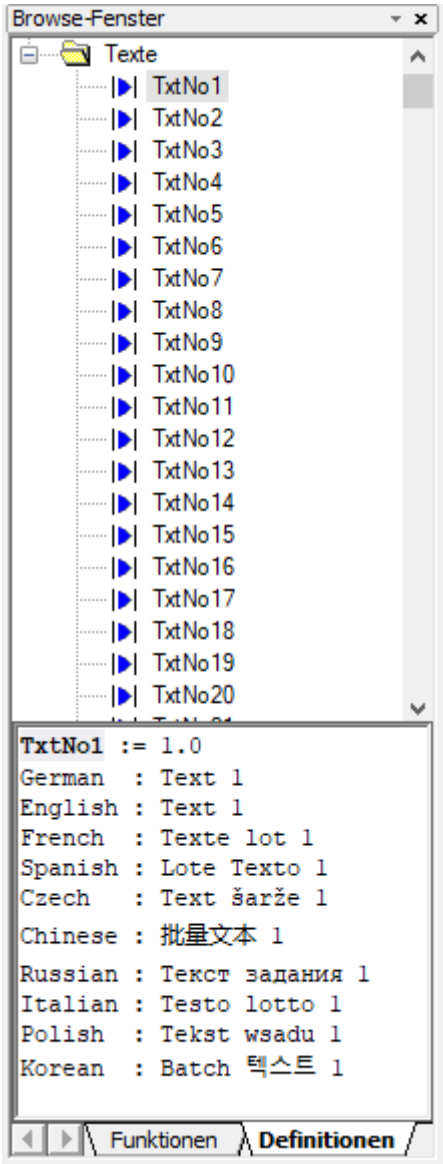
Ein weiterer Vorteil einer benutzerdefinierten Definition ergibt sich bei mehrfacher Verwendung im ST-Code. Im Falle einer Änderung des Wertes muss nicht der ST-Code geändert werden, sondern lediglich der Wert in der Definition.

Im Falle einer CASE-Anweisung können anstelle von festen Werten (Konstanten sind hier nicht erlaubt) Definitionen verwendet werden.

Beispiel:

```
CASE integer_variable OF
    definition1: integer_variable := 5;
    definition2, definition3: integer_variable := 10;
END_CASE;
```

Texte



Im Ordner „Texte“ sind alle Definitionen aufgelistet (TxtNo1 bis TxtNo...), die die Nummern der Texte in der Textliste des Setup-Programms repräsentieren. Diese Definitionen können im ST-Code verwendet werden, um zum Beispiel einen Chargentext zu adressieren.

Die Definitionen können aus dem oberen Teil des Fensters per Drag & Drop im Bearbeitungsfenster an der gewünschten Stelle in den ST-Code eingefügt werden (oder über Kontextmenü: Kopieren).

Im unteren Teil des Fensters wird in der ersten Zeile die Definition angezeigt (hier: TxtIndex0 := 0). In den folgenden Zeilen wird der jeweilige Text in den einzelnen Gerätesprachen dargestellt.

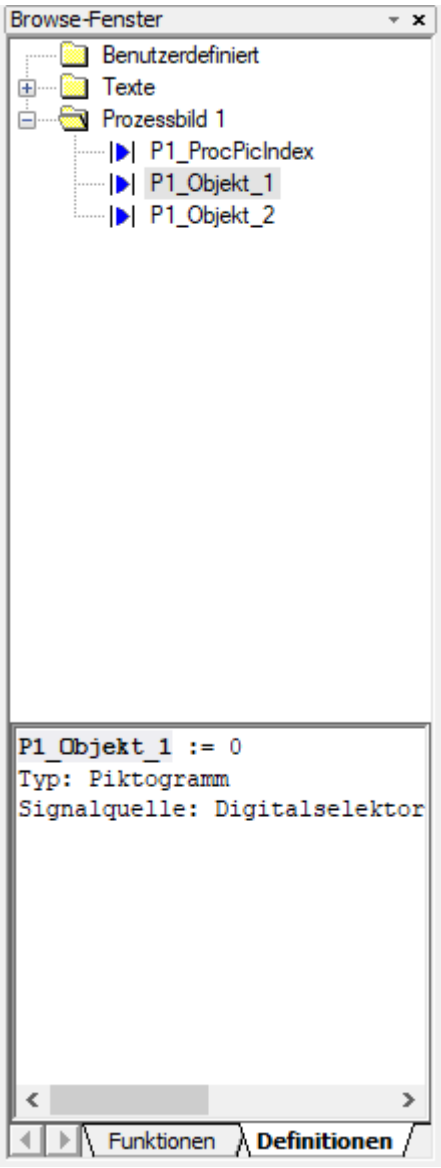
Die Texte selbst können auch im ST-Editor geändert werden (Kontextmenü: Ändern...). Beim Beenden des ST-Editors werden die Änderungen in die Konfiguration im Setup-Programm übernommen, wenn das Fenster „ST-Code“ im Setup-Programm mit „OK“ verlassen wird.

Diese Definitionen – sowie die Möglichkeit der Änderung von Texten im ST-Editor – sind insbesondere für folgende Zwecke vorgesehen:

– Prozessbild: Steuerung des Textes in einem Objekt durch einen ST-Analogausgang (Objekt „Textauswahl über Wert“)
– Chargenkonfiguration: Vorgabe des Textes der rechten Spalte durch einen ST-Analogausgang (Wert als Textnummer)

Weitere Informationen hierzu sind der Betriebsanleitung des Gerätes zu entnehmen.

Prozessbild x



Wurde im Setup-Programm für ein Prozessbildobjekt ein Objektname vergeben, dann wird dieser zusammen mit der Prozessbildnummer im Browse-Fenster dargestellt.

In einem solchen Fall ist ein Ordner für das betreffende Prozessbild (hier: Prozessbild 1) vorhanden. Im Ordnernamen entspricht die Nummerierung der Prozessbilder derjenigen im Setup-Programm (Prozessbilder 1 bis 10).

3 Definitionen

Der Ordner enthält die Definitionen für das Prozessbild (hier: P1_ProcPicIndex) und für jedes Objekt, für das im Prozessbild ein Name vergeben wurde. Dabei wird dem vom Anwender vergebenen Objektnamen (hier: Objekt 1, ...) ein „Px_“ für das betreffende Prozessbild vorangestellt (hier: P1_Objekt_1, ...). So ist gewährleistet, dass Objektnamen, die in mehreren Prozessbildern verwendet werden, im ST-Code eindeutig sind. Ein Leerzeichen im Objektnamen wird durch einen Unterstrich ersetzt.

Der untere Teil des Fensters enthält Informationen zu der aktuell ausgewählten (im oberen Teil markierten) Definition. In der ersten Zeile ist die Definition aufgeführt (hier: P1_Objekt_1 := 0). Im Falle eines Objektes sind weitere Zeilen vorhanden, die zusätzliche Informationen enthalten.

Die in den Definitionen vergebenen Werte entsprechen der Nummerierung der Prozessbilder (0 bis 9) bzw. der Objekte (0 bis 99) im ST-Code.

Beispiel: Definition für Prozessbild 1
P1_ProcPicIndex := 0

Beispiel: Definition für Objekt 1 im Prozessbild 1
P1_Objekt_1 := 0

Ändert sich die Nummer des Objektes im Prozessbild, indem das Objekt im Prozessbildeditor verschoben wird (andere Reihenfolge, z. B. wegen Überlagerung von Objekten), wird die geänderte Nummer beim nächsten Start des ST-Editors an diesen übergeben. In der betreffenden Definition wird dann die bisherige Nummer automatisch durch die geänderte Nummer ersetzt, so dass der ST-Code nicht geändert werden muss.

Diese Definitionen sollten insbesondere innerhalb der speziellen Funktionen verwendet werden, mit denen bestimmte Eigenschaften von Prozessbildobjekten abgefragt und geändert werden.

⇨ Kapitel 8.9 „Weitere Funktionen“, Seite 46

HINWEIS!

Im ST-Code beginnt die Nummerierung der Prozessbilder und Objekte mit 0 (Prozessbilder 0 bis 9, Objekte 0 bis 99). Im Setup-Programm beginnt die Nummerierung jeweils mit 1 (Prozessbilder 1 bis 10, Objekte 1 bis 100).

HINWEIS!

Wird der Prozessbild-Editor im Setup-Programm mit „OK“ beendet, wird der ST-Code neu kompiliert. Somit ist sichergestellt, dass die aktuellen Definitionen im ST-Code verwendet werden.

Zu den Systemvariablen gehören die Eingangs- und Ausgangsvariablen sowie die internen Variablen.

Über die Eingangs- und die Ausgangsvariablen können Werte unterschiedlichen Datentyps zwischen dem Gerät und dem im Gerät integrierten ST-Modul ausgetauscht werden. Die internen Variablen sind nur innerhalb des ST-Moduls von Bedeutung und können zur Realisierung bestimmter Funktionen verwendet werden.

4.1 Eingangsvariablen

Die Eingangsvariablen stellen Werte des Geräts für die Verwendung im ST-Modul zur Verfügung.

Die Zuweisung der Werte zu den Eingangsvariablen erfolgt im Setup-Programm, indem die betreffenden Signale aus den sogenannten Selektoren (Analogselektor bzw. Digitalselektor) ausgewählt werden.

Variablen mit fester Zuordnung im Gerät stehen nicht in den Selektoren zur Verfügung.

bool_in

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
bool_in01 ... bool_in40	bool_in01 ... bool_in40	Boolesche Eingangsvariablen; flexible Zuweisung im Setup-Programm (Digitalselektor)

real_in

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
real_in01 ... real_in40	real_in01 ... real_in40	Real-Eingangsvariablen; flexible Zuweisung im Setup-Programm (Analogselektor)

dword_in

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
dword_in01 ... dword_in08	(keine Bezeichnung, keine Zuordnung)	Diese Double-Word-Eingangsvariablen stehen geräteseitig nicht zur Verfügung. Sie können im ST-Code als Zwischenspeicher verwendet werden.

rtc.cdt

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
rtc.cdt	(keine Bezeichnung, feste Zuordnung)	Diese Eingangsvariable vom Typ DT liefert das aktuelle Datum und die aktuelle Uhrzeit des Gerätes. ⇨ Kapitel 8.8 „Datum und Uhrzeit“, Seite 45

4.2 Ausgangsvariablen

Mit den Ausgangsvariablen werden die im ST-Modul erzeugten Werte an das Gerät übergeben.

