

# ***TiCoBasic***

**Echtzeit-Entwicklungstool für  
*TiCo*-Prozessoren**

***TiCoBasic* Version 1.0**

**Juni 2010**

**License Key:**

*ADwin* - die schnellsten Echtzeitsysteme unter Windows

**Hier finden Sie immer einen Ansprechpartner für Ihre Fragen:**

Hotline: (0 62 51) 9 63 20  
Fax: (0 62 51) 5 68 19  
E-Mail: [info@ADwin.de](mailto:info@ADwin.de)  
Internet: [www.ADwin.de](http://www.ADwin.de)



Jäger Computergesteuerte  
Messtechnik GmbH  
Rheinstraße 2-4  
D-64653 Lorsch

## **Inhaltsverzeichnis**

Inhaltsverzeichnis .....	III
Konventionen .....	2
1 Einführung .....	3
2 TiCoBasic für ADbasic-Nutzer .....	5
3 Entwicklungsumgebung .....	9
3.1 Grundlegende Schritte .....	9
3.1.1 Entwicklungsumgebung erstmals starten .....	9
3.1.2 Lizenzen für TiCoBasic prüfen und ändern .....	10
3.1.3 Kommunikation initialisieren .....	11
3.1.4 Grundelemente der Entwicklungsumgebung .....	12
3.2 Quelltexte erstellen .....	16
3.2.1 Online-Hilfe aufrufen .....	17
3.2.2 Kontextmenü im Quelltextfenster .....	18
3.2.3 Editor-Leiste .....	20
3.3 Quelltext formatieren .....	21
3.3.1 Syntax hervorheben .....	21
3.3.2 Automatisch formatieren .....	21
3.3.3 Textzeilen einrücken .....	22
3.3.4 Zeilen in Kommentar ändern .....	22
3.3.5 Textbereiche ein- / ausfallen .....	23
3.4 Suchen, markieren und ersetzen .....	24
3.4.1 Text schnell suchen .....	24
3.4.2 Text suchen und ersetzen .....	25
Beispiele für das Suchen von Text .....	28
Beispiele für das Ersetzen von Text .....	29
3.4.3 Reguläre Ausdrücke .....	30
3.4.4 Kontrollstruktur markieren .....	32
3.4.5 Textmarken nutzen .....	33
3.4.6 Zu einer Programmzeile springen .....	33

3.4.7 Zur Deklaration eines Befehls oder Variablen springen	33
3.5 Programme leichter schreiben	35
3.5.1 Befehle und Variablen automatisch vervollständigen	35
3.5.2 Textbausteine einfügen	37
3.5.3 Befehlsparameter anzeigen	37
3.5.4 Deklaration eines Befehls oder Variablen anzeigen	38
3.5.5 Deklarationen einer Datei anzeigen	38
3.5.6 Verwendete globale Variablen und Felder anzeigen	39
3.6 TiCo-Binärdatei zum TiCo-Prozessor übertragen	39
3.6.1 TiCo-Binärdatei übertragen	39
3.6.2 TiCo-Bootloader programmieren	40
3.7 Projektverwaltung	41
3.8 Menüs	42
3.8.1 Das Menü „File“	43
3.8.2 Das Menü „Edit“	44
3.8.3 Das Menü „View“	44
3.8.4 Das Menü „Build“	45
3.8.5 Das Menü „Options“	46
Dialogfenster „Compiler Options“	46
Dialogfenster „Process Options“	49
Dialogfenster „Settings“	53
3.8.6 Das Menü „Tools“	57
3.8.7 Das Menü „Window“	57
3.8.8 Das Menü „Help“	58
3.9 Fenster	59
3.9.1 Toolbox	59
3.9.2 Projektfenster	59
3.9.3 Parameterfenster	61
3.9.4 Prozessfenster	62
3.9.5 Registerfenster	65
3.9.6 Statusleiste	66
3.10 Info-Bereich	66
3.10.1 Infofenster	67
3.10.2 ToDo-Liste	68
3.10.3 Fenster „Global Variables“	69
3.10.4 Fenster „Declarations“	70
3.11 ADtools	71

4 Prozesse programmieren	73
4.1 Programmaufbau	73
4.1.1 Die Programmabschnitte	75
4.1.2 Benutzerdefinierte Befehle und Variablen	75
4.2 Variablen und Felder	77
4.2.1 Übersicht	77
4.2.2 Datenstrukturen	77
4.2.3 Datentypen	78
4.2.4 Zahlenwerte eingeben	79
4.2.5 Globale Variablen (Parameter)	79
4.2.6 Globale Felder (Arrays)	80
4.2.7 System-Variablen	81
4.2.8 Lokale Variablen und Felder	82
4.3 Variablen und Felder – Details	83
4.3.1 Variablen und Felder im Datenspeicher	83
4.3.2 Speicherbereiche	84
4.3.3 Die Datenstruktur RingBuffer	85
4.4 Berechnungsausdrücke	93
4.4.1 Auswertung von Operatoren	93
4.5 Bedingungen, Schleifen und Module	94
4.5.1 Unterprogramm- und Funktions-Makros	95
4.5.2 Include-Dateien	95
4.5.3 Bibliotheken (Libraries)	96
5 Prozesse optimieren	99
5.1 Bearbeitungszeit messen	99
5.2 Verschiedene Tipps	100
5.2.1 Zugriff auf Hardware-Adressen	100
5.2.2 Konstanten anstelle von Variablen	100
5.2.3 Schnellere Messfunktion	101
5.2.4 Wartezeit genau einstellen	101
5.2.5 Wartezeiten nutzen	101
5.2.6 Optimierung des Speicherzugriffs	103
6 Prozesse im Betriebssystem	105
6.1 Prozessverwaltung	107
6.1.1 Zeitgesteuerter Prozess (Timer)	107

6.1.2 Extern gesteuerter Prozess (External) . . . . .	108
6.1.3 Prozess ohne Ansteuerung (None) . . . . .	109
6.2 Zeitverhalten von Prozessen . . . . .	109
6.2.1 Processdelay . . . . .	109
6.2.2 Auslastung des TiCo-Prozessors . . . . .	110
6.2.3 Verschiedene Betriebszustände im Betriebssystem . . . . .	111
6.3 Kommunikation . . . . .	112
6.3.1 Datenaustausch zwischen Prozessen . . . . .	112
6.3.2 Kommunikation zwischen PC und TiCo-Prozessor . . . . .	113
6.3.3 Kommunikation zwischen ADwin CPU und TiCo-Prozessor 114	
6.3.4 Die Device No . . . . .	115
7 Befehlsreferenz. . . . .	117
7.1 Befehlssyntax . . . . .	117
7.2 Basis-Befehlssatz <i>TiCoBasic</i> . . . . .	118
7.3 Gold II: TiCo-Prozessor . . . . .	197
7.4 Pro II: <i>TiCo</i> -Prozessor . . . . .	245
8 Was tun bei Problemen? . . . . .	295
Anhang . . . . .	A-1
A.1 Tastaturkürzel in <i>TiCoBasic</i> . . . . .	A-1
A.2 ASCII-Zeichensatz . . . . .	A-4
A.3 Lizenzvertrag . . . . .	A-5
A.4 Kommandozeilen-Aufruf . . . . .	A-9
A.5 Befehle für <i>ADwin-Gold II</i> . . . . .	A-18
A.6 Befehle für <i>ADwin-Pro</i> . . . . .	A-20
A.7 Index . . . . .	A-23
A.8 Befehle in diesem Handbuch . . . . .	A-37

## Liebe Leserin, lieber Leser!

*TiCoBasic* ist das Werkzeug, mit dem Sie *TiCo*-Prozessoren im *ADwin*-System für Ihre spezielle Mess-, Regel- oder Steuer-Aufgabe programmieren. Dieses Handbuch führt Sie in die Grundlagen der Programmierung von Echtzeit-Prozessen ein und soll Ihnen als Nachschlagewerk dienen. Nutzen Sie parallel auch die Online-Hilfe mit F1.

Die Entwicklungsumgebung wie auch die Sprache *TiCoBasic* sind eng verwandt mit *ADbasic*; ein Umstieg fällt daher leicht. Beachten Sie bitte Kapitel 2, in dem die Unterschiede zwischen *TiCoBasic* und *ADbasic* dargestellt sind.

Wenn Sie den *TiCo*-Prozessor und *TiCoBasic* erstmals verwenden, empfehlen wir den schnellen Einstieg mit Kapitel 2 und 4. Wir setzen voraus, dass Sie bereits über grundlegende Kenntnisse des Programmierens z.B. in Basic verfügen.

Kapitel 3 beschreibt die Entwicklungsumgebung und wird für alle Anwender empfohlen.

Sollten Sie Hinweise haben, wo wir unsere Dokumentation verbessern können, bitten wir um Ihre Rückmeldung. Sie helfen uns damit, Ihnen ein gut verständliches und leicht bedienbares Werkzeug an die Hand zu geben.

Wir wünschen Ihnen viel Erfolg beim Programmieren.

Für Rückfragen wenden Sie sich bitte an unseren Support (Adresse in der vorderen Innenseite des Handbuchs).

## Konventionen

Wir verwenden in diesem Handbuch die folgenden typographischen Konventionen und Zeichen:



Dieses Zeichen (Achtung) steht neben einem Absatz mit wichtigen Informationen für eine korrekte Funktion und fehlerfreien Betrieb.



Bei einem „Hinweis“ finden Sie Tipps und Ratschläge für einen effizienten Betrieb.



Das Informations-Zeichen verweist auf weiterführende Informationen im Handbuch oder in anderen Quellen (Dokumentation, Datenblätter, Literatur etc.).



Ein Beispiel verdeutlicht Ihnen, wie Sie das Gelesene einfach in die Praxis umsetzen können.

Am Schrifttyp „Courier“ erkennen Sie Texte, die auf dem Bildschirm erscheinen, z.B. in Fenstern oder Menüs, oder die Sie mit der Tastatur eingeben. In ähnlicher Weise sind die Namen von Menüs und Untermenüs dargestellt: „Menü ► Untermenü“.

Dateinamen und Pfadnamen sind zusätzlich in spitze Klammern gesetzt: `<path\xx.ext>`.

Elemente eines Quelltextes wie **Befehle**, *Variablen*, *Kommentar* und *sonstiger Text* werden ähnlich dargestellt wie in der Entwicklungsumgebung.

Tastenbezeichnungen werden in eckigen Klammern und in Kapitälchen angegeben wie [RETURN] oder [CTRL].

Die Bits eines Datenwortes (hier: 16 Bit) werden wie folgt nummeriert:

Bit-Nr.	15	14	13	...	01	00
Wert des Bits	$2^{15}$	$2^{14}$	$2^{13}$	...	$2^1=2$	$2^0=1$
Bezeichnung	MSB	-	-	-	-	LSB

Wenn Zahlen nicht dezimal angegeben werden, erhalten sie als Anhang einen kennzeichnenden Buchstaben; z. B. für die Zahl 17:

- hexadezimale Schreibweise: **11h**
- binäre Schreibweise: **10001b**



## 1 Einführung

Das ADwin-Echtzeitsystem übernimmt alle zeitkritischen Aufgaben in Ihren schnellen dynamischen Prüfständen und Fertigungsanlagen. In diesem Rahmen ist der – im Echtzeitsystem integrierte – *TiCo*-Prozessor dort vorgesehen, wo besonders exaktes und fein einstellbares Timing gefordert ist.

Während Sie das ADwin-Echtzeitsystem in *ADbasic* programmieren, verwenden Sie für den *TiCo*-Prozessor die Sprache und Bedienoberfläche *TiCo-Basic*.

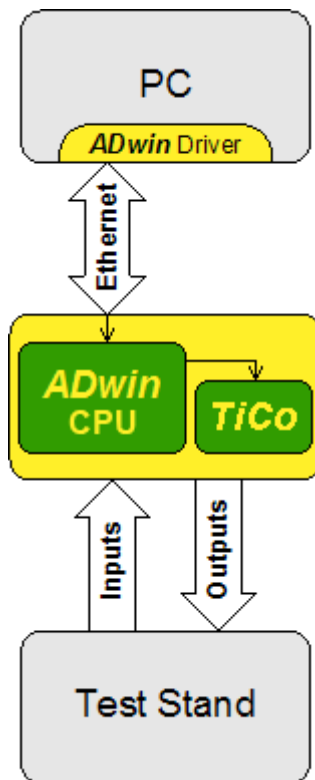
Damit Sie schnell und effizient mit der Programmierung beginnen können, möchten wir Ihnen zunächst das Konzept des ADwin-Systems mit *TiCo*-Prozessor erklären:

Jedes ADwin-System besitzt als zentrale Einheit die ADwin CPU, die zeitkritische Aufgaben in Echtzeit ausführt. Daneben kann ein ADwin-System (*Pro II* und *Gold II*) einen oder auch mehrere *TiCo*-Prozessoren besitzen. Analoge und digitale Ein- und Ausgänge sowie Erweiterungen wie Zähler und Bussysteme bilden die Verbindung zu Ihrem Prüfstand.

Im ADwin-System arbeitet der *TiCo*-Prozessor (*Timing Controller*) als eigenständiger, frei programmierbarer Zusatzprozessor, der auf Ein- und Ausgänge zugreift und dadurch Sonderfunktionen wie Filterung, Umrechnung, Kommunikationsprotokoll (SPI), Signalgenerator, Regelung etc. ausführen kann. Je nach Zielsetzung kann der *TiCo*-Prozessor der ADwin CPU zuarbeiten, indem er z.B. Daten vorverarbeitet; oder er übernimmt eine vollkommen eigenständige Aufgabe.

Der *TiCo*-Prozessor ist für schnelle Reaktionszeiten und exaktes Timing im Nanosekundenraster optimiert.

Die Kommunikation zwischen PC und ADwin CPU findet direkt über Ethernet statt. Der *TiCo*-Prozessor dagegen ist als Softcore im FPGA implementiert, zu dem der PC keine direkte Verbindung hat. Der Datenaustausch zwischen PC



und *TiCo*-Prozessor läuft daher immer über die *ADwin* CPU als Zwischenstation.