Die Ausgangsvariablen stehen im Setup-Programm und im Gerät in den Selektoren (Analogselektor bzw. Digitalselektor) für die Konfiguration zur Verfügung und können individuell verwendet werden.

Zu bestimmten Ausgangsvariablen gibt es im Variablenfenster zugehörige Variablen mit den Bezeichnungen „...conf“ und „...sec“. Mit diesen Variablen lässt sich im ST-Code ein definiertes Verhalten im Fehlerfall realisieren.

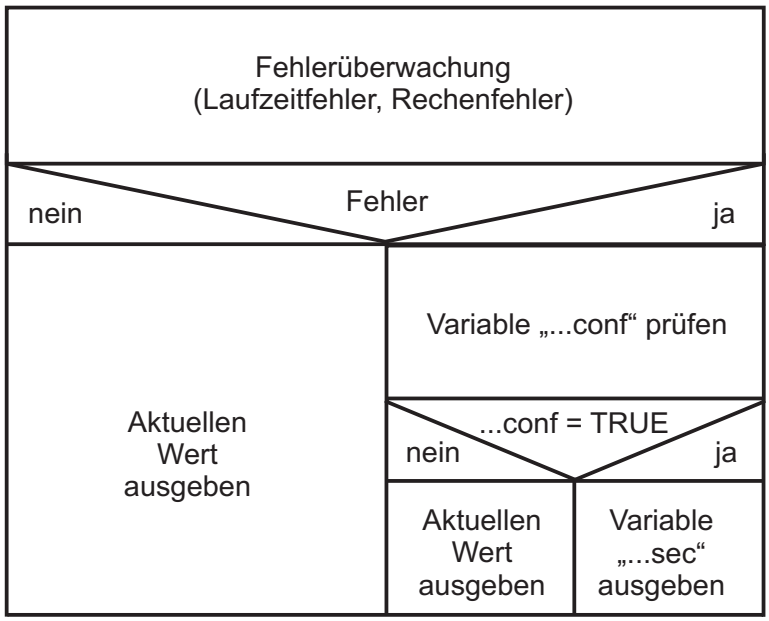
- Die **Variable „...conf“** kann im ST-Code gesetzt werden, um zu entscheiden, welchen Wert die zugehörige Ausgangsvariable im Fehlerfall annimmt: FALSE = aktueller Wert; TRUE = Ersatzwert.

  Default-Einstellung: FALSE
- Die **Variable „...sec“** enthält den Ersatzwert; sie kann im ST-Code auf einen bestimmten Wert oder auf den Default-Wert gesetzt werden.

  Im Falle der REAL-Variablen wird der Ersatzwert auf Einhaltung der Skalierungsgrenzen geprüft. Liegt er außerhalb der Grenzen, wird der Default-Wert verwendet (Mathematik-Fehlerwert 5.0E+37).

  Zum Setzen auf den Default-Wert steht die Funktion SET_DEFAULT zur Verfügung.

  ⇨ Kapitel 8.9 „Weitere Funktionen“, Seite 46



bool_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
bool_out01 ... bool_out40	ST-Digitalausg. 1 ... ST-Digitalausg. 40	Boolesche Ausgangsvariablen; flexible Verwendung im Setup-Programm und im Gerät (Digitalselektor) Im Setup-Programm kann jeder Variablen ein Beschreibungstext zugewiesen werden.

real_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
real_out01 ... real_out40	ST-Analogausg. 1 ... ST-Analogausg. 40	Real-Ausgangsvariablen; flexible Verwendung im Setup-Programm und im Gerät (Analogselektor) Im Setup-Programm kann jeder Variablen ein Beschreibungstext zugewiesen werden. Außerdem sind dort Einstellungen erforderlich, die die Variable betreffen (z. B. Skalierung).

alarm

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
alarm01 ... alarm32	ST-Alarmausg. 1 ... ST-Alarmausg. 32	Boolesche Ausgangsvariablen; flexible Verwendung im Setup-Programm und im Gerät (Digitalselektor) Diese Variablen haben im Gegensatz zu den Variablen bool_out eine spezielle Bezeichnung. Sie können jedoch im ST-Code frei verwendet werden.

error_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
error_out	ST-Fehler	Boolesche Ausgangsvariable; flexible Verwendung im Setup-Programm und im Gerät (Digitalselektor) Diese Variable wird auf TRUE gesetzt, wenn der ST-Code nicht weiter abgearbeitet werden kann, wie beispielsweise in folgenden Fällen: • Division durch 0 • falscher Array-Zugriff • Laufzeitüberschreitung Die Variable wird wieder auf FALSE gesetzt, wenn der ST-Code nach Auftreten des Fehlers einmal komplett durchlaufen wurde.

dword_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
dword_out01 ... dword_out08	(keine Bezeichnung, keine Zuordnung)	Diese Double-Word-Ausgangsvariablen stehen geräteseitig nicht zur Verfügung. Sie können im ST-Code als Zwischenspeicher verwendet werden.

4 Systemvariablen

4.3 Interne Variablen

Die internen Variablen können zur Realisierung bestimmter Funktionen im ST-Code verwendet werden.

ext_error

Bezeichnung	Beschreibung
ext_error	Integer-Variable Die Variable zeigt Fehler an, die bei Zugriffen außerhalb des ST-Moduls aufgetreten sind. Im Fehlerfall liefert die Variable einen Wert > 0 (kein Fehler = 0).

reset_flag

Bezeichnung	Beschreibung
reset_flag	Boolesche Variable Die Variable wird nach Netz-Ein auf TRUE gesetzt und dient zur Initialierung von Variablen. Beispiel: ⇨ Kapitel 7.1 „Auswahlanweisungen“, Seite 31 Wird nach Netz-Ein der ST-Code einmal komplett durchlaufen, ohne dass ein Fehler auftritt, wird die Variable auf FALSE gesetzt. Anderenfalls bleibt sie auf TRUE. Debug-Modus: Beim ersten Start im Debug-Modus oder Start nach einem Programmstopp ist die Variable TRUE. Ab einem zweiten Durchlauf wird sie auf FALSE gesetzt.

sys_error

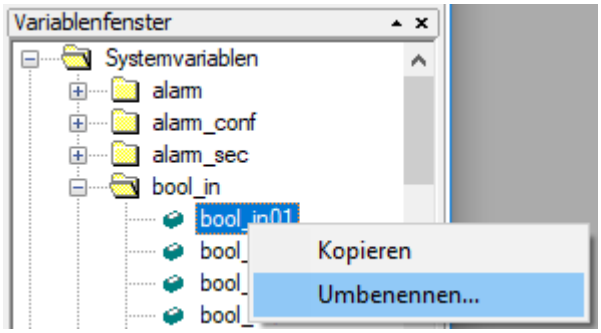
Bezeichnung	Beschreibung
sys_error	Real-Variable Die Variable liefert einen Fehlerwert, wenn während des Programmablaufs ein Fehler auftritt (kein Fehler = 0): 1.00E+37 = allgemeiner Fehlerwert 1.0E+37 = Bereichsunterschreitung (Unterlauf) 2.0E+37 = Bereichsüberschreitung (Überlauf) 3.0E+37 = ungültiger Eingangswert 4.0E+37 = Division durch Null 5.0E+37 = fehlerhafter Mathematikwert 7.0E+37 = ungültiger Wert (allgemein)

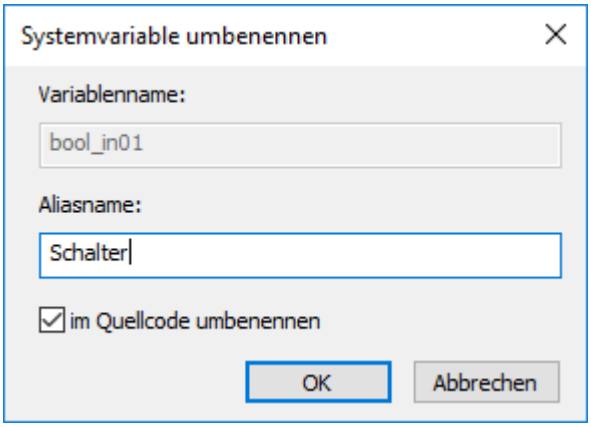
week_day

Bezeichnung	Beschreibung
week_day	Integer-Variable Die Variable enthält den aktuellen Wochentag: 0 = Sonntag, 1 = Montag, 2 = Dienstag, 3 = Mittwoch, 4 = Donnerstag, 5 = Freitag, 6 = Samstag Der Wochentag wird aus der Eingangsvariablen rtc.cdt abgeleitet.

4.4 Alias-Namen

Für Systemvariablen können Alias-Namen vergeben werden:



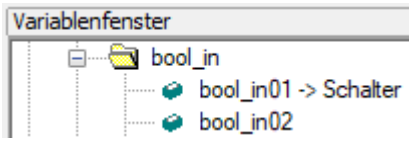


Durch die Verwendung eines Alias-Namens kann die Funktion der Systemvariablen beschrieben werden. In diesem Beispiel wird der Systemvariablen bool_in01 der Alias-Name „Schalter“ zugewiesen.

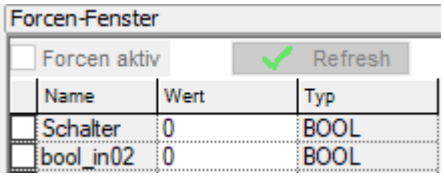
Beim Beenden des Dialogs mit „OK“ wird der eingegebene Name auf Eindeutigkeit überprüft. Wird der Name bereits verwendet, erscheint ein entsprechender Hinweis. In diesem Fall muss ein anderer Name vergeben werden.

Ist die Funktion „Im Quellcode umbenennen“ aktiv (Haken gesetzt), wird der Variablenname im vorhandenen ST-Code durch den Alias-Namen ersetzt. Anderenfalls wird der Alias-Name erst dann im ST-Code verwendet, wenn die Variable erneut eingefügt wird.

Nach der Zuweisung des Alias-Namens werden im Variablenfenster der Variablenname und der Alias-Name angezeigt:



Im Forcen-Fenster wird ausschließlich der Alias-Name angezeigt:



4 Systemvariablen

5.1 Datentypen

Das ST-Modul unterstützt folgende Datentypen:

- Boolscher Wert (BOOL)
- Integer-Zahl ohne Vorzeichen, 2 Byte (UINT)
- Double-Integer-Zahl ohne Vorzeichen, 4 Byte (UDINT)
- Gleitkommazahl (REAL)
- Datum und Uhrzeit (DT oder DATE_AND_TIME)

HINWEIS!

Bei Operationen findet keine Bereichsüberwachung der Datentypen statt.
Ausnahmen: Quadratwurzelfunktion (SRQT), Kehrwert (1/x)

Boolescher Wert

Schlüsselwort: BOOL

Wertebereich: TRUE oder FALSE

Deklaration einer Variablen (Beispiel):

```
VAR
    bExample : BOOL;
END_VAR
```

Deklaration einer Konstanten (Beispiel):

```
VAR CONSTANT
    bConstExample : BOOL := FALSE;
END_VAR
```

Zuweisung (Beispiel):

```
bExample := TRUE;
```

Integer-Zahl (2 Byte)

Schlüsselwort: UINT

Wertebereich: 0 bis 65535 (0 bis 2^{16}-1)

Deklaration einer Variablen (Beispiel):

```
VAR
    uiExample : UINT;
END_VAR
```

Deklaration einer Konstanten (Beispiel):

```
VAR CONSTANT
    uiConstExample : UINT := 0;
END_VAR
```

Zuweisung (Beispiel):

```
uiExample := 1;
```

Double-Integer-Zahl (4 Byte)

Schlüsselwort: UDINT

Wertebereich: 0 bis 4294967295 (0 bis 2^{32}-1)

Deklaration einer Variablen (Beispiel):

```
VAR
    udExample : UDINT;
END_VAR
```

Deklaration einer Konstanten (Beispiel):

```
VAR CONSTANT
    udConstExample : UDINT := 100000;
END_VAR
```

Zuweisung (Beispiel):

```
udExample := 200000;
```

Gleitkommazahl

Schlüsselwort: REAL

Wertebereich: -1.5E-45 bis 1.0E37
(Werte ≥ 1.0E37 werden als Fehlerwerte interpretiert.)

Deklaration einer Variablen (Beispiel):

```
VAR
    rExample : REAL;
END_VAR
```

Deklaration einer Konstanten (Beispiel):

```
VAR CONSTANT
    rConstExample : REAL := 0.0;
END_VAR
```

Zuweisung (Beispiel):

```
rExample := 1.0;
```

Datum und Uhrzeit

Schlüsselworte: DT oder DATE_AND_TIME

Wertebereich: yyyy-mm-dd-hh:mm:ss

Deklaration einer Variablen (Beispiel):

```
VAR
    dtExample : DT;
END_VAR
```

Deklaration einer Konstanten (Beispiel):

```
VAR CONSTANT
    dtConstExample : DT := dt#2017-12-31-23:59:59;
END_VAR
```

Zuweisung (Beispiel):

```
dtExample := dt#2017-01-01-08:00:00;
```

5.2 Felder

Alle Datentypen können auch als eindimensionales oder zweidimensionales Feld (Array) deklariert werden (siehe Norm DIN IEC 61131-3). In den folgenden Abschnitten wird exemplarisch der Datentyp REAL verwendet.

Eindimensionales Feld

Deklaration von Variablen (Beispiel: Feld mit 3 Variablen):

```
VAR
    rArrayExample : ARRAY [0..2] OF REAL;
END_VAR
```

Deklaration von Konstanten (Beispiel: Feld mit 3 Konstanten):

```
VAR CONSTANT
    rArrayConstExample : ARRAY [0..2] OF REAL := [0.0, 0.1, 0.2];
END_VAR
```

Zuweisung (Beispiel):

```
rArrayExample[0] := 0.0;
rArrayExample[1] := 0.1;
rArrayExample[2] := 0.2;
```

Zweidimensionales Feld

Deklaration von Variablen (Beispiel: Feld mit 3 x 2 Variablen):

```
VAR
    rArrayExample : ARRAY [0..2, 0..1] OF REAL;
END_VAR
```

Deklaration von Konstanten (Beispiel: Feld mit 3 x 2 Konstanten):

```
VAR CONSTANT
    rArrayConstExample : ARRAY [0..2, 0..1] OF REAL :=
    [0.0, 0.1, 0.2,
    1.0, 1.1, 1.2];
END_VAR
```

Zuweisung (Beispiel):

```
rArrayExample[0,0] := 0.0;
rArrayExample[1,0] := 0.1;
rArrayExample[2,0] := 0.2;
rArrayExample[0,1] := 1.0;
rArrayExample[1,1] := 1.1;
rArrayExample[2,1] := 1.2;
```

5 Datentypen und Felder

In der folgenden Tabelle sind alle Operatoren aufgeführt, die das ST-Modul unterstützt. Die Reihenfolge in der Tabelle richtet sich nach der Rangfolge der Operatoren, beginnend mit dem höchsten Rang.

Operation	Symbol	Zulässige Datentypen	Beispiel
Klammerung	(Ausdruck)		`a := 3.0 * (b - 1.0);`
Funktion	Bezeichner (Argumentliste)		`i := MIN (3, j);`
Negation	-	REAL	`a := -a;`
Komplement	NOT	BOOL, UINT, UDINT[a]	`a := NOT b;`
Multiplikation	*	UINT, UDINT, REAL	`i := 5 * j;`
Division	/		`a := 5.0 / b;`
Modulo	MOD	UINT, UDINT	`j := i MOD 10;`
Addition	+	UINT, UDINT, REAL	`i := 5 + j;`
Subtraktion	-		`a := b - 5.0E20;`
Vergleich	<, >, <=, >=	UINT, UDINT, REAL, DATE_AND_TIME	`bExample := 5 <= j;`
Gleichheit	=	BOOL, UINT, UDINT, REAL, DATE_AND_TIME	`bExample := 5 = i;`
Ungleichheit	<>		`bExample := 5 <> j;`
UND	& oder AND	BOOL, UINT, UDINT[a]	`bExample := x AND y;`
Exklusives ODER	XOR	BOOL, UINT, UDINT[a]	`bExample := x XOR y;`
ODER	OR	BOOL, UINT, UDINT[a]	`bExample := x OR y;`

[a] Bei Booleschen Variablen findet eine logische Verknüpfung statt, bei Integer-Variablen eine bitweise Verknüpfung.

6 Operatoren

Das ST-Modul unterstützt die folgenden Arten von Anweisungen:

- Auswahlanweisungen (IF, CASE)
- Wiederholanweisungen (FOR, WHILE, REPEAT)

7.1 Auswahlanweisungen

Eine Auswahlanweisung führt aufgrund einer festgelegten Bedingung eine Anweisung oder eine Gruppe von Anweisungen aus.

IF-Anweisung

Die IF-Anweisung legt fest, dass eine Gruppe von Anweisungen nur ausgeführt wird, wenn der zugehörige boolesche Ausdruck den Wert TRUE (wahr) liefert. Falls die Bedingung falsch ist, darf entweder keine Anweisung ausgeführt werden oder die Anweisungsgruppe ist auszuführen, die dem Schlüsselwort ELSE folgt (oder dem Schlüsselwort ELSIF, falls seine zugehörige boolesche Bedingung wahr ist).

Schlüsselworte:

IF, THEN, ELSIF, ELSE, END_IF

Beispiel:

```
VAR
    i : UINT;
END_VAR


IF reset_flag THEN
    bool_out01 := FALSE;
    bool_out02 := FALSE;
    i := 1;
END_IF;
(* nach dem Reset werden zunächst die Ausgänge bool_out01 und bool_out02 zurückgesetzt
und i auf 1 gesetzt *)


IF i>0 AND i<=100 THEN
    bool_out01 := TRUE;
    bool_out02 := FALSE;
ELSIF i>100 AND i<=200 THEN
    bool_out01 := FALSE;
    bool_out02 := FALSE;
ELSE
    i := 1;
    bool_out01 := FALSE;
    bool_out02 := FALSE;
END_IF;
(* im Durchlauf 1 bis 100 wird der Ausgang bool_out01 gesetzt und bool_out02 zurück-
gesetzt *)
(* im Durchlauf 101 bis 200 wird der Ausgang bool_out01 zurückgesetzt und bool_out02
gesetzt *)
(* im Durchlauf 201 wird i auf 1 gesetzt und die Ausgänge bool_out01 und bool_out02
werden zurückgesetzt; das ganze beginnt von vorne *)


i := i + 1;
(* i wird um 1 erhöht *)
```

CASE-Anweisung

Mit der CASE-Anweisung werden mehrere bedingte Anweisungen zusammengefasst. Sie besteht aus einen Ausdruck, der eine Integer-Variable enthält, und einer Liste von Anweisungsgruppen. Jede Gruppe ist durch eine Marke gekennzeichnet, die aus einer oder mehreren (durch Komma getrennten) ganzen Zahlen besteht. Der Wert der Variablen bestimmt die Marke und somit die Anweisungsgruppe, die ausgeführt wird. Optional kann auch das Schlüsselwort ELSE verwendet werden. Dann wird, wenn kein Wert zutrifft, die Anweisungsgruppe ausgeführt, die dem Schlüsselwort ELSE folgt.

Schlüsselworte:

CASE, OF, ELSE, END_CASE

Beispiel:

```
VAR
    b1: BOOL;
    b2: BOOL;
END_VAR


CASE i OF (* Variable i muss UINT oder UDINT sein *)
    1: b1 := TRUE;
    2, 3, 4: b2 := TRUE;
ELSE
    b1 := FALSE;
    b2 := FALSE;
END_CASE;
```

7.2 Wiederholanweisungen

Mit Wiederholanweisungen (Iterationen) werden Anweisungen und Gruppen von Anweisungen wiederholt ausgeführt.

Es gibt drei Typen von Wiederholanweisungen:

- FOR
- WHILE
- REPEAT UNTIL

FOR-Anweisung

Die FOR-Anweisung wird verwendet, wenn die Anzahl der Wiederholungen festliegt.