Der *TiCo*-Prozessor wird mit dem Echtzeit-Entwicklungs-Tool *TiCoBasic* programmiert, das die einfache und schnelle Erstellung von zeitkritischen Echtzeit-Prozessen ermöglicht. *TiCoBasic* ist eine integrierte Entwicklungsumgebung unter Windows. Die Befehle entsprechen weitgehend denen von *ADbasic* und erlauben Ihnen, auf Ein- und Ausgänge zuzugreifen, Echtzeitprozesse zu steuern und den Datenaustausch mit der *ADwin* CPU vorzubereiten. In Kapitel 4 ist erklärt, wie Sie *TiCoBasic*-Programme aufbauen, in Kapitel 6 außerdem der Ablauf Ihrer Prozesse im Betriebssystem.



Mit nur wenigen Programmzeilen können Sie beispielsweise:

- Messgrößen bis zu Abtastfrequenzen von 800kHz erfassen
- schnelle digitale Regler mit Abtastraten bis zu 400kHz entwickeln
- gleichzeitig analoge Signale erzeugen *und* messen, z.B. für dynamische Kennlinien-Messungen

Die Entwicklungsumgebung *TiCoBasic* unterstützt Sie bei der Umsetzung Ihrer Aufgabe in einen Prozess. Zunächst erstellen Sie den Quelltext in einer erweiterten Basic-Syntax; mit den Befehlen und Funktionen können Sie die Hardware Ihres *ADwin*-Systems komfortabel programmieren. In Kapitel 4 ist erklärt, wie Sie Programme aufbauen.



Mit dem integrierten Compiler erzeugen Sie aus dem Quelltext lauffähigen Binärcode, der als Prozess auf das *ADwin*-System übertragen und getestet wird. *TiCoBasic* bietet Ihnen auch die Hilfsmittel, mit denen Sie Ihre Prozesse



beobachten, Fehler suchen und die Programme optimieren können (siehe Kapitel 2).

Sobald die Echtzeit-Prozesse zu Ihrer Zufriedenheit laufen, ist Ihre Arbeit mit *TiCoBasic* bereits beendet.

Sie können die Prozesse und Prozessdaten des *TiCo*-Prozessors von der *ADwin* CPU aus steuern und beobachten, d.h. Daten lesen und schreiben sowie Prozesse starten, steuern und stoppen.

Obwohl der *TiCo*-Prozessor autark arbeitet, können Sie aus von der *ADwin* CPU jederzeit auf globale Variablen und Felder zugreifen, ohne zeitkritische Prozesse zu verzögern. Über die globalen Variablen und Felder können alle Prozesse untereinander oder mit der *ADwin* CPU schnell Daten austauschen.

Die klare Trennung von Echtzeit-Prozessen im *TiCo*-Prozessor und im *ADwin*-System einerseits und der Bedienoberfläche im PC andererseits garantiert Ihnen höchste Betriebssicherheit und zeitlich nachvollziehbare Abläufe.

## 2 TiCoBasic für ADbasic-Nutzer


Mit *TiCoBasic* arbeiten Sie wie mit *ADbasic 5*: Sie werden viele Funktionen und Arbeitsabläufe in gewohnter Weise nutzen können, Sie finden Menüeinträge und Schaltflächen an der gleichen Stelle. Das erleichtert Ihnen den Einstieg in *TiCoBasic*.


Es gibt aber auch einige wichtige Unterschiede, die Sie im Folgenden kennen lernen. So ist der *TiCo*-Prozessor keine zweite *ADwin* CPU im Kleinformaat, sondern er ist für schnelle Reaktionszeiten und exaktes Timing im Nanosekundenraster optimiert.


### Kommunikation mit dem *TiCo*-Prozessor

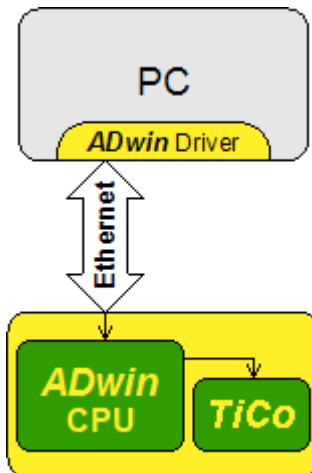
Der *TiCo*-Prozessor ist als Softcore im FPGA implementiert, zu dem der PC keine direkte Verbindung hat. Der Datenaustausch zwischen PC und *TiCo*-Prozessor läuft daher immer über die *ADwin* CPU als Zwischenstation.

Beachten Sie:

- Sie stellen unter `Options > Compiler` neben *ADwin*-System und Gerätenummer (Device No.) auch die *ADwin* CPU und den *TiCo*-Prozessor ein.
- Initialisieren statt Booten: Die Schaltfläche  initialisiert die gewählte *ADwin* CPU für den Datenaustausch mit dem *TiCo*-Prozessor.

Der *TiCo*-Prozessor selbst wird erst initialisiert, wenn ein Prozess kompiliert  wird. Dabei werden alle globalen Variablen auf 0 gesetzt und der kompilierte Prozess startet automatisch.

- Der *TiCo*-Prozessor läuft weiter, auch wenn die *ADwin* CPU nicht verfügbar ist, z. B. weil er gebootet wurde. Um den Datenaustausch wieder einzurichten, genügt es dann, die *ADwin* CPU mit der Schaltfläche  wieder zu initialisieren.



## TiCo-Prozesse wie gewohnt

TiCo-Prozesse werden grundsätzlich programmiert wie ADbasic-Prozesse.

- Es gibt folgende Anzahl und Typen von Prozessen:

Prozesstyp	Priorität hoch	Priorität niedrig
zeitgesteuert	1	1
extern gesteuert	1	–
nicht gesteuert	1	–

Bitte beachten Sie: Es sollte nur *ein* Prozess mit hoher Priorität laufen.

- Es gibt die 3 Programmabschnitte: **Init:**, **Event:** und **Finish:**, alle Programmabschnitte haben gleiche Priorität. Es gibt keinen Programmabschnitt **LowInit:**.
- Ein Taktzyklus des Prozessors dauert 20ns.

## Vereinfachter Befehlssatz

- Der TiCo-Prozessor verarbeitet ausschließlich den Datentyp **Long**; der Datentyp **Float** steht nicht zur Verfügung.

Die folgenden Befehle gehören zum Basisbefehlssatz. Darüber hinaus gibt es Befehle zum Zugriff auf die Ein-/Ausgänge und Schnittstellen der ADwin-Hardware, die in einem separaten Dokument beschrieben sind.

Standard	<b>Init:</b> , <b>Event:</b> , <b>Finish:</b> , <i>REM</i> , : (Doppelpunkt)
Variablen	<b>DIM</b> , <b>PAR_1...PAR_80</b> , <b>DATA_1...DATA_16</b> <i>Ringbuffer_For_Read</i> <i>Ringbuffer_For_Write</i> Lokale Felder
Mathematik, Operatoren	+ - * / <b>INC</b> , <b>DEC</b> , <b>SHIFT_LEFT</b> , <b>SHIFT_RIGHT</b> <b>ABSI</b> , <b>NOT</b> , <b>OR</b> , <b>XOR</b> , <b>AND</b>
Vergleichen	= < > <b>OR</b> , <b>AND</b>
Struktur	<b>FOR...NEXT</b> , <b>DO...UNTIL</b> <b>IF...THEN</b> , <b>SELECTCASE</b> <b>FUNCTION</b> , <b>SUB</b> , <i>Lib_FUNCTION</i> , <i>Lib_SUB</i>

Prozesse, Systemvariablen	<code>END</code> <code>Processdelay</code> , <code>PROZESS_RUNNING</code> , <code>NWTIME</code>
Pre-Compiler	<code>#IF...#ENDIF</code> , <code>#DEFINE</code> , <code>#INCLUDE</code>
Zeitsteuerung	<code>NOP</code> , <code>NOPS</code> , <code>SLEEP</code> , <code>READ_TIMER</code>
I/O-Zugriff	<code>IN</code> , <code>OUT</code>

Fließkomma-Berechnungen wie Integer-Potenzen, trigonometrische Funktionen und Division mit Rest sind nicht möglich.

- Es können maximal 16 globale Felder `DATA_1...DATA_16` deklariert werden (in ADbasic bis `DATA_200`). Der Datentyp FIFO ist nicht vorhanden.
- Es gibt die neuen Datentypen `Ringbuffer_For_Read` und `Ringbuffer_For_Write` (Ringspeicher).

Ringspeicher werden für schnelle Datenübertragung eingesetzt; die möglichen Anwendungen schließen sich gegenseitig aus:

- Der TiCo-Prozess greift auf Daten im externen DRAM zu und zwar lesend wie schreibend über den gleichen Ringspeicher.
- ADbasic-Prozesse (auf der ADwin CPU) und TiCoBasic-Prozesse tauschen über einen Ringspeicher miteinander Daten aus. Für jede Übertragungsrichtung ist ein separater Ringspeicher erforderlich.
- Mehrere Prozesse auf dem TiCo-Prozessor tauschen über einen Ringspeicher miteinander Daten aus. Es sind 2 Ringspeicher erforderlich, einer zum Schreiben und einer zum Lesen.



Der Umgang mit dem Datentyp `Ringbuffer` ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung des Datentyps `Ringbuffer` ist daher erfahrenen Benutzern von ADbasic und TiCoBasic vorbehalten.

- Der Befehl `EXIT` wird durch `END` ersetzt.
- Die Befehle `In` und `Out` ersetzen `Peek` und `Poke`.

## ADbasic-Befehle zur TiCo-Steuerung

Die ADwin CPU kann direkt auf Daten des TiCo-Prozessors zugreifen. Hierzu stehen Ihnen eine Reihe von ADbasic-Befehlen zur Verfügung, mit denen Sie Daten austauschen und Prozesse steuern können, siehe Kapitel 7.3 und 7.4.

### **Umstellungen in der Oberfläche**

- Derzeit unterstützt die Oberfläche *TiCoBasic* weder Debug- noch Timing-Modus, die in *ADbasic* zur Verfügung stehen.
- Wenn mehrere Dateien zu einem Projekt gehören, werden sie in jedem Fall gemeinsam kompiliert. Eine einzelne Datei zu kompilieren ist daher nur möglich, wenn sie nicht zu einem Projekt gehört.
- In Zukunft wird es möglich sein, in *TiCoBasic* Assembler-Befehle einzufügen. Hierzu wurde das Registerfenster in die Oberfläche eingefügt, das die Registerwerte des *TiCo*-Prozessors anzeigt.

### 3 Entwicklungsumgebung

Die Entwicklungsumgebung *TiCoBasic* dient zum Entwickeln von Programmen für *TiCo*-Prozessoren. Der *TiCoBasic*-Compiler verarbeitet eine erweiterte Basic-Syntax wie *ADbasic*, jedoch mit etwas geringerem Sprachumfang.

Das fertig entwickelte *TiCoBasic*-Programm wird in eine Binärdatei kompiliert und – indirekt über die *ADwin* CPU – auf den *TiCo*-Prozessor übertragen. Im Bootloader-Modus wird das Programm nun automatisch und unabhängig von der Entwicklungsumgebung ausgeführt.

#### 3.1 Grundlegende Schritte

##### 3.1.1 Entwicklungsumgebung erstmals starten

Zum Starten gehen Sie vor wie folgt:

1. Sie starten die Entwicklungsumgebung, indem Sie im Windows-Startmenü `Programs ▶ ADwin ▶ TiCoBasic` wählen.

Beim ersten Starten kann es einige Sekunden dauern, bis die Oberfläche erscheint, weil dabei auch das Windows-Programmpaket .Net-Framework gestartet wird.

Sie sehen nun die Entwicklungsumgebung mit den Windows-typischen Elementen wie Fenster, Menü- und Werkzeug-Leiste.

2. Beim ersten Start erscheint das Fenster `License key`. Geben Sie hier Ihren Lizenzschlüssel ein. Sie finden den `License key` auf dem Deckblatt dieses *TiCoBasic*-Handbuchs.

Ohne Eingabe des gültigen `License key` befindet sich *TiCoBasic* im Demo-Modus. In diesem Modus ist das Arbeiten mit der Entwicklungsumgebung nur zu Prüf-, Demonstrations- und Bewertungszwecken erlaubt. Sie können beispielsweise keine Binärdateien erzeugen.

Weitere Informationen zur *TiCoBasic*-Lizenz finden Sie im Kapitel 3.1.2 auf Seite 10.

3. Stellen Sie im Menü `Options\Compiler` ein, wo sich der *TiCo*-Prozessor befindet, den Sie als Nächstes programmieren:
  - den Typ des *ADwin*-Systems und der *ADwin* CPU,
  - die Gerätenummer (Device No.)
  - bei einem Pro II-System auch das Modul mit dem *TiCo*-Prozessor.

Die Entwicklungsumgebung speichert Ihre Angaben, so dass Sie sie bei einem erneuten Start von *TiCoBasic* nicht mehr eingeben müssen – es sei denn, Sie verwenden einen anderen *TiCo*-Prozessor.

### 3.1.2 Lizenzen für *TiCoBasic* prüfen und ändern

Um den Lizenzschlüssel für *TiCoBasic* zu prüfen oder zu ändern, gehen Sie vor wie folgt:

1. Wählen Sie den Menüeintrag **Help ► About**.

Das Fenster **About TiCoBasic** öffnet sich. Dort sind neben der Version der Entwicklungsumgebung unter **Licenses** die aktuellen Lizenzen angegeben (Liste möglicher Lizenzen siehe unten).



2. Zum Eingeben oder Ändern der Lizenzen klicken Sie die Schaltfläche **Change License**.

Das Eingabefenster **License key** öffnet sich.

3. Geben Sie den Lizenzschlüssel ein.



Sie finden den **License key** auf dem Deckblatt dieses *TiCoBasic*-Handbuchs.