Schlüsselworte:

FOR, TO, BY, DO, END_FOR

Beispiele:

```
VAR
    i: UINT;
    j: UINT;
    a: UINT;
    b: UINT;
END_VAR
FOR i := 10 TO 100 BY 10 DO
    j := j + i;
END_FOR;
(* i läuft von 10 bis 100 in Schritten von 10 (10, 20, 30, …, 100) *)
FOR a := 1 TO 5 DO
    b := b + a;
END_FOR;
(* ohne Angabe von „BY x“ wird die Laufvariable in jedem Durchlauf um 1 erhöht (1, 2, 3, 4, 5) *)
```

WHILE-Anweisung

Die WHILE-Anweisung bewirkt, dass eine Gruppe von Anweisungen wiederholt ausgeführt wird, bis der zugehörige boolesche Ausdruck falsch (FALSE, unwahr) ist. Ist der boolesche Ausdruck von Beginn an falsch, wird die Gruppe von Anweisungen überhaupt nicht ausgeführt.

Schlüsselworte:

WHILE, DO, END_WHILE

Beispiel:

```
WHILE j < 100 DO
    j := j + 2;
END_WHILE;
```

REPEAT-Anweisung

Die REPEAT-Anweisung bewirkt, dass eine Gruppe von Anweisungen bis zum Schlüsselwort UNTIL wiederholt (und mindestens einmal) ausgeführt wird, bis die zugehörige boolesche Bedingung wahr (TRUE) wird.

Schlüsselworte:

REPEAT, UNTIL, END_REPEAT

Beispiel:

```
REPEAT
    i := i - 2;
    a := a + 2.0;
UNTIL i = 0 END_REPEAT;
```

EXIT

Mit EXIT wird eine Wiederholanweisung beendet, bevor die Endbedingung der Iteration erfüllt ist. Wird EXIT innerhalb eines geschachtelten Wiederholungskonstrukts ausgeführt, wird die innerste Schleife verlassen, innerhalb der EXIT liegt.

HINWEIS!

Bei Wiederholanweisungen muss darauf geachtet werden, dass sich keine Endlosschleifen oder sehr lange laufende Schleifen ergeben. Die Laufzeit wird überwacht.

Das ST-Modul unterstützt folgende Funktionen:

- Typumwandlung (Konvertierung)
- Arithmetische Funktionen
- Numerische Funktionen
- Bitfolge-Funktionen (Shift-Funktionen)
- Logische Funktionen
- Auswahl (Statistik)
- Vergleich
- Datum und Uhrzeit
- Weitere Funktionen (Sonstige Funktionen)

8.1 Typumwandlung

Zulässige Datentypen

Argument: UINT, UDINT

Ergebnis: BOOL, UINT, UDINT, REAL

INT_TO_REAL

Wandelt eine INTEGER- in eine REAL-Zahl um.

Beispiel:

```
a := INT_TO_REAL(10); (* a := 10.0 *)
```

INT_TO_DINT

Wandelt eine INTEGER- in eine DOUBLE-INTEGER-Zahl um.

Beispiel:

```
a := INT_TO_DINT(10); (* a := 10 *)
```

INT_TO_BOOL

Wandelt eine INTEGER-Zahl in einen BOOL-Wert um.

Das Ergebnis ist FALSE, wenn das Argument 0 ist. In allen anderen Fällen ist das Ergebnis TRUE.

Beispiele:

```
a := INT_TO_BOOL(0); (* a = FALSE *)
b := INT_TO_BOOL(1); (* b = TRUE *)
c := INT_TO_BOOL(8); (* c = TRUE *)
```

DINT_TO_REAL

Wandelt eine DOUBLE-INTEGER-Zahl in eine REAL-Zahl um.

Beispiele:

```
a := DINT_TO_REAL(100000); (* a = 100000.0 *)
```

DINT_TO_INT

Wandelt eine DOUBLE-INTEGER-Zahl in eine INTEGER-Zahl um.

Dabei werden nur die unteren beiden Byte ausgewertet (Bit 0 bis Bit 15).

Beispiele:

```
a := DINT_TO_INT(1); (* a = 1; 1 = 0000 0000 0000 0001 = 0x0001 *)
b := DINT_TO_INT(65535); (* b = 65535; 65535 = 1111 1111 1111 1111 = 0xFFFF *)
c := DINT_TO_INT(65536); (* c = 0; 65536 = 0001 0000 0000 0000 0000 = 0x10000 *)
d := DINT_TO_INT(65537); (* d = 1, 65537 = 0001 0000 0000 0000 0001 =0x10001 *)
```

DINT_TO_BOOL

Wandelt eine DOUBLE-INTEGER-Zahl in einen BOOL-Wert um.

Das Ergebnis ist FALSE, wenn das Argument 0 ist. In allen anderen Fällen ist das Ergebnis TRUE.

Beispiele:

```
a := DINT_TO_BOOL(0); (* a = FALSE: *)
b := DINT_TO_BOOL(1); (* b = TRUE *)
c := DINT_TO_BOOL(100000); (* c = TRUE *)
```

REAL_TO_INT

Wandelt eine REAL-Zahl mit Rundung in eine INTEGER-Zahl um.

Beispiele:

```
a := REAL_TO_INT(1.378); (* a = 1 *)
b := REAL_TO_INT(1.897); (* b = 2 *)
```

REAL_TO_DINT

Wandelt eine REAL-Zahl mit Rundung in eine DOUBLE-INTEGER-Zahl um.

Beispiele:

```
a := REAL_TO_DINT(100000.378); (* a = 100000 *)
b := REAL_TO_DINT(100000.897); (* b = 100001 *)
```

8.2 Arithmetische Funktionen

Zulässige Datentypen

Argument: UINT, UDINT, REAL (bei Negation nur REAL, bei Modulo nur UINT oder UDINT)

Ergebnis: UINT, UDINT, REAL

Addition

Die Addition ist eine erweiterbare Funktion. Als Ergebnis wird die Summe der Argumente zurückgegeben.

Beispiel:

```
OUT := IN1 + IN2 + ... INn;
```

Subtraktion

Als Ergebnis wird das Zweite von dem ersten Argument abgezogen.

Beispiel:

```
OUT := IN1 - IN2;
```

Multiplikation

Die Multiplikation ist eine erweiterbare Funktion. Das Ergebnis ergibt sich aus der Multiplikation der Argumente.

Beispiel:

```
OUT := IN1 * IN2 * ...INn;
```

Division

Als Ergebnis wird der Quotient aus den beiden Argumenten zurückgegeben.

Das Ergebnis der Division von ganzen Zahlen (UINT oder UDINT) ist eine ganze Zahl mit Abschneiden der Nachkommastellen (Beispiel: 7/3 = 2).

Beispiel:

```
OUT := IN1 / IN2;
```

Negation

REAL-Zahlen können negiert werden.

Beispiel:

```
OUT := -IN;
```

Modulo

Die Argumente der Modulo-Funktion (%) müssen ganze Zahlen (UINT oder UDINT) sein. Das Ergebnis ist der Rest aus der Division beider Zahlen (Beispiel: 17/3 = 5, Rest 2).

Beispiel:

```
OUT := IN1 % IN2;
```

8.3 Numerische Funktionen

Zulässige Datentypen

Argument: REAL

Ergebnis: REAL

ABS (IN)

Liefert den Absolutwert von IN.

Beispiel:

```
IN := -2.3;
OUT := ABS(IN); (* OUT = 2.3 *)
```

SQRT (IN)

Liefert die Quadratwurzel von IN.

Ist IN negativ, wird der Fehlerwert 5.0E+37 geliefert.

Beispiel:

```
IN := 4.0;
OUT := SQRT(IN); (* OUT = 2.0 *)
```

LN (IN)

Liefert den natürlichen Logarithmus von IN.

Beispiel:

```
IN := 2.718282;
OUT := LN(IN);
(* 2.718282 = Zahl e (gerundet); OUT = 1.0 (gerundet) *)
```

LOG (IN)

Liefert den Logarithmus zur Basis 10 von IN.

Beispiel:

```
IN := 100.0;
OUT := LOG(IN); (* OUT = 2.0 *)
```

EXP (IN)

Liefert die natürliche Exponentialfunktion von IN.

Beispiel:

```
IN := 2.0;
OUT := EXP(IN); (* OUT = 7.389056 *)
```

SIN (IN)

Liefert den Sinus von IN (Bogenmaß).

Beispiel:

```
IN := 1.5708;
OUT := SIN(IN);
(* 1.5708 entspricht 90°; OUT = 1.0 *)
```

COS (IN)

Liefert den Cosinus von IN (Bogenmaß).

Beispiel:

```
IN := 0.0;
OUT := COS(IN);
(* 0.0 entspricht 0°; OUT = 1.0 *)
```

TAN (IN)

Liefert den Tangens von IN (Bogenmaß).

Beispiel:

```
IN := 0.7854;
OUT := TAN(IN);
(* 0.7854 entspricht 45°; OUT = 1.0 *)
```

ASIN (IN)

Liefert den Arcussinus (Bogenmaß) von IN.

Beispiel:

```
IN := 1.0;
OUT := ASIN(IN);
(* OUT = 1.5708; entspricht 90° *)
```

ACOS (IN)

Liefert den Arcuscosinus (Bogenmaß) von IN.

Beispiel:

```
IN := 1.0;
OUT := ASIN(IN);
(* OUT = 0.0; entspricht 0° *)
```

ATAN (IN)

Liefert den Arcustangens (Bogenmaß) von IN.

Beispiel:

```
IN := 1.0;
OUT := ATAN(IN);
(* OUT = 0.7854; entspricht 45° *)
```

8.4 Bitfolge-Funktionen

Zulässige Datentypen

Argument: UINT, UDINT

Ergebnis: UINT, UDINT

SHL (IN, n)

Verschiebt die Bitfolge des Arguments IN nach links um n Bit. Nachrückende Stellen von rechts werden mit 0 gefüllt.

Beispiel:

```
IN := 255;          (* 0000 0000 1111 1111 *)
OUT := SHL(IN, 4); (* 0000 1111 1111 0000; OUT = 4080 *)
```

SHR (IN, n)

Verschiebt die Bitfolge des Arguments IN nach rechts um n Bit. Nachrückende Stellen werden von links mit 0 gefüllt.

Beispiel:

```
IN := 255;          (* 0000 0000 1111 1111 *)
OUT := SHR(IN, 4); (* 0000 0000 0000 1111; OUT = 15 *)
```

ROL (IN, n)

Rotiert die Bitfolge des Arguments IN nach links um n Bit im Kreis.

Beispiel:

```
IN := 43690;        (* 1010 1010 1010 1010 *)
OUT := ROL(IN, 1); (* 0101 0101 0101 0101; OUT = 21845 *)
```

ROR (IN, n)

Rotiert die Bitfolge des Arguments IN nach rechts um n Bit im Kreis.

Beispiel:

```
IN := 21845;        (* 0101 0101 0101 0101 *)
OUT := ROR(IN, 1); (* 1010 1010 1010 1010; OUT = 43690 *)
```

8.5 Logische Funktionen

Bei booleschen Variablen wird logisch, bei UINT- und UDINT-Variablen bitweise verknüpft.

Bei mehr als zwei Parametern wird immer paarweise von links nach rechts verknüpft.

Zulässige Datentypen

Operand: BOOL, UINT, UDINT

Ergebnis: BOOL, UINT, UDINT

AND

UND-Verküpfung

Beispiele:

Logische Verknüpfung:

```
x := TRUE;
y := FALSE;
z := x AND y; (* z = FALSE *)
```

Bitweise Verknüpfung:

```
a := 5;       (* 0101 *)
b := 6;       (* 0110 *)
c := a AND b; (* 0100 = 4 *)
```

OR

ODER-Verknüpfung

Beispiele:

Logische Verknüpfung:

```
x := TRUE;
y := FALSE;
z := x OR y; (* z = TRUE *)
```

Bitweise Verknüpfung:

```
a := 5;      (* 0101 *)
b := 6;      (* 0110 *)
c := a OR b; (* 0111 = 7 *)
```

XOR

Exklusiv-ODER-Verknüpfung

Beispiele:

Logische Verknüpfung:

```
x := TRUE;
y := FALSE;
z := x XOR y; (* z = TRUE *)
```

Bitweise Verknüpfung:

```
a := 5;       (* 0101 *)
b := 6;       (* 0110 *)
c := a XOR b; (* 0011 = 3 *)
```

NOT

Invertierung (Komplement bilden)

Beispiele:

```
IN := 1;
OUT := NOT IN; (* OUT = 0 *)

IN := 43690;   (* 1010 1010 1010 1010 *)
OUT := NOT IN; (* 0101 0101 0101 0101; OUT = 21845 *)
```

8.6 Auswahl

Zulässige Datentypen

Argument: UINT, UDINT, REAL (bei SEL zusätzlich BOOL für den Selektor)

Ergebnis: UINT, UDINT, REAL

MAX

Gibt das größte Argument als Ergebnis zurück.

Beispiel:

```
OUT := MAX(8, 9, 10, 11, 12); (* OUT := 12 *)
```

MIN

Gibt das kleinste Argument als Ergebnis zurück.

Beispiel:

```
OUT := MIN(8, 9, 10, 11, 12); (* OUT := 8 *)
```

LIMIT

Prüft, ob ein Wert (IN) innerhalb eines bestimmten Bereichs (MIN, MAX) liegt. Liegt der Wert innerhalb des Bereichs, wird der Wert selbst zurückgegeben. Liegt der Wert unterhalb des Bereichs, wird die untere Bereichsgrenze zurückgegeben. Liegt der Wert oberhalb des Bereichs, wird die obere Bereichsgrenze zurückgegeben.

Beispiele:

```
OUT := LIMIT(IN, MIN, MAX);
OUT := LIMIT(15, 10, 20); (* OUT = 15 *)
OUT := LIMIT(5, 10, 20); (* OUT = 10 *)
OUT := LIMIT(25, 10, 20); (* OUT = 20 *)
```

SEL

Wählt in Abhängigkeit einer boolschen Variablen (G) einen von zwei Werten (IN0, IN1) aus. Der Datentyp der Werte IN0 und IN1 muss identisch sein.

Beispiele:

```
OUT := SEL(G, IN0, IN1);
OUT := SEL(G, 10, 20); (* G = FALSE: OUT = 10 *)
OUT := SEL(G, 10, 20); (* G = TRUE: OUT = 20 *)
```

8.7 Vergleich

Zulässige Datentypen

Argument bei LT, LE, GT, GE: UINT, UDINT, REAL, DT

Argument bei EQ und NE: BOOL, UINT, UDINT, REAL, DT

Ergebnis: BOOL

< (lower than)

Vergleicht Argumente darauf, ob eines kleiner ist als ein anderes.

Beispiele:

```
bOUT := IN1 < IN2; (* bOUT := TRUE, wenn IN1 kleiner IN2 ist *)
bOUT := (IN1 < IN2) & (IN2 > IN3) & ... & (INn-1 > INn);
```

<= (lower or equal)

Vergleicht Argumente darauf, ob eines kleiner oder gleich einem anderen ist.

Beispiele:

```
bOUT := IN1 <= IN2; (* bOUT := TRUE, wenn IN1 kleiner oder gleich IN2 ist *)
bOUT := (IN1 <= IN2) & (IN2 <= IN3) & ... & (INn-1 <= INn);
```

> (greater than)

Vergleicht Argumente darauf, ob eines größer als ein anderes ist.

Beispiele:

```
bOUT := IN1 > IN2; (* bOUT := TRUE, wenn IN1 größer IN2 ist *)
bOUT := (IN1 > IN2) & (IN2 > IN3) & ... & (INn-1 > INn);
```

>= (greater or equal)

Vergleicht Argumente darauf, ob eines größer oder gleich einem anderen ist.

Beispiel:

```
bOUT := IN1 >= IN2; (* bOUT := TRUE, wenn IN1 größer oder gleich IN2 ist *)
```

= (equal)

Vergleicht Argumente darauf, ob sie gleich sind.

Beispiel:

```
bOUT := IN1 = IN2; (* bOUT := TRUE, wenn IN1 gleich IN2 ist *)
```

<> (not equal)

Vergleicht Argumente darauf, ob sie ungleich sind.

Beispiel:

```
bOUT := IN1 <> IN2; (* bOUT := TRUE, wenn IN1 ungleich IN2 ist *)
```

8.8 Datum und Uhrzeit

Hierbei handelt es sich um spezifische Funktionen, die nicht Bestandteil der Norm DIN IEC 61131-3 sind.

Mit diesen Funktionen kann beispielsweise die Eingangsvariable rtc.cdt abgefragt werden, die Datum und Uhrzeit des Gerätes liefert.

Zulässige Datentypen

Argument: DT, DATE_AND_TIME

Ergebnis: UINT

GET_YEAR

Gibt das Jahr (mit Jahrhundert) zurück.

Beispiele:

```
iOUT := GET_YEAR(dt#2017-03-20-17:45:12); (* iOUT = 2017 *)
```

GET_MONTH

Gibt den Monat zurück.

Beispiel:

```
iOUT := GET_MONTH(dt#2017-03-20-17:45:12); (* iOUT = 03 *)
```

GET_DAY

Gibt den Tag zurück.

Beispiel:

```
iOUT := GET_DAY(dt#2017-03-20-17:45:12); (* iOUT = 20 *)
```

GET_HOUR

Gibt die Stunde zurück.

Beispiel:

```
iOUT := GET_HOUR(dt#2017-03-20-17:45:12); (* iOUT = 17 *)
```

GET_MINUTE

Gibt die Minute zurück.

Beispiel:

```
iOUT := GET_MINUTE(dt#2017-03-20-17:45:12); (* iOUT = 45 *)
```

GET_SECOND

Gibt die Sekunde zurück.

Beispiel:

```
iOUT := GET_SECOND(dt#2017-03-20-17:45:12); (* iOUT = 12 *)
iOUT := GET_SECOND(rtc_cdt); (* iOUT = 59, wenn rtc_cdt = 1970-01-01 00:00:59 *)
```

8 Funktionen

8.9 Weitere Funktionen

Hierbei handelt es sich um spezifische Funktionen, die nicht Bestandteil der Norm DIN IEC 61131-3 sind.

IS_VALID

Prüft die Gültigkeit einer REAL-Zahl. Das Ergebnis ist ein boolscher Wert (gültig = TRUE, ungültig = FALSE).

Werte ≥ 1.00E+37 (allgemeiner Fehlerwert) gelten als ungültig, ebenso die Werte „Unendlich“ (INF) und „Keine Zahl“ (NaN).

Beispiele:

```
bOUT := IS_VALID(1.2); (* bOUT =  TRUE *)
```

SET_DEFAULT

Setzt eine REAL- oder eine BOOL-Zahl auf den Default-Wert.

Default-Wert REAL: 5.0E+37 (Mathematik-Fehlerwert)
Default-Wert BOOL: FALSE

Beispiel:

```
SET_DEFAULT(bOUT); (* bOUT =  FALSE *)
SET_DEFAULT(rOUT); (* rOUT = 5.0E+37 *)
```

8.9.1 Prozessbild-Funktionen

Mit den folgenden Funktionen können bestimmte Eigenschaften von Prozessbildobjekten abgefragt und geändert werden (Farben, Sichtbarkeit, Editierbarkeit).

HINWEIS!

Im ST-Code beginnt die Nummerierung der Prozessbilder und Objekte mit 0 (z. B. Prozessbilder 0 bis 9, Objekte 0 bis 99). Im Setup-Programm beginnt die Nummerierung mit 1 (z. B. Prozessbilder 1 bis 10, Objekte 1 bis 100).

HINWEIS!

Anstelle der Nummern der Prozessbilder und Objekte können auch deren Definitionen verwendet werden.

Argumente der Funktionen

ProcPicIndex (UINT): Index (Nummer) des Prozessbildes (Index 0 = Prozessbild 1)

ObjectIndex (UINT): Index (Nummer) des Objektes im Prozessbild (Index 0 = Objekt 1)

ColorType (UINT):
- 0 = Vordergrundfarbe
- 1 = Hintergrundfarbe
- 2 = Farbe für EIN-Zustand (nur bei Objekt „Bargraph“)
- 3 = Farbe für AUS-Zustand (nur bei Objekt „Bargraph“)
- 4 = Farbe für LOW-Zustand (nur bei Objekten „Digitalsignal“ und „Eingabe Digitalwert“)
- 5 = Farbe für HIGH-Zustand (nur bei Objekten „Digitalsignal“ und „Eingabe Digitalwert“)

ValueRed (UINT): RGB-Wert des Rot-Anteils der Farbe

ValueGreen (UINT): RGB-Wert des Grün-Anteils der Farbe

ValueBlue (UINT): RGB-Wert des Blau-Anteils der Farbe

Value (BOOL): Sichtbarkeit oder Editierbarkeit (TRUE, FALSE)

SET_OBJECT_COLOR

Setzt die Farbe eines Objektes in einem Prozessbild.