Für *TiCoBasic* sind folgende Lizenzen möglich:

- Keine Lizenz (Demo mode)

Ohne Eingabe des gültigen **License key** befindet sich *TiCoBasic* im Demo-Modus. In diesem Modus ist das Arbeiten mit der Entwicklungs-



umgebung nur zu Prüf-, Demonstrations- und Bewertungszwecken erlaubt. Sie können beispielsweise keine Binärdateien erzeugen.

- Zeitlich begrenzte Lizenz (Evaluation license)

Die Lizenz schaltet alle Funktionen der Entwicklungsumgebung für einen definierten Zeitraum frei. Anschließend arbeitet *TiCoBasic* wieder im Demo-Modus (siehe oben).

- Zeitlich unbegrenzte Lizenz für den Lizenznehmer

Folgende Lizenzen können freigeschaltet werden:

- *ADbasic* 5, arbeitet mit allen *ADwin*-Prozessoren
- *ADbasic* 3.0, arbeitet mit *ADwin*-Prozessoren bis Version T9
- *ADbasic* 2.0, arbeitet mit *ADwin*-Prozessoren bis Version T8
- *TiCoBasic*
- *ADlab* (Matlab-Treiber für *ADwin*)

Die Lizenzen für *TiCoBasic* und *ADlab* können mit einer der *ADbasic*-Lizenzen kombiniert werden.

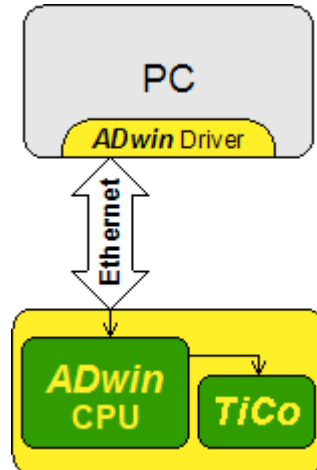
Die Lizenzbedingungen für *TiCoBasic* sind im Lizenzvertrag (Anhang Seite A-5) beschrieben.

### 3.1.3 Kommunikation initialisieren

Der *TiCo*-Prozessor ist als Softcore im FPGA implementiert, zu dem der PC keine direkte Verbindung hat. Der Datenaustausch zwischen PC und *TiCo*-Prozessor läuft daher immer über die *ADwin* CPU als Zwischenstation.

Sie initialisieren die gewählte *ADwin* CPU für den Datenaustausch mit dem *TiCo*-Prozessor, indem Sie auf die Schaltfläche **I** klicken (= initialisieren). Dadurch wird ein Prozess in der *ADwin* CPU eingerichtet, der Daten zwischen PC und *TiCo*-Prozessor austauscht.

Der *TiCo*-Prozessor selbst wird erst initialisiert, wenn Sie einen Prozess kompilieren **C**. Dabei werden die Inhalte des Programm- und Datenspeichers überschrieben, alle globalen *TiCo*-Variablen auf 0 gesetzt und der kompilierte Prozess startet automatisch.




Wenn kein *TiCo*-Prozessor vorhanden ist, erhalten Sie eine entsprechende Fehlermeldung.

Der *TiCo*-Prozessor läuft weiter, auch wenn die *ADwin* CPU nicht verfügbar ist, z.B. weil sie gebootet wurde. Um den Datenaustausch wieder einzurichten, genügt es dann, die *ADwin* CPU mit der Schaltfläche **I** (Initialize) wieder zu initialisieren.

Sie können den *TiCo*-Prozessor auch mit der Schaltfläche **R** (Reset) stoppen und zurücksetzen. Dadurch gehen alle Werte in den globalen *TiCo*-Variablen verloren und der Prozess wird gelöscht.

### 3.1.4 Grundelemente der Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus mehreren Leisten und Fenstern (siehe Abb. 1); Sie können die Höhe der Fenster frei anpassen.

Sie erhalten Hilfe zu einem Fenster oder zu dem aktuell markierten Kennwort, wenn Sie die Taste [F1] drücken. Mit der Schaltfläche  öffnen Sie das Indexverzeichnis der Hilfe.

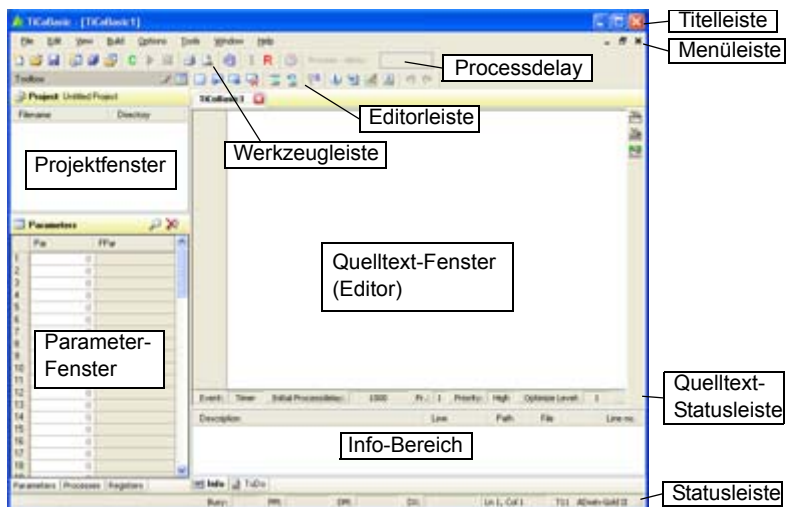


Abb. 1 – Elemente der *TiCoBasic*-Entwicklungsumgebung

Sie finden die Befehle der Entwicklungsumgebung über:

- die Werkzeugleiste und die Editorleiste (siehe Abb. 2).
- die Kontextmenüs der Fenster (rechte Maustaste).
- die Menüleiste.

Wenn Sie einen Befehl anwenden, sehen Sie am linken Rand der Statusleiste eine Befehlserklärung.

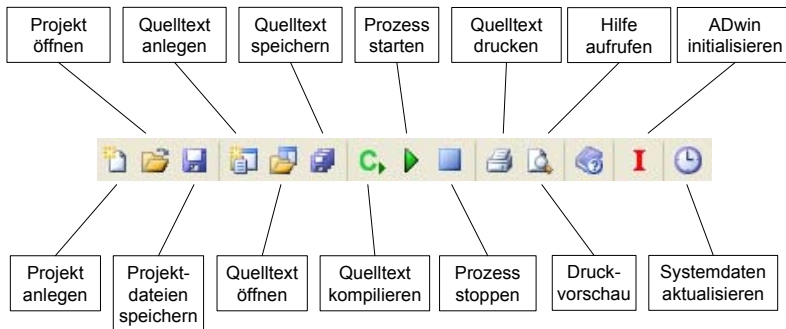



Abb. 2 – Die Werkzeugleiste

Sie wählen ein Menü (und dann einen Befehl) aus, indem Sie das Menüfeld mit der linken Maustaste anklicken, oder indem Sie die Tastenkombination [ALT] + [ANFANGSBUCHSTABE] des entsprechenden Menüs eingeben. Viele Befehle besitzen auch Tastatur-Kürzel (Short-Cuts, siehe Anhang A.1), die in den Menüs angezeigt werden.

Im aktiven Quelltext-Fenster (Editor) bearbeiten Sie den Quelltext für je einen Prozess. Sie können mehrere Fenster parallel geöffnet haben; die Fenstergrößen sind frei einstellbar. Weitere Informationen zum jeweiligen Quelltextfenster werden an verschiedenen Stellen angezeigt.

- Die Titelleiste zeigt den Namen des aktiven Quelltext-Fensters an.
- Die Quelltext-Statusleiste zeigt die von Ihnen eingestellten Prozess-Optionen.

Ein Doppelklick auf die Quelltext-Statusleiste öffnet das Dialogfenster „Process Options“.

- Im Parameterfenster (siehe Kapitel 3.9.3 auf Seite 61) können Sie durch Drücken der Schaltfläche *Scan Global Variables*  (einmalig) markieren lassen, welche globalen Parameter Sie im aktiven

Quelltext oder Projekt verwenden; siehe Verwendete globale Variablen und Felder anzeigen auf Seite 39.

- Im Infobereich unten werden in verschiedenen Fenstern eine Reihe von Informationen angezeigt:
  - Infofenster: Fehlermeldungen (rot hinterlegt) und Warnungen des Compilers (siehe Kapitel 3.10.1 auf Seite 67).
  - ToDo-Liste: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.10.2 auf Seite 68).
  - Suchergebnisse bei einer Suche über alle Dateien eines Projekts (siehe Kapitel 3.4.2 auf Seite 25).
  - Fenster „Global Variables“: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.10.2 auf Seite 68).
  - Fenster „Declarations“: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.10.2 auf Seite 68).

Beachten Sie, dass das Editieren im Quelltextfenster durch eine Reihe von Hilfsfunktionen unterstützt wird (siehe Seite 16).

Im Projektfenster werden der Name des offenen Projekts und die zugehörigen Dateien angezeigt, anderenfalls bleibt es leer.

Einige Daten des *TiCo*-Prozessors werden kontinuierlich ausgelesen und angezeigt (nur, wenn der PC mit dem *TiCo*-Prozessor kommuniziert):

- Das Processdelay (Prozess-Zykluszeit) zu der Prozessnummer, die Sie für das aktive Quelltextfenster eingestellt haben, dargestellt rechts von der Werkzeugleiste.
- Die Werte der globalen Variablen im Parameterfenster; wenn Sie einen dieser Werte verändern, wird er sofort zum *ADwin*-System übertragen.
- Der Zustand der laufenden Prozesse im Prozessfenster (Seite 62).
- Der Registerwerte des *TiCo*-Prozessor im Registerfenster (Seite 65).
- Informationen zur Speicherauslastung in der Statusleiste (siehe Kapitel 3.9.6).

## 3.2 Quelltexte erstellen

Öffnen Sie für jeden Prozess-Quelltext ein eigenes Fenster (mit **File ▶ New**). Wenn Sie für eine Aufgabe mehrere Dateien verwenden, empfehlen wir, diese gemeinsam in einer Projektdatei zu verwalten (siehe Seite 41: Projektverwaltung).

Der Editor und der *TiCoBasic*-Compiler unterscheiden nicht zwischen Groß- und Kleinschreibung. In unseren Beispielen verwenden wir allerdings zur besseren Unterscheidung eine einheitliche Schreibweise.

Bei Unklarheiten oder Problemen sollten Sie die Online-Hilfe aufrufen (siehe unten).

Beim Editieren im Quelltextfenster können Sie eine Reihe von Hilfsfunktionen nutzen. Sie rufen die Funktionen über das Kontextmenü im Quelltextfenster (Seite 18) oder über die Editor-Leiste (Seite 20) auf:

Sie können Zahlenwerte im Quelltext nicht nur dezimal, sondern auch in hexadezimaler, binärer und Exponential-Schreibweise eingeben (siehe Kapitel 4.2.4 „Zahlenwerte eingeben“).

Weitere Editor-Funktionen finden Sie unter den Rubriken:

- Quelltext formatieren, Seite 21
- Suchen, markieren und ersetzen, Seite 24
- Programme leichter schreiben, Seite 35

## 3.2.1 Online-Hilfe aufrufen

Das Menü „Help“ (Seite 58) erlaubt den Aufruf von ausgesuchten Hilfe-Seiten, z.B. das Inhaltsverzeichnis oder sortierte Befehlslisten.













Mit [F1] öffnen Sie eine Hilfeseite zu dem gerade geöffneten Dialogfenster oder zu dem Befehl an der Cursor-Position.

Wenn an der Cursor-Position ein ungültiger Befehl steht, öffnet die Hilfe das Indexverzeichnis. Gründe hierfür sind in der Regel:

- Es handelt sich nicht um einen Befehl, sondern um eine vom Benutzer definierte Deklaration: Variable / Feld, symbolischer Name, Makro (Sub, Function). Hierzu kann es keine Hilfeseiten geben.
- Der Befehl ist falsch geschrieben, z.B. `Digin_Wrod` anstatt **`Digin_Word`**.  
Wenn Sie den Fehler korrigiert haben, wird der Befehl wieder farbig hervorgehoben.
- Stellen Sie die passende *ADwin*-Hardware ein oder verwenden Sie einen für eingestellte Hardware gültigen Befehl.
- Im Quelltext fehlt die (benutzerdefinierte) Include- oder Library-Datei, in der der Befehl definiert ist.  
Fügen Sie die entsprechende Zeile am Anfang des Quelltexts ein.

### 3.2.2 Kontextmenü im Quelltextfenster

Im Quelltextfenster können Sie verschiedene Hilfsfunktionen über das Kontextmenü erreichen (Klick mit rechter Maustaste).

	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
<hr/>		
	Comment Block	Ctrl+B
	Uncomment Block	Ctrl+Shift+B
<hr/>		
	Indent	Ctrl+I
	Outdent	Ctrl+Shift+I
	Mark Controlblock	
	Unmark Controlblock	
<hr/>		
	Add to Project	
<hr/>		
	Declaration Info	F2
	Jump to Declaration	Ctrl+F2
	Codesnippets	Ctrl+K X
	Show Declarations	Shift+F2



Die folgenden Befehle verwenden die Cursor-Position oder die aktive Markierung:

- `Cut`: Markierung löschen und in die Zwischenablage kopieren.
- `Copy`: Markierung in die Zwischenablage kopieren.
- `Paste`: Markierung löschen und durch den Inhalt der Zwischenablage ersetzen.
- `Comment Block`, `Uncomment Block`: Zeilen in Kommentar ändern, Seite 22.
- `Indent`, `Outdent`: Textzeilen einrücken, Seite 22.
- `Mark Control block`, `Unmark Control block`: Kontrollstruktur markieren, Seite 32.
- `Declaration Info`: Deklaration eines Befehls oder Variablen anzeigen, Seite 38.
- `Jump to Declaration`: Zur Deklaration eines Befehls oder Variablen springen, Seite 33.