Der Rückgabewert (BOOL) gibt an, ob die Funktion erfolgreich ausgeführt wurde (TRUE) oder ob ein Fehler aufgetreten ist (FALSE).

Reihenfolge der Argumente:
SET_OBJECT_COLOR (ProcPicIndex, ObjectIndex, ColorType, ValueRed, ValueGreen, ValueBlue)

Beispiel:

```
bOUT := SET_OBJECT_COLOR(0, 9, 0, 255, 0, 0);
(* Setzt die Vordergrundfarbe von Prozessbild 1, Objekt 10 auf Rot. *)
```

GET_OBJECT_COLOR_RED

Der Rückgabewert (UINT) liefert den RGB-Wert (UINT) des Rot-Anteils eines Objektes in einem Prozessbild.

Reihenfolge der Argumente:
GET_OBJECT_COLOR_RED (ProcPicIndex, ObjectIndex, ColorType)

Beispiel:

```
iOUT := GET_OBJECT_COLOR_RED(0, 9, 0);
(* Liefert den betreffenden Wert der Vordergrundfarbe von Prozessbild 1, Objekt 10. *)
```

GET_OBJECT_COLOR_GREEN

Der Rückgabewert (UINT) liefert den RGB-Wert (UINT) des Grün-Anteils eines Objektes in einem Prozessbild.

Reihenfolge der Argumente:
GET_OBJECT_COLOR_GREEN (ProcPicIndex, ObjectIndex, ColorType)

Beispiel:

```
iOUT := GET_OBJECT_COLOR_GREEN(0, 9, 1);
(* Liefert den betreffenden Wert der Hintergrundfarbe von Prozessbild 1, Objekt 10. *)
```

GET_OBJECT_COLOR_BLUE

Der Rückgabewert (UINT) liefert den RGB-Wert (UINT) des Blau-Anteils eines Objektes in einem Prozessbild.

Reihenfolge der Argumente:
GET_OBJECT_COLOR_BLUE (ProcPicIndex, ObjectIndex, ColorType)

Beispiel:

```
iOUT := GET_OBJECT_COLOR_BLUE(0, 9, 2);
(* Liefert den betreffenden Wert der Farbe bei EIN von Prozessbild 1, Objekt 10. *)
```

SET_OBJECT_VISIBLE

Setzt die Sichtbarkeit eines Objektes in einem Prozessbild.

Der Rückgabewert (BOOL) gibt an, ob die Funktion erfolgreich ausgeführt wurde (TRUE) oder ob ein Fehler aufgetreten ist (FALSE).

Reihenfolge der Argumente:
SET_OBJECT_VISIBLE (ProcPicIndex, ObjectIndex, Value)

Value (BOOL): FALSE = nicht sichtbar; TRUE = sichtbar

Beispiel:

```
bOUT := SET_ OBJECT_VISIBLE(1, 4, TRUE);
(* Setzt den Parameter „Sichtbar“ von Prozessbild 2, Objekt 5 auf TRUE. *)
```

GET_OBJECT_VISIBLE

Der Rückgabewert (UINT) liefert den Status der Sichtbarkeit eines Objektes in einem Prozessbild.

Reihenfolge der Argumente:
GET_OBJECT_VISIBLE (ProcPicIndex, ObjectIndex)

Beispiel:

```
bOUT := GET_ OBJECT_VISIBLE(1, 4);
(* Liefert den Status des Parameters „Sichtbar“ von Prozessbild 2, Objekt 5. *)
```

SET_OBJECT_EDITABLE

Setzt die Editierbarkeit eines Objektes in einem Prozessbild.

Der Rückgabewert (BOOL) gibt an, ob die Funktion erfolgreich ausgeführt wurde (TRUE) oder ob ein Fehler aufgetreten ist (FALSE).

Reihenfolge der Argumente:
SET_OBJECT_EDITABLE (ProcPicIndex, ObjectIndex, Value)

Beispiel:

```
bOUT := SET_ OBJECT_EDITABLE(9, 19, FALSE);
(* Setzt den Parameter „Editierbar“ von Prozessbild 10, Objekt 20 auf FALSE. *)
```

GET_OBJECT_EDITABLE

Der Rückgabewert (UINT) liefert den Status der Editierbarkeit eines Objektes in einem Prozessbild.

Reihenfolge der Argumente:
GET_OBJECT_EDITABLE (ProcPicIndex, ObjectIndex)

Beispiel:

```
bOUT := GET_ OBJECT_EDITABLE(9, 19);
(* Liefert den Status des Parameters „Sichtbar“ von Prozessbild 10, Objekt 20. *)
```

Instanz

Ein Funktionsbaustein – im ST-Editor als Funktionsblock bezeichnet – ist eine Programm-Organisationseinheit, die bei Ausführung einen oder mehrere Werte liefert (siehe auch Norm DIN IEC 61131-3). Ein Funktionsbaustein kann in mehreren Instanzen (Kopien) existieren. Jede dieser Instanzen besitzt einen zugehörigen Bezeichner. Die Instanzen eines Funktionsbausteins sind unabhängig voneinander.

Existiert ein Funktionsbaustein namens TON z. B. in 4 Instanzen, so werden die einzelnen Instanzen mit TON01, TON02, TON03, TON04 angesprochen.

Ein- und Ausgänge

Dem Funktionsbaustein werden Parameter (einer oder mehrere) als Eingangswerte übergeben.

Die Parameter folgen, nach der Angabe der Instanz, in runden Klammern. Die Reihenfolge der Parameter ist fest vorgegeben. Der Aufruf eines Funktionsbausteins wird, wie jede Anweisung, durch ein „;“ abgeschlossen.

Beispiel: Aufruf des des Funktionsbausteins TON mit der Instanz 01:

```
TON01(TRUE,5,1);
```

Die Ausgänge werden vom Funktionsbaustein gesetzt und können abgefragt und in beliebigen Operationen verwendet werden. Eine Zuweisung von Werten ist nicht möglich.

Beispiel: Abfragen des Ausgangs ET des Funktionsbausteins TON mit der Instanz 01:

```
OUT := TON01.ET;
```

Die Zustände der Ein- und Ausgänge bleiben bis zum nächsten Aufruf des Funktionsbausteins konstant.

Funktionsbausteine

Das ST-Modul des Bildschirmschreibers unterstützt folgende Funktionsbausteine:

Bezeichner	Instanzen	Funktion
CTUD	32 1 – 16: non retain; 17 – 32: retain	Software-Up-/Down-Zähler
TP	8 1 – 4: non retain; 5 – 8: retain	Pulsgeber
TON	8 1 – 4: non retain; 5 – 8: retain	Einschaltverzögerung
TOF	8 1 – 4: non retain; 5 – 8: retain	Ausschaltverzögerung
R_TRIG	8	Flankenerkennung steigend (0 -> 1)
F_TRIG	8	Flankenerkennung fallend (1 -> 0)
SR	8	Bistabiler Funktionsbaustein (vorrangig setzen)
RS	8	Bistabiler Funktionsbaustein (vorrangig rücksetzen)

9 Funktionsbausteine

9.1 Software-Up-/Down-Zähler

Der Zähler stellt folgende Funktionen bereit:

- Aufwärtszählen
- Abwärtszählen
- Rücksetzen des Zählerstands auf 0
- Setzen eines oberen Zählwertes
- Abfragen auf Überschreiten des oberen Zählwertes
- Abfragen auf Zählerstand 0
- Zählwert abfragen

Der Zähler zählt die positiven Flanken an den Eingängen CU (aufwärts zählen) und CD (abwärts zählen).

Aufruf

CTUD<Instanz> (CU, CD, R, LD, PV);

Eingänge

Die Parameter R, LD, CU und CD sind in der Tabelle entsprechend ihrer Priorität absteigend sortiert.

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
R	BOOL	TRUE = Zählerstand auf 0 setzen
LD	BOOL	TRUE = oberen Zählwert setzen
CU	BOOL	positive Flanke = aufwärts zählen
CD	BOOL	positive Flanke = abwärts zählen
PV	UINT	oberer Zählwert

Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
CV	UINT	OUT := CTUD<Instanz>.CV; (* OUT = Zählerstand *)
QU	BOOL	OUT := CTUD<Instanz>.QU; (* OUT = TRUE, wenn Zählerstand > oberer Zählwert *)
QD	BOOL	OUT := CTUD<Instanz>.QD; (* OUT = TRUE, wenn Zählerstand <= 0 *)

Bemerkung

Um mit LD den oberen Zählwert zu setzen, darf R nicht TRUE sein.