Ohne vorherige Markierung sind folgende Befehle verfügbar:

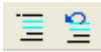
- `Add to Project`: Datei dem aktiven Projekt zufügen.
- `Code snippets`: Textbausteine einfügen, Seite 37.
- `Show all Declarations`: Deklarationen einer Datei anzeigen, Seite 38.

### 3.2.3 Editor-Leiste

Im Quelltextfenster können Sie verschiedene Hilfsfunktionen über die Editor-Leiste erreichen.



Textmarken nutzen, Seite 33.



Zeilen in Kommentar ändern, Seite 22.



Textbereiche ein- / ausfalten, Seite 23.



Deklaration eines Befehls oder Variablen anzeigen, Seite 38.



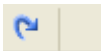
Zur Deklaration eines Befehls oder Variablen springen, Seite 33.



Textbausteine einfügen, Seite 37.



Deklarationen einer Datei anzeigen, Seite 38.



Einen Befehl rückgängig machen oder wieder herstellen.

## 3.3 Quelltext formatieren

Quelltext kann (meist automatisch) formatiert werden, um die Übersicht über die Programmstruktur zu zeigen:

- Syntax hervorheben, Seite 21
- Automatisch formatieren, Seite 21
- Textzeilen einrücken, Seite 22
- Zeilen in Kommentar ändern, Seite 22
- Textbereiche ein- / ausfalten, Seite 23

Hier finden Sie weitere Editor-Funktionen:

- Quelltexte erstellen, Seite 16
- Suchen, markieren und ersetzen, Seite 24
- Programme leichter schreiben, Seite 35

### 3.3.1 Syntax hervorheben

Wenn Sie eine Befehlszeile eingegeben haben, ändert der Editor automatisch die Farbe der eingegebenen Befehlswörter, Variablen- und Feldnamen, und er rückt die Zeilen so ein, dass sich ein übersichtliches Bild ergibt.

Der Editor unterscheidet die von Ihnen eingegebenen Zeichenketten in mehrere Gruppen von Syntaxelementen, die unterschiedlich dargestellt werden. Sie können die Farbgebung unter `Options ► Settings, Editor - Syntax Colors` (siehe Seite 54) nach eigener Vorstellung verändern; dort ist auch eine Übersicht der Syntax-Gruppen angegeben.

Die vollständige Syntax-Hervorhebung erfordert, dass die Option `Parse Declarations` unter `Editor - General` (siehe Seite 53) aktiv ist.

### 3.3.2 Automatisch formatieren

Wenn Sie eine Befehlszeile eingegeben haben, korrigiert der Editor automatisch die Leerzeichen, so dass ein einheitliches Bild entsteht. So erhalten z.B. Operatoren wie „=“ oder Kennwörter wie „If“ rechts und links ein Leerzeichen.

Wenn Sie lieber manuell formatieren, müssen Sie zunächst unter `Editor - General`, `Smart format` das automatische Formatieren abschalten (siehe Seite 53).

### 3.3.3 Textzeilen einrücken

Wenn Sie eine Befehlszeile eingegeben haben, rückt der Editor die Zeilen automatisch so ein, dass sich ein übersichtliches Bild ergibt. Wegen der Automatik ist manuelles Einrücken nicht möglich.

Wenn Sie lieber manuell formatieren, müssen Sie zunächst unter Editor - General, `AutoIndent` die automatische Einrückung abschalten. Anschließend können Sie Einrückungen mit Tabulator [TAB] oder Leerzeichen [SPACE] erzeugen. Mehrere markierte Zeilen können gemeinsam ein- oder ausgerückt werden, indem im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) die Einträge `Indent` oder `Outdent` gewählt werden.

Unter dem Menüpunkt `Options ► Settings, Editor - General, Tabsize` wird eingestellt, wie viele Leerzeichen eine Einrückung hat.

### 3.3.4 Zeilen in Kommentar ändern

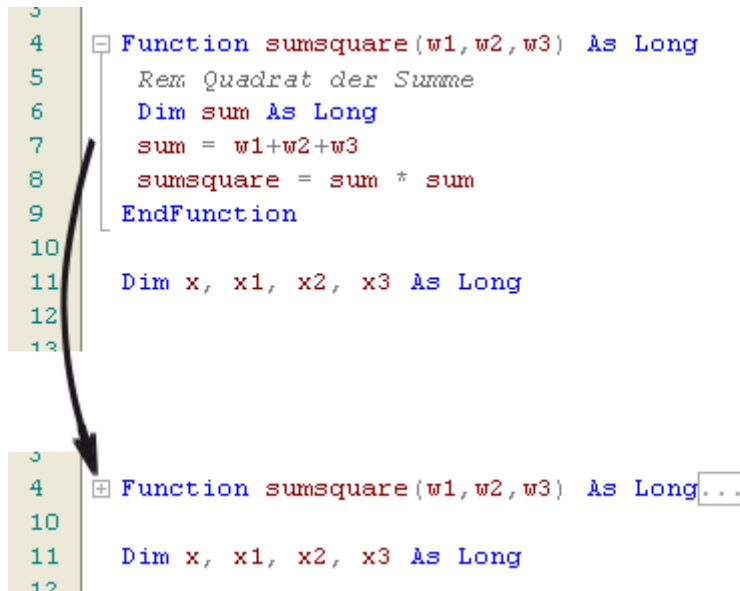
Mehrere markierte Zeilen können gemeinsam in Kommentarzeilen umgewandelt werden, indem im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) der Eintrag `Comment Block` gewählt wird. Als Folge fügt der Editor an jedem Zeilenanfang ein Kommentarzeichen ein, so dass der Compiler diese Zeilen überspringt.


In gleicher Weise löscht `Uncomment Block` ein Kommentarzeichen am Zeilenanfang wieder.

### 3.3.5 Textbereiche ein- / ausfalten

Der Editor erkennt Kontrollstrukturen wie Bedingungen und Schleifen, Programmabschnitte, Makros und Bibliotheks-Module als faltbare Textbereiche. Ein solcher Bereich wird durch eine graue Linie am Zeilenanfang markiert, mit einem Minuszeichen in der ersten Zeile des Bereichs.

Sie falten einen Bereich mit einem Klick auf das Minuszeichen in der ersten Zeile ein; im Beispiel unten würden Sie links neben `Function sumsquare` klicken.



Mit der Schaltfläche Toggle Outlining  können alle faltbaren Textbereiche auf einmal ein- oder ausgefaltet werden.

Faltbare Textbereiche werden nur erkannt, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 53) aktiv ist.

### 3.4 Suchen, markieren und ersetzen

Suchen, markieren oder ersetzen Sie einen beliebigen Text mit diesen Funktionen:

- Text schnell suchen, Seite 24
- Text suchen und ersetzen, Seite 25
- Reguläre Ausdrücke, Seite 30
- Kontrollstruktur markieren, Seite 32
- Textmarken nutzen, Seite 33
- Zu einer Programmzeile springen, Seite 33
- Zur Deklaration eines Befehls oder Variablen springen, Seite 33

Hier finden Sie weitere Editor-Funktionen:

- Quelltexte erstellen, Seite 16
- Quelltext formatieren, Seite 21
- Programme leichter schreiben, Seite 35

#### 3.4.1 Text schnell suchen

Mit dem Tastenkürzel [CTRL]-[F3] können Sie Text schnell suchen. Für die Rückwärtssuche verwenden Sie das Tastenkürzel [CTRL]-[SHIFT]-[F3].

Gesucht wird der markierte Text oder, falls kein Text markiert ist, das Wort an der Cursor-Position. Folgende Suchoptionen sind fest eingestellt:

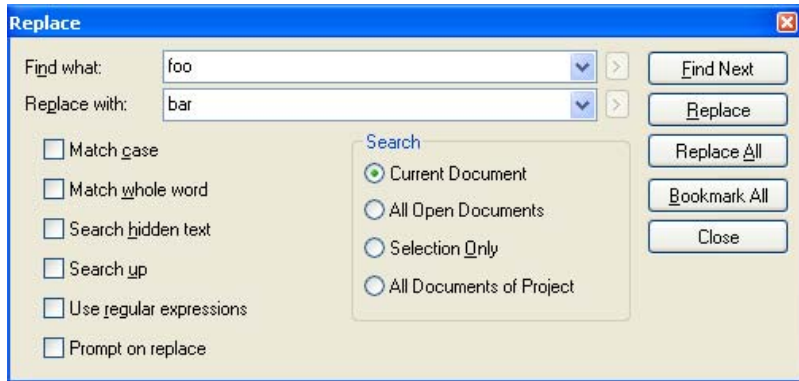
- Groß-Kleinschreibung spielt keine Rolle.
- Der Suchausdruck kann auch ein Teil eines Wortes sein.
- Es werden auch eingeklappte Textbereiche durchsucht.
- Alle offenen Dokumente werden durchsucht.

Bei der Schnellsuche können Sie keine regulären Ausdrücke verwenden und auch keine Textmarken erzeugen.

## 3.4.2 Text suchen und ersetzen

Sie können nach jedem Auftreten einer beliebigen Kombination von Zeichen suchen, zum Beispiel auch nach Groß- und Kleinbuchstaben, ganzen Wörtern, Teilen von Wörtern und regulären Ausdrücken (siehe Reguläre Ausdrücke auf Seite 30).

1. Wählen Sie den Menüeintrag **Edit ► Find** für eine Suche oder **Edit ► Replace** zum Ersetzen. Es erscheint ein Dialogfeld, das geöffnet bleibt, bis Sie es schließen.



2. Geben Sie im Eingabefeld **Find what** den gewünschten Suchausdruck an. Sie können auch einen der zuletzt benutzten Ausdrücke aus der Liste wählen.
3. Nur bei Ersetzen: Geben Sie im Eingabefeld **Replace With** den gewünschten Ersetzungsausdruck an. Sie können auch einen der zuletzt benutzten Ausdrücke aus der Liste wählen.
4. Stellen Sie die Optionen für die Suche ein:

Option	Beschreibung
Match case	<p>Option aktiviert: Es wird nur nach Text gesucht, der genau die Groß-/Kleinschreibung des Suchausdrucks hat.</p> <p>Option deaktiviert: Die Groß-/Kleinschreibung spielt bei der Suche keine Rolle.</p>

Option	Beschreibung
Match whole word	Option aktiviert: Es wird nur nach Text gesucht, bei dem der Suchausdruck ein ganzes Wort ist. Option deaktiviert: Der Suchausdruck kann auch ein Teil eines Wortes sein.
Search hidden text	Die Option bezieht sich auf das Textbereiche ein-/ausfalten (siehe Seite 23). Option aktiviert: Es werden auch eingeklappte Textbereiche durchsucht. Option deaktiviert: Eingeklappte Textbereiche werden übersprungen.
Search up	Option aktiviert: Es wird in Richtung zum Dateianfang hin gesucht. Option deaktiviert: Es wird in Richtung zum Dateiende hin gesucht.
Use regular expressions	Die Option bezieht sich auf Reguläre Ausdrücke (siehe Seite 30). Option aktiviert: Der eingegebene Suchausdruck ist ein regulärer Ausdruck; das ist eine Methode, die variable Textmuster beschreibt. Option deaktiviert: Der eingegebene Suchausdruck ist konstanter Text.
Prompt on replace	Die Option ist nur in Verbindung mit Replace All einsetzbar. Option aktiviert: Bei jedem Vorkommen erscheint ein Dialogfenster, mit dem Sie das Ersetzen steuern. Option deaktiviert: Alle Vorkommen werden ohne Rückfrage ersetzt.

##### 5. Stellen Sie einen Bereich für die Suche ein:



Option	Beschreibung
Current Document	Die Suche beginnt im aktuellen Quelltext an der Cursor-Position. Wenn Text markiert ist, steht der Cursor hinter der Markierung.
All open Documents	Alle geöffneten Dateien werden durchsucht, beginnend mit dem aktuellen Quelltext.
Selection only	Nur der markierte Bereich wird durchsucht. Falls keine Markierung vorhanden ist, startet die Suche an der Cursor-Position.
All Documents of Project	Alle Dateien des Projekts werden durchsucht, auch wenn der aktuelle Quelltext nicht zum Projekt gehört. Für Ersetzen nicht verfügbar. Die Suchergebnisse werden im Fenster Find im Infobereich angezeigt. Mit einem Doppelklick auf ein Ergebnis springt der Cursor in die entsprechende Zeile. Alternativ springen Sie mit den Pfeil-Schaltflächen im Fenster Find von einem Ergebnis zum nächsten.

6. Starten Sie die gewünschte Aktion durch eine der Schaltflächen.
  - **FindNext**: markiert das nächste Vorkommen des Suchausdrucks.
  - **Replace**: ersetzt die aktuelle Markierung durch den Ersetzungsausdruck und markiert das nächste Vorkommen des Suchausdrucks.
  - **Replace All**: ersetzt alle Vorkommen des Suchausdrucks durch den Ersetzungsausdruck.
  - **Bookmark All**: markiert alle Zeilen, in denen der Suchausdruck vorkommt, mit einer Textmarke.
7. Schließen Sie das Dialogfeld, indem Sie auf **Close** klicken. Sie können das Fenster auch geöffnet lassen und im Quelltext weiter arbeiten.

Mit der Option **All Documents of Project** schließt das Dialogfeld automatisch. Die Suchergebnisse finden Sie im Fenster **Find** (im Infobereich unten).

## Anmerkungen

- Der Menüeintrag **Edit ► Find Next** sucht das nächste Vorkommen des Suchausdrucks. Die aktuellen Suchoptionen sind dabei gültig, auch wenn das Suchen-Dialogfenster geschlossen ist.
- Die Aktion **Replace** ersetzt die aktuelle Markierung nur dann, wenn die Markierung dem Suchausdruck entspricht.
- Vorsicht ist angebracht beim Ersetzen mit regulären Ausdrücken, die auch auf „nichts“ passen, z. B. „.+“ oder „a\*“. In solchen Fällen können Schleifen entstehen, in denen so lange ersetzt wird, bis die Zeile zu lang ist.



- Tipp: Wenn Sie mit regulären Ausdrücken eine große Anzahl von Ersetzungen in einem oder gar allen geöffneten Dokumenten vornehmen wollen, können Sie sich zunächst mit **Find Next** und **Replace** vergewissern, dass Sie den Such- und den Ersetzungsausdruck korrekt formuliert haben, bevor Sie mit **Replace All** den Rest ersetzen lassen.

## Beispiele für das Suchen von Text

Beispiele für das Suchen mit regulären Ausdrücken.

1. Alle Leerzeichen oder Tabulatoren am Ende einer Zeile finden:

`[ ]+$`

Der Suchausdruck findet ein oder mehrere Leerzeichen oder Tabulatoren, auf die das Ende der Zeile folgt.

2. Alles, was in einer Zeile steht, finden:

`^.+`

Der Suchausdruck findet den Anfang einer Zeile, auf den ein oder mehrere beliebige Zeichen folgen, bis zum Ende der Zeile.

3. `$12.34` finden:

`\$12\.34`

Vor `.` und `$` steht jeweils der Backslash (`\`), weil diese beiden Zeichen Metazeichen sind. Metazeichen werden nicht als Zeichen selbst, sondern als Bestandteil eines Musters interpretiert. Der Backslash (als

„Escape-Code“) hebt diese besondere Wirkung auf, so dass tatsächlich nach einem Punkt und nach einem Dollar-Zeichen gesucht wird.

4. Eine Zeichenfolge finden, die in *TiCoBasic* als Variablenname gültig ist:

```
\b[a-z][_a-z0-9]*
```

Der Suchausdruck findet ein Wort, das mit einem Groß- oder Kleinbuchstaben beginnt, worauf kein, ein oder mehrere Buchstaben, Ziffern oder Unterstriche folgen.

5. Bei verschachtelten Klammern den innersten Ausdruck finden:

```
\([^\\(\\)]*\)
```

Der Suchausdruck findet eine öffnende Klammer, auf die kein, ein oder mehrere beliebige Zeichen außer der öffnenden und der schließenden Klammer folgen, auf die wiederum eine schließende Klammer folgt.

6. Wiederholungen finden:

```
([0-9]+)-\1
```

Das Suchmuster in Klammern (...) steht für eine oder mehrere Ziffern; durch die Klammern wird das Muster gespeichert. Anschließend folgen ein Bindestrich und mit \1 wieder das gespeicherte Muster. Dieser reguläre Ausdruck würde also etwa 14-14 und 08-08 finden, aber nicht zum Beispiel 08-15.

### Beispiele für das Ersetzen von Text

Beispiele für das Ersetzen mit regulären Ausdrücken.

1. Finde zwei Zahlen, die durch ein oder mehrere Leerzeichen getrennt sind:

```
([0-9]+) + ([0-9]+)
```

vertausche sie, und trenne sie durch einen Doppelpunkt:

```
$2:$1
```

2. Um die folgenden Veränderungen simultan durchzuführen:

aus X100000 soll werden X100.000

aus Y100123 soll werden Y100.123

aus Z600 soll werden Z.600

Suche nach: ([XYZ])([0-9]\*)([0-9][0-9][0-9])

Ersetze durch: \$1\$2.\$3

### 3.4.3 Reguläre Ausdrücke

Ein regulärer Ausdruck ist ein Suchausdruck, bei dem so genannte Metazeichen (siehe Liste unten) ein variables Suchmuster beschreiben. Die Metazeichen sind nur für das Suchen gültig, nicht für das Ersetzen.

Um mit einem regulären Ausdruck beim Suchen/Ersetzen zu verwenden, müssen Sie die Option `Use regular expressions` im Dialogfenster aktivieren. Rechts neben den Eingabefeldern werden dadurch Schaltflächen „>“ aktiv, über die Sie Metazeichen eingeben können.

Die Syntax für reguläre Ausdrücke ist im .NET-Framework 2.0 definiert. Eine ausführliche Beschreibung finden Sie im Internet unter der Adresse <http://msdn2.microsoft.com> (nach „reguläre Ausdrücke“ suchen).

Meta-zeichen:	Bedeutung:
.	Ein beliebiges einzelnes Zeichen. Beispiel: Das Muster <code>Ma.s</code> trifft unter anderem zu auf <code>Mais</code> , <code>Mars</code> und <code>Maus</code> , nicht aber auf <code>Mas</code> .
[ ]	Ein beliebiges Zeichen von denen, die <ol style="list-style-type: none"> <li>explizit in den eckigen Klammern angegeben sind, oder</li> <li>eines aus dem Bereich von Zeichen, der durch den Bindestrich (-) definiert ist.</li> </ol> Beispiele: Der Suchausdruck <code>h[aeiou][a-z]t</code> findet unter anderem: <code>hart</code> , <code>halt</code> , <code>heft</code> , <code>holt</code> und <code>hut</code> . Das Muster <code>[A-Za-z]</code> entspricht einem beliebigen Buchstaben. Der reguläre Ausdruck <code>x[0-9]</code> entspricht <code>x0</code> , <code>x1</code> usw. bis <code>x9</code> .
[^]	Ein beliebiges Zeichen außer denen, die hinter dem Caret (^, „Dach“) stehen. Beispiel: Das Muster <code>h[^uo]t</code> passt unter anderem auf <code>hat</code> und <code>hit</code> , aber nicht auf <code>hut</code> oder <code>hot</code> .
^	Der Anfang einer Zeile (Spalte 1). Beispiel: Der Suchausdruck <code>^Anfang</code> entspricht nur dann dem Text <code>Anfang</code> , wenn dieser ganz vorne in einer Zeile steht.

Meta- zeichen:	Bedeutung:
\$	Das Ende einer Zeile (letzte Spalte). Benutzen Sie dieses Zeichen und nicht das Zeilenschaltzeichen <code>\n</code> , wenn Sie nach einem Ausdruck suchen, der am Ende einer Zeile steht. Beispiel: Der Suchausdruck <code>Ende\$</code> findet die Zeichenfolge <code>Ende</code> nur dann, wenn <code>Ende</code> das letzte Wort einer Zeile ist.
\b	Der Anfang eines Wortes.
\B	Das Ende eines Wortes.
\n	Zeilenschaltzeichen (Newline). Zu verwenden bei Ausdrücken, die sich über mehrere Zeilen erstrecken. Hinter <code>\n</code> dürfen nicht die Operatoren <code>*</code> , <code>+</code> oder <code>{ }</code> stehen. Um ein Suchmuster am Ende einer Zeile zu finden, sollten Sie unbedingt den Operator <code>\$</code> und nicht das Zeilenschaltzeichen verwenden.
( )	Der Text innerhalb der Klammern wird als Muster in internen Registern abgelegt. Der Inhalt eines Registers kann an anderer Stelle im Suchausdruck oder im Ersetzungsausdruck wieder abgerufen werden. Bis zu 9 gespeicherte Muster sind zulässig. Sie werden in der Reihenfolge ihres Auftretens in dem regulären Ausdruck durchnummeriert. Der Rückbezug auf das Muster lautet <code>\$x</code> im Ersetzungsausdruck und <code>\x</code> im Suchausdruck, mit <code>x = 1 ... 9</code> . Beispiel: Der Ausdruck <code>([a-z]+) ([a-z]+)</code> findet <code>isses so. \$2 \$1</code> würde ihn ersetzen durch <code>so isses</code> .
*	Beliebig häufiges Auftreten des vorangehenden Zeichens oder Ausdrucks – kein Mal, einmal oder mehrmals. Beispiel: <code>hu*f</code> findet unter anderem <code>hf</code> , <code>huf</code> und <code>huuuf</code> .
?	Kein oder genau ein Auftreten des vorangehenden Zeichens oder Ausdrucks. Beispiel: <code>hu?f</code> findet <code>hf</code> und <code>huf</code> , nicht aber <code>huuuf</code> .
+	Mindestens ein (d.h. ein- oder mehrmaliges) Auftreten des vorangehenden Zeichens oder Ausdrucks. Beispiel: <code>hu+f</code> findet <code>huf</code> und <code>huuf</code> , nicht aber <code>hf</code> .

Meta- zeichen:	Bedeutung:
	Entweder der Ausdruck auf der linken Seite des Striches oder der auf der rechten Seite. Beispiel: <code>huf huuf</code> findet <code>huf</code> oder <code>huuf</code> .
\	Hebt die besondere Wirkung aller Operatoren auf, so dass sie wie normaler, konstanter Text behandelt werden. Um nach dem Backslash <code>\</code> zu suchen, muss also <code>\\</code> verwendet werden. Beispiel: <code>^a</code> bedeutet, dass nach einem <code>a</code> am Anfang einer Zeile gesucht wird, dagegen sucht <code>\\^a</code> nach der Zeichenfolge <code>^a</code> .

### 3.4.4 Kontrollstruktur markieren

Sie können die Zeilen einer Kontrollstruktur oder eines Programmabschnitts auf einmal markieren, z.B. um verschachtelte Strukturen optisch zu prüfen. Hierzu setzen Sie den Cursor auf eines der Kennwörter der Struktur und wählen im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) den Eintrag `Mark Control block`.

Es kann nur eine Kontrollstruktur gleichzeitig markiert werden.

Die Markierung wird mit `Unmark Control block` (im Kontextmenü) wieder gelöscht. Hierbei ist es gleichgültig, an welcher Stelle der Cursor steht.

Folgende Kontrollstrukturen werden erkannt:

- Programmabschnitte **Init:**, **Event:**, **Finish:**
- `Do ... Until`
- `For ... Next`
- `If ... EndIf`
- `SelectCase ... EndSelect`
- `Function ... EndFunction`
- `Sub ... EndSub`
- `Lib_Function ... Lib_EndFunction`
- `Lib_Sub ... Lib_EndSub`

Alle Kontrollstrukturen sind auch faltbare Bereiche (siehe Textbereiche ein- / ausfallen, Seite 23).

### 3.4.5 Textmarken nutzen

Mit Textmarken können Sie, ähnlich einem Lesezeichen, bestimmte Zeilen in einem Quelltext kennzeichnen. Derart markierte Zeilen können einfach angesprungen werden.

Folgende Aktionen stehen zur Verfügung:

- Textmarke setzen

Sie können eine Textmarke auf eine Zeile setzen, indem Sie entweder in der Editor-Leiste den Befehl `Toggle Bookmark` wählen oder im Dialogfeld `Replace` auf `Bookmark All` klicken.

Mit `Toggle Bookmark` werden gesetzte Textmarken wieder einzeln entfernt.

- Zur nächsten Textmarke springen

Wählen Sie in der Editor-Leiste den Befehl `Next Bookmark`.

- Zur vorherigen Textmarke springen

Wählen Sie in der Editor-Leiste den Befehl `Previous Bookmark`.

- Alle Textmarken entfernen

Wählen Sie in der Editor-Leiste den Befehl `Delete all Bookmarks`.

Einzeln werden Textmarken mit `Toggle Bookmark` entfernt.

Textmarken werden mit der Datei zusammen abgespeichert.

### 3.4.6 Zu einer Programmzeile springen

Sie können zu einer bestimmten Programmzeile im Quelltext springen, indem Sie auf die Zeilennummer in der Statusleiste doppelklicken oder im Menü `Edit` den Eintrag `Goto Line` wählen. Es öffnet sich ein Dialogfenster, in dem Sie die Nummer der gewünschten Programmzeile eingeben.

Um Zeilennummern anzuzeigen, muss die Option `show linenumbers` unter `Editor - General` (siehe Seite 53) aktiv sein.

### 3.4.7 Zur Deklaration eines Befehls oder Variablen springen

Sie können von einem Variablennamen aus direkt zu deren Deklaration springen. Diese Möglichkeit gilt für alle selbst deklarierten Namen: lokale Variablen, Felder, Befehle (`Sub`, `Function`) und symbolische Namen (`#Define`).

Sie springen zur Deklaration, indem Sie den Cursor auf den selbst deklarierten Namen setzen und dann entweder im Kontextmenü (rechte Maustaste)

den Eintrag `Jump to Declaration` aufrufen, oder die Schaltfläche `Jump to Declaration` in der Editor-Leiste aufrufen.

Der Sprung zur Deklaration ist nur möglich, wenn die Option `Parse Declarations` unter `Editor - General` (siehe Seite 53) aktiv ist.

Für Befehle aus den Standard-Include-Dateien sowie für globale Variablen **PAR** steht die Funktion verständlicherweise nicht zur Verfügung.



### 3.5 Programme leichter schreiben

Die folgenden Funktionen erleichtern das Programmieren erheblich:

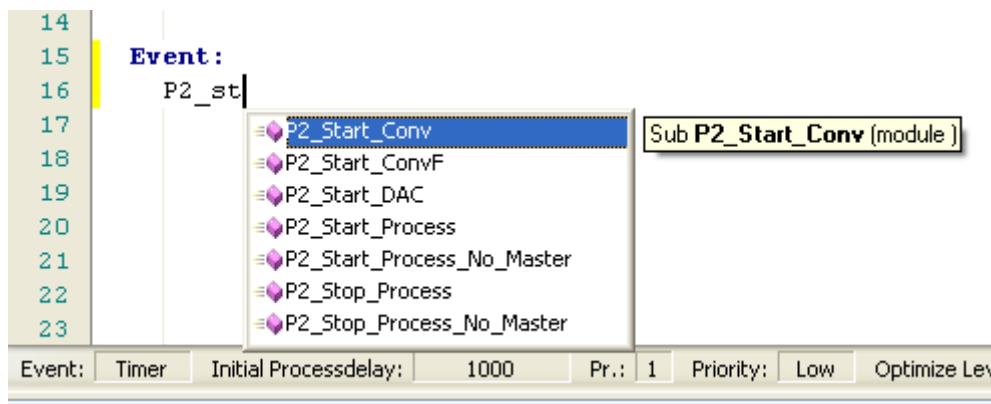
- Befehle und Variablen automatisch vervollständigen, Seite 35
- Befehlsparameter anzeigen, Seite 37
- Textbausteine einfügen, Seite 37
- Deklaration eines Befehls oder Variablen anzeigen, Seite 38
- Deklarationen einer Datei anzeigen, Seite 38
- Verwendete globale Variablen und Felder anzeigen, Seite 39

Hier finden Sie weitere Editor-Funktionen:

- Quelltexte erstellen, Seite 16
- Quelltext formatieren, Seite 21
- Suchen, markieren und ersetzen, Seite 24

#### 3.5.1 Befehle und Variablen automatisch vervollständigen

Sie können Schlüsselwörter, Befehls- und Variablennamen und sogar Textbausteine automatisch vervollständigen, indem Sie die Tastenkombination [CTRL-SPACE] drücken.