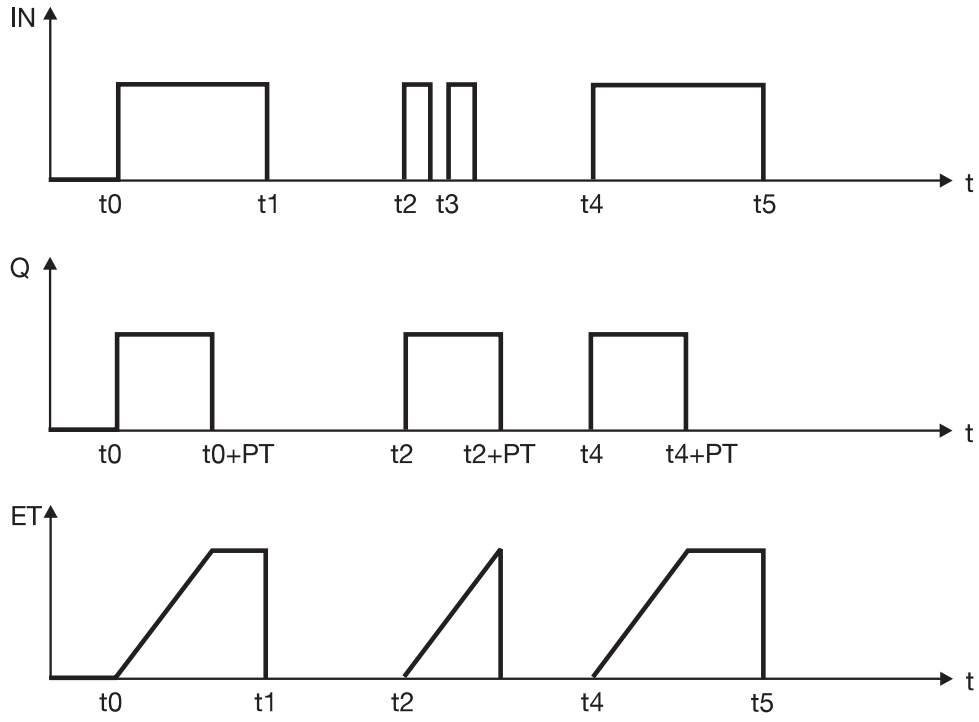
Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

Beispiele

```
CTUD01 (IN, FALSE, FALSE, FALSE, 100);
(* positive Flanken von IN aufwärts zählen *)

CTUD01 (FALSE, IN, FALSE, FALSE, 100);
(* positive Flanken von IN abwärts zählen *)

CTUD01 (IN, FALSE, TRUE, FALSE, 100);
(* Zählerstand auf 0 setzen *)

CTUD01 (IN, FALSE, FALSE, TRUE, 100);
(* oberen Zählwert setzen: 100 *)
```

9.2 Pulsgeber

Die Funktionsweise des Pulsgebers veranschaulicht folgendes Diagramm:



Aufruf

TP<Instanz> (IN, PT, TimeBase);

Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
IN	BOOL	positive Flanke an IN setzt Q für die Zeit PT auf TRUE
PT	UINT	Zeit (Einheit: TimeBase), für die Q = TRUE wird bei IN = TRUE
TimeBase	UINT	Einheit von PT: 1 = ms 2 = s 3 = min

Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
Q	BOOL	OUT := TP<Instanz>.Q; (* OUT = TRUE für die Zeit PT, wenn IN := 0 -> 1 *)
ET	UINT	OUT := TP<Instanz>.ET; (* OUT = vergangene Zeit seit IN = TRUE bis Q = FALSE (vergangene Zeit seit Beginn der aktiven Pulsphase) *)

Bemerkung

Der Parameter TimeBase ist eine Erweiterung zur DIN IEC 61131-3. Diese Erweiterung stellt keine Einschränkung gegenüber der Norm dar. Die Zeitvorgabe ist laut Norm „implementierungsabhängig“.

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

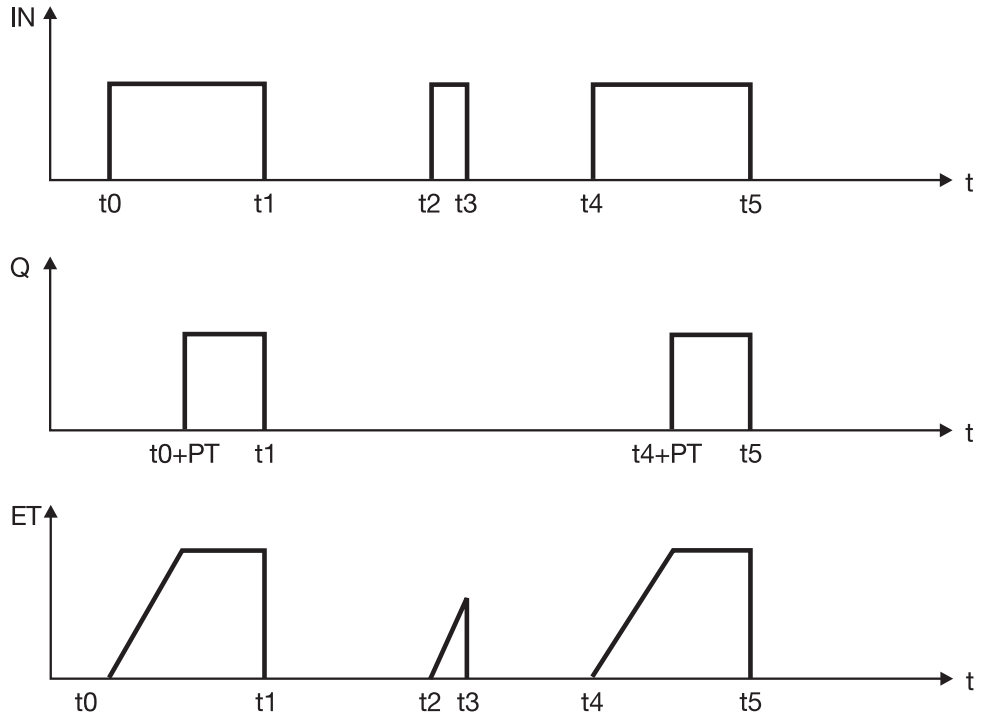
Die zeitliche Auflösung der Pulslänge hängt von der Zykluszeit des Gerätes ab (Beispiel: 150 ms).

Beispiel

```
TP01(IN, 5, 3);
(* Positive Flanke an IN setzt TP01.Q für ca. 5 Minuten auf TRUE (zeitliche Auflösung
beachten).
TP01.ET liefert die vergangene Zeit in Minuten seit Beginn der aktiven Pulsphase. *)
```

9.3 Einschaltverzögerung

Die Funktionsweise der Einschaltverzögerung veranschaulicht folgendes Diagramm:



Aufruf

TON<Instanz> (IN, PT, TimeBase);

Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
IN	BOOL	IN = TRUE setzt Q = TRUE, **nachdem** die Zeit PT abgelaufen ist
PT	UINT	Zeitverzögerung (Einheit: TimeBase)
TimeBase	UINT	Einheit von PT: 1 = ms 2 = s 3 = min

Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
Q	BOOL	OUT := TON<Instanz>.Q; (* OUT = TRUE, wenn IN = TRUE und PT abgelaufen *)
ET	UINT	OUT := TON<Instanz>.ET; (* OUT = vergangene Zeit seit IN = TRUE bis Q = TRUE oder bis IN = FALSE *)

Bemerkung

Der Parameter TimeBase ist eine Erweiterung zur DIN IEC 61131-3. Diese Erweiterung stellt keine Einschränkung gegenüber der Norm dar. Die Zeitvorgabe ist laut Norm „implementierungsabhängig“.

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

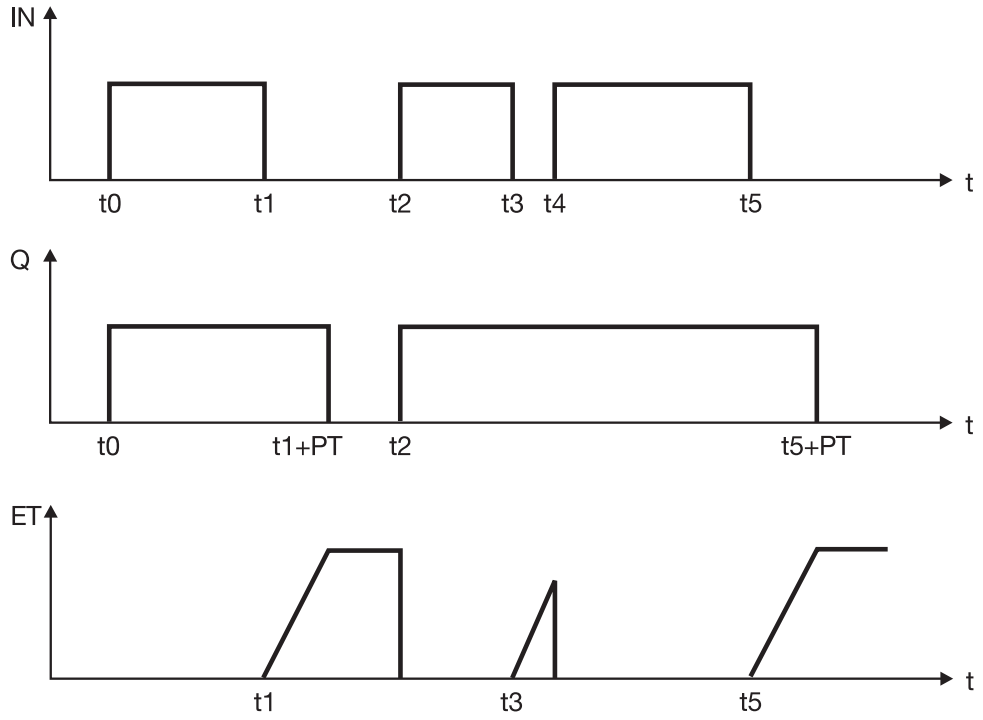
Die zeitliche Auflösung der Verzögerungszeit hängt von der Zykluszeit des Gerätes ab (Beispiel: 150 ms).

Beispiel

```
TON01(IN, 6, 2);

(* IN = TRUE setzt nach 6 Sekunden TON01.Q auf TRUE.
TON01.Q bleibt solange TRUE, wie IN = TRUE ist.
TON01.ET liefert die vergangene Zeit in Sekunden von IN = TRUE bis TON01.Q = TRUE (max.
6 Sekunden) oder bis IN = FALSE. *)
```

9.4 Ausschaltverzögerung

Die Funktionsweise der Ausschaltverzögerung veranschaulicht folgendes Diagramm:



Aufruf

TOF<Instanz> (IN, PT, TimeBase);

Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
IN	BOOL	IN = TRUE setzt Q = TRUE IN = FALSE setzt Q um die Zeit PT verzögert auf FALSE
PT	UINT	Zeitverzögerung (Einheit: TimeBase)
TimeBase	UINT	Einheit von PT: 1 = ms 2 = s 3 = min

Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
Q	BOOL	OUT := TOF<Instanz>.Q; (* OUT = TRUE, wenn IN = TRUE oder wenn IN = FALSE ist und PT noch nicht abgelaufen ist *)
ET	UINT	OUT := TOF<Instanz>.ET; (* OUT = vergangene Zeit seit IN = FALSE bis Q = FALSE oder bis IN = TRUE *)

Bemerkung

Der Parameter TimeBase ist eine Erweiterung zur DIN IEC 61131-3. Diese Erweiterung stellt keine Einschränkung gegenüber der Norm dar. Die Zeitvorgabe ist laut Norm „implementierungsabhängig“.