Durch die automatische Vervollständigung müssen Sie in den meisten Fällen Schlüsselwörter, Befehle und Variablen nicht vollständig ausschreiben.

Um die automatische Vervollständigung zu nutzen, gehen Sie vor wie folgt:

1. Schreiben Sie die ersten Buchstaben des Wortes und drücken [CTRL-SPACE].

Es wird eine Wortliste angezeigt, deren Einträge den bisher eingegebenen Text vervollständigen können.

Wenn Sie die Vervollständigung nach einem Leerzeichen aufrufen, enthält die Liste alle verfügbaren Kennwörter.

2. Wählen Sie den gewünschten Listeneintrag mit der Maus oder mit den Pfeiltasten aus.

Rechts neben dem gerade markierten Listeneintrag wird nach einem Moment eine von mehreren Erläuterungen angezeigt:

- die Deklaration des Befehls oder der Variablen
- die Angabe „Reserved Keyword“
- der vollständige Textbaustein (siehe unten)

3. Wenn Sie weiteren Text eingeben, wird die Liste nicht automatisch aktualisiert. Drücken Sie zur Aktualisierung der Liste erneut die Tastenkombination [CTRL-SPACE].

4. Sie übernehmen den markierten Eintrag aus der Liste am besten durch Eingeben einer Klammer (bei einem Befehl) oder eines Leerzeichens.


Sie können stattdessen aber auch die [EINGABE]-Taste verwenden oder ein anderes, nicht-alphanumerisches Zeichen eingeben.

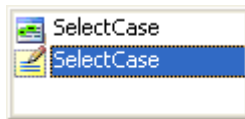
Automatisches Vervollständigen ist nur möglich, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 53) aktiv ist.

### 3.5.2 Textbausteine einfügen

Der Editor bietet Ihnen die Möglichkeit, mit Textbausteinen (code snippets) zu arbeiten, welche in einer vorgegebenen Sammlung zusammengefasst sind. Je nach Definition kann ein Textbaustein wenige Zeichen, mehrere Zeilen oder ein ganzes Programm einfügen.

Sie fügen einen Textbaustein an der Cursor-Position ein, indem Sie eine der folgenden Aktionen ausführen:

- Sie geben die ersten Zeichen eines Textbaustein-Kennworts ein, z. B. `Sele` für eine `SelectCase`-Struktur, drücken die Tastenkombination [CTRL-SPACE] und wählen aus der Liste den Textbaustein  `Select-Case`.  
Siehe auch Befehle und Variablen automatisch vervollständigen.



- Sie rufen über Kontextmenü oder Editor-Leiste die Funktion `Code-snippets` auf. Es erscheint eine Auswahlliste mit Ordnern, die jeweils mehrere Textbausteine (oder weitere Ordner) enthalten.

Sie navigieren durch die Menüs per Maus oder per Tastatur. Folgende Tasten sind belegt:

- Pfeil oben/unten: Listeneintrag markieren.
- Eingabe: Markierten Textbaustein einfügen oder neue Menüebene öffnen.
- Backspace: In vorige Menüebene zurückkehren.

Sobald Sie einen Textbaustein markiert haben, erscheint rechts davon das zugehörige Tastaturkürzel.

- Sie geben das Tastaturkürzel des Textbausteins ein, gefolgt von [TAB].

Um die Liste der Textbausteine und der Tastaturkürzel zu sehen, öffnen Sie die Datei `<codesnippets.xml>` im Ordner `C:\ADwin\TiCoBasic\Common\` mit einem Browser.

### 3.5.3 Befehlsparameter anzeigen

Bei Befehlen werden die zugehörigen Übergabeparameter automatisch als Tooltip angezeigt, sobald Sie nach dem Befehlsnamen eine öffnende Klammer eingeben. Im Tooltip wird derjenige Übergabeparameter fett dargestellt, den Sie gerade eingeben.

Der Tooltip verschwindet, sobald der Cursor außerhalb der Klammern um die Parameter steht. Sie können die Anzeige wieder aktivieren, wenn Sie die öffnende Klammer überschreiben. Alternativ können Sie über Kontextmenü oder Editor-Leiste die Funktion `Declaration Info` aufrufen, die die vollständige Deklaration des Befehls anzeigt.

Die automatische Anzeige von Befehlsparametern ist nur möglich, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 53) aktiv ist.

### 3.5.4 Deklaration eines Befehls oder Variablen anzeigen

Sie können zu Befehlen, Variablen und anderen deklarierten Kennwörtern die zugehörige Deklaration sowie Hinweise als Tooltip anzeigen, indem Sie

- mit der Maus über das Wort fahren.

Die Deklaration wird nur dann automatisch angezeigt, wenn die Option `Display quick info on mouse over` unter Editor - General (siehe Seite 53) aktiv ist.

- den Cursor auf das Kennwort setzen und die Taste [F2] drücken.
- den Cursor auf das Kennwort setzen und in der Editor-Werkzeugliste oder im Kontextmenü den Eintrag `Declaration Info` wählen.

Diese Möglichkeit gilt für alle Kennwörter, die zum Sprachumfang von *TiCo-Basic* gehören oder selbst deklariert wurden: lokale und globale Variablen, Felder, Befehle (`Sub`, `Function`) und symbolische Namen (`#Define`).


Die Anzeige der Deklaration ist nur möglich, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 53) aktiv ist.

### 3.5.5 Deklarationen einer Datei anzeigen

Sie können alle zu einer Quelltextdatei gehörenden Deklarationen, Include- und Library-Dateien anzeigen, indem Sie im Info-Bereich das Fenster „Declarations“ in den Vordergrund stellen.

Die Deklarationen können nur angezeigt werden, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 53) aktiv ist.

## 3.5.6 Verwendete globale Variablen und Felder anzeigen

Um globale Variablen und Felder anzuzeigen, die im aktiven Quelltext und im zugehörigen Projekt (falls vorhanden) verwendet werden, drücken Sie die Schaltfläche `Scan Global Variables`  im Parameterfenster (siehe auch Seite 61) drücken.

Sie erhalten zwei Anzeigen:

- im Fenster „Global Variables“ werden alle verwendeten globalen Variablen und Felder angezeigt. Näheres siehe Seite 69.
- im Parameterfenster werden alle verwendeten globalen Variablen farbig hervorgehoben (globale Felder nicht).

Die Hervorhebung ist farbig je nach Verwendung der Parameter:

- Grün: Parameter wird nur im aktiven Quelltext verwendet. 

2	1336227
---	---------
- Rot: Parameter wird im aktiven Quelltext verwendet, aber auch in einem anderen Quelltext des Projekts. 

1	707279
---	--------
- Blau: Parameter wird im Projekt verwendet, jedoch nicht im aktiven Quelltext. 

9	5B12H
---	-------

Mit der Schaltfläche `Clear Scan`  werden die beide Anzeigen wieder gelöscht.

Bei Änderungen im Quelltext werden die Anzeigen nicht automatisch aktualisiert. Rufen Sie stattdessen `Scan Global Variables` erneut auf.

## 3.6 TiCo-Binärdatei zum TiCo-Prozessor übertragen

Sie können mit der Oberfläche *TiCoBasic* eine vorhandene Binärdatei zum *TiCo*-Prozessor oder in den *TiCo*-Bootloader übertragen.

### 3.6.1 TiCo-Binärdatei übertragen


Um eine Binärdatei als Prozess zum *TiCo*-Prozessor zu übertragen, gehen Sie vor wie folgt:

- Erzeugen Sie eine Binärdatei mit `Build ► Make Bin File`; siehe auch Seite 74.

Stellen dabei schon die Eigenschaften (Priorität, Prozessnummer, Prozessstyp etc.) für den Prozess ein.

- Wählen Sie im Menü **Tools** den Eintrag **Load Bin File** und im nächsten Fenster die Binärdatei.

Bestätigen Sie die Auswahl mit **Öffnen**; *TiCoBasic* überträgt nun die Binärdatei als Prozess zum *TiCo*-Prozessor, der in den Compiler-Optionen eingestellt ist.

- Der Prozess wird nicht automatisch gestartet. Um den Prozess zu starten, klicken Sie auf die Schaltfläche **Start Process** .

### 3.6.2 TiCo-Bootloader programmieren

Der *TiCo*-Bootloader lädt und startet ausgewählte *TiCoBasic*-Prozesse beim Starten der *ADwin*-Hardware automatisch.

Um den *TiCo*-Bootloader zu programmieren, gehen Sie vor wie folgt:

- Erzeugen Sie eine Binärdatei mit **Build ▶ Make Bin File**; siehe Seite 74.

Stellen Sie dabei schon die Eigenschaften (Priorität, Prozessnummer, Prozessstyp etc.) für den Prozess ein.

- Wählen Sie im Menü **Tools** den Eintrag **Bootloader....**

Das Dialogfenster **Bootloader** öffnet sich.



- Klicken Sie auf die Schaltfläche `Load Bootloader` und wählen im nächsten Fenster die Binärdatei.
- Geben Sie mit der Schaltfläche `Enable Bootloader` den Bootloader-Betrieb frei. Damit werden beim Einschalten der *ADwin*-Hardware die Prozesse der Binärdatei automatisch gestartet.  
Mit der Schaltfläche `Disable Bootloader` können Sie den Bootloader-Betrieb zeitweilig sperren und später mit `Enable Bootloader` wieder freigeben.
- Schließen Sie das Dialogfenster mit `Close`.

## 3.7 Projektverwaltung

Sie können beliebig viele Quelltexte, Include-Dateien und Library-Dateien gemeinsam als Projekt verwalten, beispielsweise wenn Sie eine Anwendung mit mehreren Prozessen programmieren. Es kann jeweils nur ein einziges Projekt geöffnet sein.

Mit dem Projekt wird auch die Darstellung der Bedienoberfläche gespeichert: Fenstergröße-, position, geöffnete Projektdateien. Beim Öffnen des Projekts wird die gespeicherte Darstellung wieder hergestellt.

Für die Zusammenstellung der Projektdateien gelten folgende Regeln (siehe auch Kapitel 6.1 „Prozessverwaltung“):

- Mindestens ein Prozess muss hohe Priorität haben.
- Erlaubt ist nur ein einziger zeitgesteuerter Prozess mit hoher Priorität.
- Es kann einen zeitgesteuerten Prozess mit niedriger Priorität geben, wenn gleichzeitig ein Prozess mit hoher Priorität zum Projekt gehört.

Sie können einen zeitgesteuerten und einen extern gesteuerten Prozess miteinander kombinieren. Wenden Sie sich in diesem Fall bitte an unseren Support ([support@adwin.de](mailto:support@adwin.de)); wir informieren Sie dann über die erforderlichen Vorkehrungen.

Folgende Funktionen stehen nur in einem Projekt zur Verfügung:

- die in einem Projekt verwendeten globalen Variablen hervorheben (siehe Kapitel 3.5.6).
- alle Dateien des Projekts durchsuchen, darunter auch die nicht geöffneten Dateien des Projekts.

Sie müssen nur die Option `All Documents of Project` im Suchen-Fenster aktivieren (siehe Kapitel 3.4.2 „Text suchen und ersetzen“). Diese Option ist für das Ersetzen nicht verfügbar.

- alle Dateien des Projekts auf einmal speichern, mit `Save all Files of Project` im Kontextmenü des Projektfensters.
- Windows Explorer mit dem Pfad der markierten Datei öffnen mit `Open Path in Explorer Window` aus dem Kontextmenü im Projektfenster.

Die Befehle zur Projektverwaltung finden Sie im Kontextmenü des Projektfensters (siehe auch „Projektfenster“ auf Seite 59) oder im Menü `File` (Kapitel 3.8.1).

### 3.8 Menüs

Sie finden in der Menüleiste folgende Funktionen:

- |                         |  |            |
|-------------------------|--|------------|
| – <code>File:</code>    | Dateien und Projekte verwalten.            | (Seite 43) |
| – <code>Edit:</code>    | Quelltexte editieren.                      | (Seite 44) |
| – <code>View:</code>    | Fenster und Leisten anzeigen.              | (Seite 44) |
| – <code>Build:</code>   | Ausführbare Programme erzeugen.            | (Seite 45) |
| – <code>Options:</code> | Optionen aller Art einstellen.             | (Seite 46) |
| – <code>Tools:</code>   | Verschiedene Hilfsfunktionen.              | (Seite 57) |
| – <code>Window:</code>  | Quelltextfenster anordnen.                 | (Seite 57) |
| – <code>Help:</code>    | Hilfe, Versions- und Lizenz-Informationen. | (Seite 58) |



## 3.8.1 Das Menü „File“

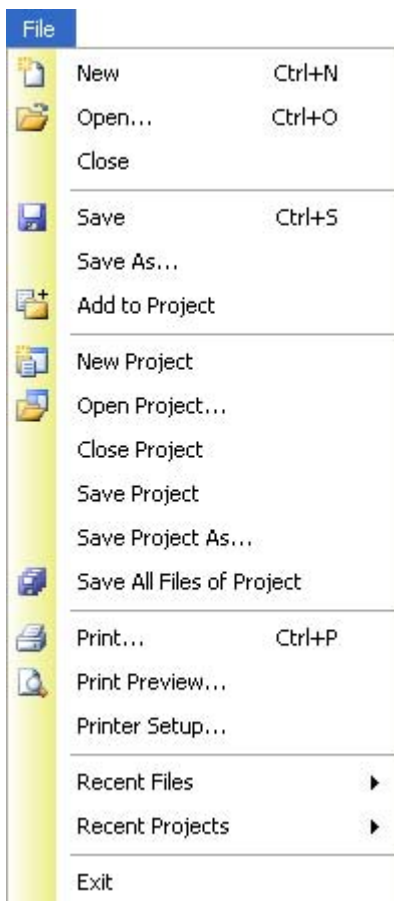
Das Menü `File` enthält Befehle zum Verwalten von Dateien und Projekten.

Mit den Befehlen können Sie Dateien öffnen, neu anlegen, speichern oder schließen. Sie können beliebig viele Quelltexte bearbeiten.

In gleicher Weise wie Dateien können Sie auch ein Projekt öffnen, speichern und neu erzeugen; es kann nur ein einziges Projekt gleichzeitig geöffnet sein kann. Weitere Befehle sind über das Projektfenster zugänglich (siehe Kapitel 3.9.2).

Das Menü enthält auch die Druckfunktionen (Drucken, Druckvorschau und Druckereinstellung).

Unter `Recent Files` und `Recent Projects` wird eine Liste der 10 zuletzt geöffneten Dateien und Projekte angezeigt.










Quelltextdateien werden mit einem eigenen Format gespeichert. Um Dateien mit *ADbasic 4* zu nutzen, kann man sie mit `Save as in den Dateityp ADbasic4 Bas-File` konvertieren.

### 3.8.2 Das Menü „Edit“

Das Menü **Edit** enthält die nach den Windows-Konventionen üblichen Editier-Funktionen.

Darüber hinaus enthält das Menü eine Such-Funktion (**Find**, **Find Next**) sowie eine Ersetzen-Funktion (**Replace**); siehe Text suchen und ersetzen auf Seite 25.

Wir empfehlen Ihnen nicht, mit **Cut** and **Paste** Zeichen oder Programmzeilen aus anderen Programmen in einen Quelltext einzufügen. Es kann hierbei zu unvorhergesehenen Fehlfunktionen kommen.

Edit		
	Undo	Ctrl+Z
	Redo	Ctrl+Shift+Z
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Select All	Ctrl+A
	Find...	Ctrl+F
	Find Next	F3
	Replace...	Ctrl+H
	Goto Line..	Ctrl+G

### 3.8.3 Das Menü „View“

Im Menü **View** stellen Sie ein, welche Leisten der Entwicklungsumgebung angezeigt werden:

- die Werkzeugleiste (Standard Toolbar)
- die Editor-Leiste
- die ADtools-Leiste
- die Statuszeile

Näheres zum Prozessfenster finden Sie in Kapitel 3.9.4 auf Seite 62, zur Werkzeugleiste siehe Abb. 2.

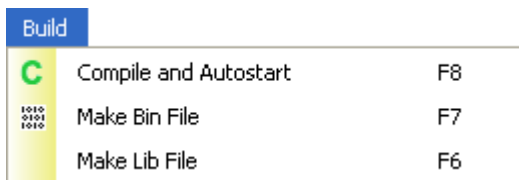
View	
<input checked="" type="checkbox"/>	Standard Toolbar
<input checked="" type="checkbox"/>	Editor Toolbar
<input checked="" type="checkbox"/>	ADtools Toolbar
<input checked="" type="checkbox"/>	Statusbar
	Restore Default Layout


Mit **Restore Default Layout** stellen Sie mit einem Tastendruck die Standard-Ansicht wieder her, die beim ersten Öffnen des Programms *TiCoBasic* aktiv war, darunter auch die Einstellungen der Toolbox (Seite 59).

## 3.8.4 Das Menü „Build“

Im Menü **Build** können Sie Quelltext übersetzen:

- mit **Compile** in einen Prozess.
- mit **Make Bin File** in eine Binärdatei.
- mit **Make Lib File** in eine Library.



Bitte beachten Sie: Vor dem Kompilieren werden automatisch alle geänderten Quelltexte, Library- und Include-Dateien gespeichert. (AutoSave) 


Eine Änderung kann auch durch automatisches Einrücken von Textzeilen (siehe Kapitel 3.3.3 auf Seite 22) stattfinden, beispielsweise beim Öffnen einer vorher nicht formatierten Datei.

**Compile** ist der umfassendste Befehl des Menüs: Er übersetzt das gesamte Projekt (oder den einzelnen Quelltext) und überträgt den erzeugten Binär-code als Prozess auf den *TiCo*-Prozessor.

Der Prozess wird auf dem *TiCo*-Prozessor automatisch gestartet.

Fehler und Warnungen beim Kompilieren werden im Infofenster angezeigt. Ein Doppelklick auf die Fehlermeldung markiert die betreffende Zeile rot.

**Make Bin File** ist nur für lizenzierte *TiCoBasic*-Benutzer verfügbar. Der Befehl übersetzt das gesamte Projekt (oder den einzelnen Quelltext) in Binär-code und speichert diesen automatisch in einer Datei ab. Die Binär-datei wird mit dem Namen und im Verzeichnis der Quelltext-Datei abgelegt, jedoch mit der Dateierdung `.TIx`. Hierbei steht `x` für den Prozessortyp (siehe auch Das Menü „Options“, Dialogfenster „Process Options“).

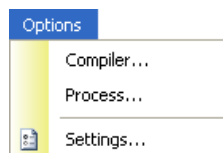
Eine Binärdatei mit der Endung `<* .TI3>` beispielsweise können Sie auf den *TiCo*-Prozessor vom Typ 1 übertragen. Sie können die übertragen (siehe Kapitel 3.6.1 „*TiCo*-Binärdatei übertragen“) oder aber aus einer Entwicklungsumgebung (wie C oder Visual Basic). 

`Make Lib File` ist nur für lizenzierte *TiCoBasic*-Benutzer verfügbar. Der Befehl übersetzt das gesamte Projekt (oder den einzelnen Quelltext) in Binärcode und speichert diesen automatisch als Library-Datei ab. Die Library wird mit dem Namen und im Verzeichnis der Quelltext-Datei abgelegt, jedoch mit der Dateierdung `.TLx`. Hierbei steht `x` wieder für den Prozessortyp (s.o.).

Anschließend können Sie die Library in andere Quelltexte einbinden und auf deren Funktionen und Unterprogramme zugreifen (siehe Kapitel 4.5.3 auf Seite 96).

### 3.8.5 Das Menü „Options“

Im Menü `Options` können Sie eine Reihe von Optionen einstellen, die unmittelbar wirksam werden. Zu jedem Menüpunkt wird ein eigenes Dialogfenster aufgerufen, in dem Sie Ihre Einstellungen vornehmen.



#### Dialogfenster „Compiler Options“

Die Einstellungen, die Sie in diesem Dialogfenster machen (bitte von oben nach unten), werden für jedes Kompilieren eines Quelltextes verwendet. Insbesondere sind dies Informationen über das *ADwin*-System und den *TiCo*-Prozessor, auf dem die übersetzten Quelltexte als Prozess laufen sollen.

Wenn Sie Quelltexte für verschiedene *TiCo*-Prozessoren kompilieren möchten, müssen Sie die Einstellungen in diesem Dialogfenster für jeden *TiCo*-Prozessor neu anpassen.



Abb. 3 – Das Dialogfenster `Compiler Options`

- **System:** Wählen Sie den Eintrag aus, der Ihrem *ADwin*-System entspricht.
- **ADwin CPU:** Wählen Sie den Prozessor des *ADwin*-System.
- **Device No.:** Wählen Sie die Gerätenummer, mit der das gewünschte *ADwin*-System angesprochen werden kann.

Sie vergeben die Gerätenummer mit dem Programm `<ADconfig.exe>`. Die Werkseinstellung ist 150 Hex.

- **Module Address** (nur für *ADwin-Pro II*-System): Wählen Sie die Adresse des Moduls, auf dem sich der *TiCo*-Prozessor befindet.
- **Do Not access the device:** Bei deaktivierter Option wird die beim Kompilieren erzeugte Binärdatei automatisch zur Hardware übertragen. Die Hardware muss daher für das Kompilieren erreichbar sein.  
Bei aktiver Option können Quelltexte auch dann für die eingestellte *ADwin*-Hardware kompiliert werden, wenn sie nicht mit dem PC verbunden ist.

Beachten Sie: Die Menüeinträge zum Übertragen von Bootloader- oder Binärdateien (Kapitel 3.8.6 „Das Menü „Tools“, Seite 57) sind nur aktiv, wenn die Option deaktiviert ist.

- Remember Device No.: Bei aktiver Option wird die zuletzt verwendete Device No. (siehe oben) beim Schließen des Programms gespeichert; beim Neustart wird diese Nummer automatisch eingestellt.

Bei deaktivierter Option wird die Device No. nicht gespeichert. Beim Start von *TiCoBasic* wird die zuletzt – bei aktiver Option – gespeicherte Device No. verwendet.

## Dialogfenster „Process Options“

Sie legen in diesem Dialogfenster Compiler-Optionen für das aktive Quelltextfenster fest, d. h. Sie bestimmen Eigenschaften des Prozesses, der aus dem aktiven Quelltext übersetzt und zur *ADwin*-Hardware übertragen wird.

Dies gilt sinngemäß auch für Library-Dateien, bei denen aber nur die Option `Optimize` eingestellt werden kann.

Sie müssen für jedes Quelltextfenster separat die nötigen Einstellungen vornehmen, indem Sie das Dialogfenster jeweils neu aufrufen (es sei denn, Sie möchten die Voreinstellung verwenden). Sie können das Dialogfenster mit einem Doppelklick auf die Statuszeile im Quelltextfenster öffnen.



Abb. 4 – Das Dialogfenster Process Options

- Process: Prozessnummer



Stellen Sie die Nummer ein, unter der der übertragene Prozess auf dem *TiCo*-Prozessor angesprochen wird.

Wenn Sie mehrere Prozesse gleichzeitig auf einem *TiCo*-Prozessor ablaufen lassen, müssen Sie jedem Prozess eine eigene Nummer zuweisen.

- **Eventsource:** Stellen Sie ein, welches Event-Signal den Abschnitt **Event:** Ihres Prozesses starten soll.
  - **Timer**  
regelmäßige Impulse des internen Zählers dienen als Event-Signal. Mit der Systemvariablen **Processdelay** legen Sie fest, in welchen Zeitabständen der Zähler ein Event-Signal auslöst.
  - **External**  
Bestimmte Werte in der Hardware-Adresse **Address** im *TiCo*-Prozessor dienen als Event-Signal. Der Prozess läuft immer mit hoher Priorität ab.  
Weitere Einstellungen siehe unten bei **External Event**.
  - **None**  
Der Prozesstyp **None** (ohne Event-Signal) wird nur für – meist in Assembler programmierte – Spezialanwendungen benötigt und schließt andere Prozesstypen aus. Wenn nicht anders programmiert, reagiert der Prozess nicht auf Event-Signale und wird nur einmal durchlaufen.
  - **External Event:** Einstellungen für externe Event-Signale.

Hier legen Sie fest, welches Hardware-Ereignis unter welchen Bedingungen ein Event-Signal auslöst. Die Werte können jeweils hexadezimal oder dezimal eingegeben werden.

Der Ablauf zum Auslösen eines Event-Signals ist wie folgt: Der Wert an der Hardware-Adresse **Address** wird mit der Bitmaske **Mask** Und-verknüpft und anschließend mit dem Wert **Value** über den Operator **Operation** verglichen. Wenn der Vergleich wahr ist, wird ein Event-Signal ausgelöst.

Die Hardware-Adressen sind für jede *ADwin*-Hardware unterschiedlich.

External Event		
Address:	H/D	70H
Mask:	H/D	12H
Value:	H/D	0
Operation:		> ▼

Beispiel: Die Einstellung oben maskiert den Wert der Adresse 70h mit 12h, so dass nur die Bits 2 und 5 erhalten bleiben. Wenn das Ergebnis > 0 ist, wenn also eines der beiden Bits gesetzt ist, wird der Prozess über ein Event-Signal gestartet.

- **Priority:** Priorität des Prozesses.

Stellen Sie die Priorität des Prozesses ein, mit er im System laufen soll. Weitere Informationen zu diesem Thema finden Sie in Kapitel 6.1 „Prozessverwaltung“.

- **Optimize:**

Die wahlweise einschaltbare Optimierung kann die Prozess-Ausführungszeit (je nach Programmierung) um bis zu 20% verkürzen sowie den benötigten Speicherplatz verringern. Eine höhere Einstellung unter **Level** führt zu kürzeren Ausführungszeiten.

Wenn Sie unerwartete Compiler- oder Laufzeitfehler eines Prozesses feststellen, können Sie dies in Ausnahmefällen durch die erneute Einstellung eines niedrigeren **Level** beheben.

- **Initial Processdelay:** Stellen Sie hier das Processdelay (Zykluszeit) ein, mit dem der Prozess beginnen soll.
- **Version:** Hier können Sie einen ganzzahligen Wert eingeben, um verschiedene Versionen Ihres Programms zu unterscheiden.


## Dialogfenster „Settings“

Das Dialogfenster hat mehrere Seiten, die Sie über das Baumdiagramm im linken Teilfenster anwählen:

- Editor
  - Editor - General
  - Editor - Syntax Colors
  - Editor - Print Settings
- Language
- Directories
- ADtools

### Editor - General

**Parse and Indent:** Der Editor kann den Quelltext automatisch formatieren, z.B. einrücken und die Syntax hervorheben. Hierfür ist es nötig, dass der Editor alle Quelltexte kontinuierlich durchsucht (engl.: to parse). Die gefundenen Informationen dienen auch als Basis für komfortable Funktionen wie: Befehle und Variablen automatisch vervollständigen, Deklarationen einer Datei anzeigen oder Befehlsparameter anzeigen.