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

Die zeitliche Auflösung der Verzögerungszeit hängt von der Zykluszeit des Gerätes ab (Beispiel: 150 ms).

Beispiel

```
TOF01(IN, 450, 1);

(* IN = TRUE setzt TOF01.Q auf TRUE.
Nach IN = FALSE bleibt TOF01.Q TRUE, bis die Zeit PT = 450  ms abgelaufen ist.
TOF01.ET liefert die vergangene Zeit in Millisekunden von IN = FALSE bis TOF01.Q =
FALSE (max. 450 ms) oder bis IN = TRUE. *)
```

9.5 Flankenerkennung steigend

Dieser Funktionsbaustein erkennt eine steigende Flanke (Übergang 0 -> 1) .

Aufruf

R_TRIG<Instanz> (CLK);

Eingang

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
CLK	BOOL	Durch eine steigende Flanke (0 -> 1) an CLK wird Q = TRUE

Ausgang

Parameter	Datentyp	Abfrage
Q	BOOL	OUT := R_TRIG<Instanz>.Q;

Bemerkung

Die Funktionsweise des Funktionsbausteins entspricht folgendem Programmcode:

VAR CLK : BOOL; END_VAR

VAR Q : BOOL; END_VAR

VAR M : BOOL := FALSE; END_VAR

Q := CLK AND NOT M;

M := CLK;

Der Q-Ausgang bleibt von einem Aufruf zum nächsten auf seinem booleschen Wert. Er folgt dem Übergang des Eingangssignals (CLK) von „0“ nach „1“ und kehrt beim nächsten Aufruf nach „0“ zurück.

Beispiel

```
IF reset_flag THEN
bool_out01 := FALSE;
END_IF;

R_TRIG01 (bool_in01);
(* Erkennung einer steigenden Flanke an Eingang bool_in01 *)

bool_out01 := R_TRIG01.Q;
(* kurzer Impuls an Ausgang bool_out01 bei steigender Flanke am Eingang *)
```

9 Funktionsbausteine

9.6 Flankenerkennung fallend

Dieser Funktionsbaustein erkennt eine fallende Flanke (Übergang 1 -> 0) .

Aufruf

F_TRIG<Instanz> (CLK);

Eingang

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
CLK	BOOL	Durch eine fallende Flanke (1 -> 0) an CLK wird Q = TRUE

Ausgang

Parameter	Datentyp	Abfrage
Q	BOOL	OUT := F_TRIG<Instanz>.Q;

Bemerkung

Die Funktionsweise des Funktionsbausteins entspricht folgendem Programmcode:

VAR CLK : BOOL; END_VAR

VAR Q : BOOL; END_VAR

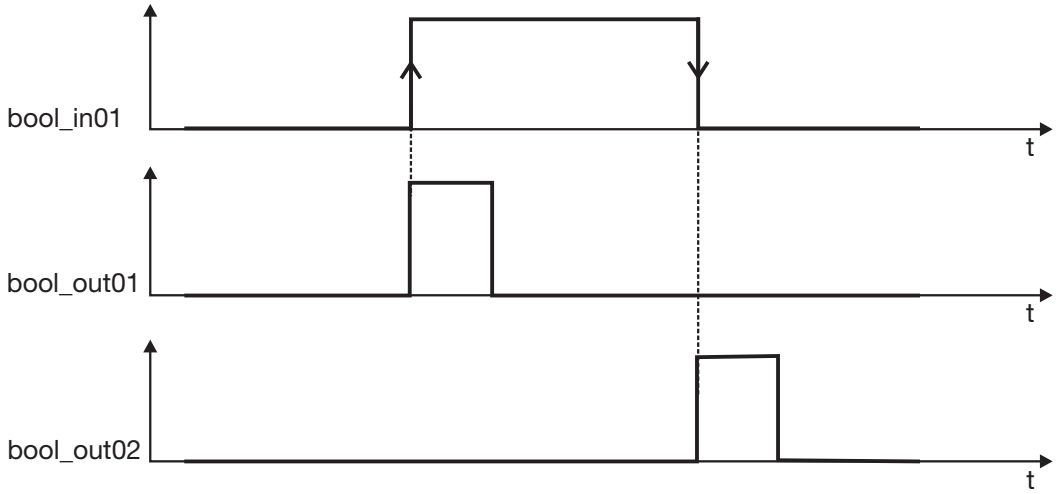
VAR M : BOOL := TRUE; END_VAR

Q := NOT CLK AND NOT M;

M := NOT CLK;

Der Q-Ausgang bleibt von einem Aufruf zum nächsten auf seinem booleschen Wert. Er folgt dem Übergang des Eingangssignals (CLK) von „1“ nach „0“ und kehrt beim nächsten Aufruf nach „0“ zurück.

Beispiel mti R_TRIG und F_TRIG




```
IF reset_flag THEN
bool_out01 := FALSE;
bool_out02 := FALSE;
END IF;

R_TRIG01 (bool_in01);
(* Erkennung einer steigenden Flanke an Eingang bool_in01 *)

bool_out01 := R_TRIG01.Q;
(* kurzer Impuls an Ausgang bool_out01 bei steigender Flanke am Eingang *)
```

```
F_TRIG01 (bool_in01);
(* Erkennung einer fallenden Flanke an Eingang bool_in01 *)
bool_out02 := F_TRIG01.Q;
(* kurzer Impuls an Ausgang bool_out02 bei fallender Flanke am Eingang *)
```

9 Funktionsbausteine

9.7 Bistabile Funktion SR

Bistabile Funktion (vorrangig Setzen) mit Selbsthaltung

Aufruf

SR<Instanz> (S1, R);

Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
S1	BOOL	S1 = TRUE setzt Q1 = TRUE
R	BOOL	R = TRUE setzt Q1 = FALSE, wenn S1 = FALSE

Ausgänge

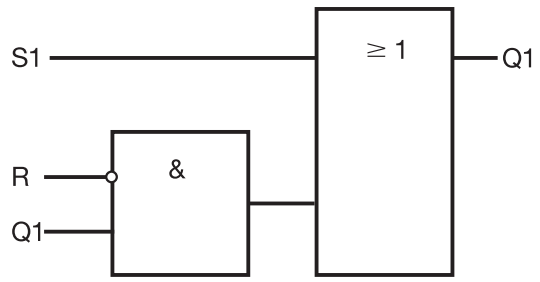
Parameter	Datentyp	Abfrage
Q1	BOOL	OUT := SR<Instanz>.Q1;

Bemerkung

Der Ausgang Q1 bleibt nach Setzen durch S1 so lange TRUE, bis er mit R zurückgesetzt wird.

Werden S1 und R gleichzeitig gesetzt (TRUE), wird der Ausgang Q1 ebenfalls gesetzt (TRUE).

Logikfunktion



Beispiel

```
SR01 (bool_in01, bool_in02);
(* Setzen mit Eingang bool_in01, Rücksetzen mit Eingang bool_in02 *)
bool_out01 := SR01.Q1;
(* Der Zustand wird über Ausgang bool_out01 ausgegeben. *)
```

9.8 Bistabile Funktion RS

Bistabile Funktion (vorrangig Rücksetzen) mit Selbsthaltung

Aufruf

RS<Instanz> (S, R1);

Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis xx (Instanz zweistellig angeben)
R1	BOOL	R1 = TRUE setzt Q1 = FALSE
S	BOOL	S = TRUE setzt Q1 = TRUE, wenn R1 = FALSE

Ausgänge

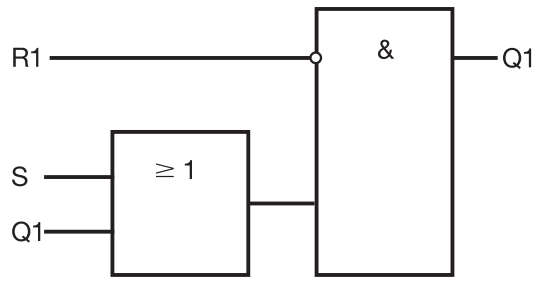
Parameter	Datentyp	Abfrage
Q1	BOOL	OUT := RS<Instanz>.Q1;

Bemerkung

Der Ausgang Q1 bleibt nach Rücksetzen durch R1 so lange FALSE, bis er mit S gesetzt wird.

Werden R1 und S gleichzeitig gesetzt (TRUE), wird der Ausgang Q1 zurückgesetzt (FALSE).

Logikfunktion



Beispiel

```
RS01 (bool_in01, bool_in02);
(* Rücksetzen mit Eingang bool_in02, Setzen mit Eingang bool_in01 *)
bool_out02 := RS01.Q1;
(* Der Zustand wird über Ausgang bool_out02 ausgegeben. *)
```

9 Funktionsbausteine

JUMO GmbH & Co. KG

Moritz-Juchheim-Straße 1
36039 Fulda, Germany

Telefon: +49 661 6003-727
Telefax: +49 661 6003-508
E-Mail: mail@jumo.net
Internet: www.jumo.net

Lieferadresse:
Mackenrodtstraße 14
36039 Fulda, Germany

Postadresse:
36035 Fulda, Germany

Technischer Support Deutschland:

Telefon: +49 661 6003-9135
Telefax: +49 661 6003-881899
E-Mail: service@jumo.net

JUMO Mess- und Regelgeräte GmbH

Pfarrgasse 48
1230 Wien, Austria

Telefon: +43 1 610610
Telefax: +43 1 6106140
E-Mail: info.at@jumo.net
Internet: www.jumo.at

Technischer Support Österreich:

Telefon: +43 1 610610
Telefax: +43 1 6106140
E-Mail: info.at@jumo.net

JUMO Mess- und Regeltechnik AG

Laubisrütistrasse 70
8712 Stäfa, Switzerland

Telefon: +41 44 928 24 44
Telefax: +41 44 928 24 48
E-Mail: info@jumo.ch
Internet: www.jumo.ch

Technischer Support Schweiz:

Telefon: +41 44 928 24 44
Telefax: +41 44 928 24 48
E-Mail: info@jumo.ch