Beachten Sie, dass das ständige Durchsuchen der Quelltexte auf langsamen Rechnern zu Geschwindigkeitseinbußen führen kann. 

**Parse Declarations:** Der Editor durchsucht die Quelltexte kontinuierlich. Davon abhängig ist eine Reihe komfortabler Funktionen.

**Autoindent:** Der Quelltext wird automatisch eingerückt (engl.: to indent). Die Einrückpositionen werden über `Tabsize` festgelegt. Siehe auch „Textzeilen einrücken“ auf Seite 22.

**Indent TiCoBasic sections:** Programmabschnitte werden zusätzlich um eine Stufe eingerückt.

**Smart format:** Zeilen automatisch formatieren, siehe „Automatisch formatieren“ auf Seite 21.

**Align comments at specified position:** Kommentar hinter Quellcode wird automatisch auf die angegebene `Position` gesetzt.

Beachten Sie: Mit doppelten Kommentarzeichen `' '` können Sie einen Kommentar dennoch manuell positionieren.

**Tabsize:** Geben Sie ein, um wieviele Leerzeichen ein einzelner Tabulatorsprung eine Zeile einrückt. Zum Einrücken werden grundsätzlich nur Leerzeichen verwendet.

**Show line numbers:** Im Randstreifen (engl.: gutter) links vom Quelltext werden die Zeilennummern des Quelltexts angezeigt. Siehe auch „Zu einer Programmzeile springen“ auf Seite 33.

**Column mark, visible:** An der angegebenen **Position** wird eine Linie angezeigt, so dass man eine selbst gewählte Zeilenlänge leicht einhalten kann. Auf diese Weise kann man z.B. zu lange Zeilen für den Druck vermeiden.

### Editor - Syntax Colors

Der Editor hebt die verschiedenen Syntax-Elemente farbig hervor; siehe auch Kapitel 3.3.1 „Syntax hervorheben“ auf Seite 21. Die vollständige Syntax-Hervorhebung erfordert, dass die Option **Parse Declarations** unter **Editor - General** aktiv ist.

Sie können die Hervorhebung für jedes Syntaxelement (Definition siehe Liste unten) separat einstellen:

- **Color:** Textfarbe
- **Bold:** Schriftschnitt **Fett**
- **Italic:** Schriftschnitt *Kursiv*

Der Beispielttext oberhalb wird mit den aktuellen Einstellungen formatiert.

**Set to Default** löscht alle Änderungen und setzt wieder die Standard-Einstellungen ein.

Der Editor unterscheidet folgende Syntax-Elemente:

- **TiCoBasic-Syntax** (System related):
  - TiCoBasic sections: Die Kennwörter **Init:**, **Event:** und **Finish:** für die Programmabschnitte.
  - Compiler Directives: Befehle für den Pre-Compiler wie **#Define**, die mit **#** beginnen.
  - Reserved Keywords: Die Basis-Befehle in *TiCoBasic* wie **Dim**.
  - Global Variables: Die globalen Variablen **Par\_1** ... **Par\_80** und **Data\_1** ... **DATA\_16**.
  - External Keywords: *TiCoBasic*-Befehle für den Zugriff auf Ein- und Ausgänge wie **P2\_ADC**. Diese Befehle werden in der Regel in mitgelieferten Include- oder Library-Dateien deklariert.
  - Symbols: Rechenzeichen wie Klammern, + oder =.
- **Benutzerdefiniert** (User related):
  - Defined Names: Symbolische Namen wie **myName**, die mit **#Define** deklariert sind.
  - Local Variables: Mit **Dim** deklarierte lokale Variablen wie **myVar**.
  - Sub Names: Namen (wie **mySub**) von benutzerdefinierten Modulen, die mit **Sub** oder **Lib\_Sub** deklariert sind.
  - Function Names: Namen (wie **myFunction**) von benutzerdefinierten Modulen, die mit **Function** oder **Lib\_Function** deklariert sind.
- **Sonstige** (Other):
  - Numbers: Zahlenkonstanten in dezimaler (**15**), hexadezimaler (**0Fh**) und binärer Schreibweise (**1111b**).
  - Strings: Zeichenketten in doppelten "**Hochkommas**".
- **Comments**: Kommentare nach *Rem* oder nach Kommentarzeichen **'**.
- **Standard Text**: Alle Elemente, die nicht zu anderen Gruppen gehören, z.B. ungültige Befehle wie **Eixt** (anstelle von **Exit**).

## Editor - Print Settings

Die Einstellungen betreffen den Ausdruck eines Quelltexts.

**Header** bezieht sich auf die Kopfzeile des Ausdrucks.

**Print Header**: Bei aktiver Option erscheint auf jeder Seite des Ausdrucks eine Kopfzeile.

**Header text**: Der Text der Kopfzeile.

**Layout** legt fest, welche Elemente der Bildschirmansicht in den Ausdruck übernommen werden.

**Syntax Highlight:** Bei aktiver Option erscheint die Syntax-Hervorhebung im Ausdruck.

**Color:** Bei inaktiver Option ist der Ausdruck schwarz/weiß.

**Line numbers:** Bei aktiver Option werden die Zeilennummern am linken Rand gedruckt.

**Font size:** Legt die Schriftgröße des Ausdrucks fest.

#### Language

Wählen Sie aus, in welcher Landessprache die Fehlermeldungen des Compilers und die Online-Hilfe ausgegeben werden. Zur Wahl stehen **Deutsch** und **English**.

#### Directories

Legen Sie die Verzeichnisse fest, in denen die Entwicklungsoberfläche und der Compiler nach Dateien suchen:

- **BTL Directory:** Hier sucht die Entwicklungsumgebung nach Treiberdateien für die Kommunikation mit dem *TiCo*-Prozessor..
- **Include Directory:** In diesem Verzeichnis sucht der Compiler nach Dateien `<*.Inc>`, die Sie mit **#Include** (und ohne Pfadangabe) in einen Quelltext einbinden.
- **Lib Directory:** In diesem Verzeichnis sucht der Compiler nach Library-Dateien `<*.lib>`, die Sie mit **Import** (und ohne Pfadangabe) in einen Quelltext einbinden.
- **Default working directory:** In diesem Verzeichnis sucht die Entwicklungsumgebung, wenn eine Datei oder ein Projekt geöffnet wird..

Wir empfehlen, die voreingestellten Verzeichnisse für BTL, Include und Library nicht zu verändern. Wenn Sie Include- und Library-Dateien aus anderen Verzeichnissen einbinden, können Sie im Einbinde-Befehl den vollständigen oder relativen Pfadnamen angeben.

#### ADtools

Die Hilfsprogramme *ADtools* (Beschreibung siehe Kapitel 3.11 auf Seite 71) können über eine eigene Symbolleiste aufgerufen werden. Bei aktiver Option erscheint das jeweilige Hilfsprogramm in der *ADtools*-Leiste.

## 3.8.6 Das Menü „Tools“

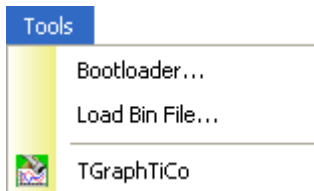
Im Menü `Tools` rufen Sie Hilfsprogramme auf.

Der Menüeintrag `Bootloader...` überträgt eine Binärdatei zum *TiCo*-Prozessor und aktiviert den *TiCo*-Bootloader. Der *TiCo*-Bootloader kann einen Prozess beim Einschalten der *ADwin*-Hardware automatisch starten. Näheres ist in Kapitel 3.6.2 „*TiCo*-Bootloader programmieren“, Seite 40 beschrieben.

Der Menüeintrag `Load Bin File...` überträgt eine vorhandene Binärdatei zum *TiCo*-Prozessor (siehe Kapitel 3.6.1 „*TiCo*-Binärdatei übertragen“, Seite 39).

Die Menüeinträge `Bootloader...` und `Load Bin File...` stehen nur zur Verfügung, wenn im Dialogfenster „Compiler Options“ die Option „Do not access the device“ deaktiviert ist (siehe Seite 46).

Der Menüeintrag `TGraphTiCo` startet ein Hilfsprogramm; Kurzbeschreibung siehe Kapitel 3.11 auf Seite 71.

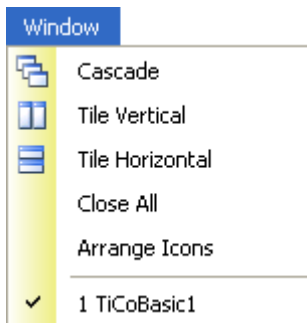


## 3.8.7 Das Menü „Window“

Mit dem Menü `Window` können Sie zwischen verschiedenen Quelltext-Fenstern umschalten und diese am Bildschirm arrangieren.

Der Menüpunkt `Arrange Icons` ordnet die Symbole verkleinerter Dateien neu an, was Sie z. B. nach einer Änderung der Bildschirmauflösung verwenden können.

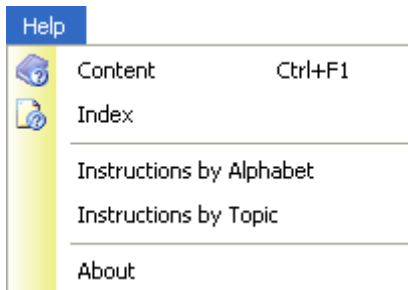
Unten im Menü können Sie über die Dateinamen einen der geöffneten Quelltexte zum aktiven Fenster machen. Der aktive Quelltext ist mit einem Haken gekennzeichnet; im Beispiel rechts ist dies `TiCoBasic1.bas`.




### 3.8.8 Das Menü „Help“

Mit dem Menü `Help` rufen Sie die Online-Hilfe der Entwicklungsumgebung auf:

- `Content`: Inhaltsverzeichnis
- `Index`: Indexverzeichnis
- `Instructions by`
  - `Alphabet`: Befehlsliste alphabetisch
  - `Topic`: Befehlsliste nach Themen.
  - `Module`: Befehlsliste nach Modulen (nur *ADwin-Pro*).



Die Befehlslisten beziehen sich auf das *ADwin*-System, das im Dialogfenster „Compiler Options“ auf Seite 46 eingestellt ist.

Alternativ können Sie auch die Schaltfläche  verwenden. Mit der Taste `F1` erhalten Sie Hilfe zu einem geöffneten Fenster oder zu einem markierten Befehlswort.

`About` öffnet ein Fenster, das die Version der Entwicklungsumgebung und den verwendeten `License key` angibt. Wenn Sie die Schaltfläche `Change License` wählen, können Sie den `License key` ändern (siehe auch, Seite 9).

Ohne Eingabe des gültigen `License key` befindet sich *TiCoBasic* im Demo-Modus. In diesem Modus ist das Arbeiten mit der Entwicklungsumgebung nur zu Prüf-, Demonstrations- und Bewertungszwecken erlaubt. Sie können beispielsweise keine Binärdateien erzeugen.



## 3.9 Fenster

Der Info-Bereich wird separat behandelt, siehe Seite 66.

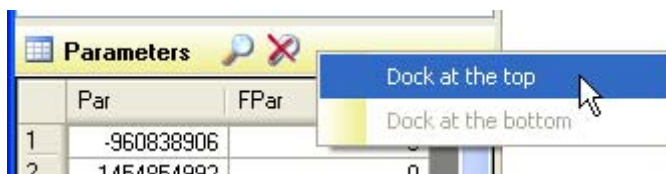
### 3.9.1 Toolbox

Die Toolbox ist der Bereich in der Oberfläche links, in dem das Projektfenster, das Parameterfenster, das Registerfenster und das Prozessfenster angezeigt werden.

Die Toolbox teilt sich in einen oberen und einen unteren Anzeigebereich. Die zugehörigen Fenster können den beiden Anzeigebereichen frei zugeordnet werden. Ein verdecktes Fenster wird durch einen Klick auf den zugehörigen Reiter in den Vordergrund geholt.

Um ein Fenster einem Bereich zuzuordnen, gehen Sie vor wie folgt:

- Öffnen Sie mit einem Rechtsklick auf die Kopfleiste des Fensters das Kontextmenü.
- Wählen Sie den Anzeigebereich oben (top) oder unten (bottom).



- Es ist möglich, alle Fenster dem gleichen Anzeigebereich zuzuordnen. Dadurch ist nur eines der Fenster im Vordergrund.

Die Standardeinstellung kann durch den Menüeintrag **View ▶ Restore default layout** wieder eingestellt werden.

Die Toolbox kann über die Symbole in der Kopfzeile als frei verschiebbares Fenster angezeigt oder ganz ausgeblendet werden.

### 3.9.2 Projektfenster

Das Projektfenster zeigt an, ob ein Projekt geöffnet und welche Quelltext- oder Include-Dateien eingebunden sind.

Das Projektfenster ist Teil der Toolbox (siehe Seite 59).

Sie können im Projektfenster folgende Aktionen ausführen:

- Eine Datei neu in das Projekt einbinden:  
Wählen Sie `Add to Project` aus dem Kontextmenü im Quelltextfenster.
- Alle offenen Dateien in das Projekt einbinden:  
Wählen Sie `Add Open Files to Project` aus dem Kontextmenü im Projektfenster.
- Eine oder mehrere Dateien aus dem Projekt löschen:  
Markieren Sie die Dateien per Mausklick im Projektfenster und
  - drücken die Taste `[ENTF]` oder
  - wählen `Remove from Project` aus dem Kontextmenü.
- Eine Datei öffnen und zum aktiven Quelltext machen:
  - Klicken Sie doppelt (linke Maustaste) auf die Datei oder
  - Markieren Sie eine Datei im Projektfenster und wählen `Open` aus dem Kontextmenü.
- Alle Dateien des Projekts speichern:  
Wählen Sie `Save all Files of Project` aus dem Kontextmenü im Projektfenster.
- Windows Explorer mit dem Pfad der markierten Datei öffnen:  
Wählen Sie `Open Path in Explorer Window` aus dem Kontextmenü im Projektfenster.

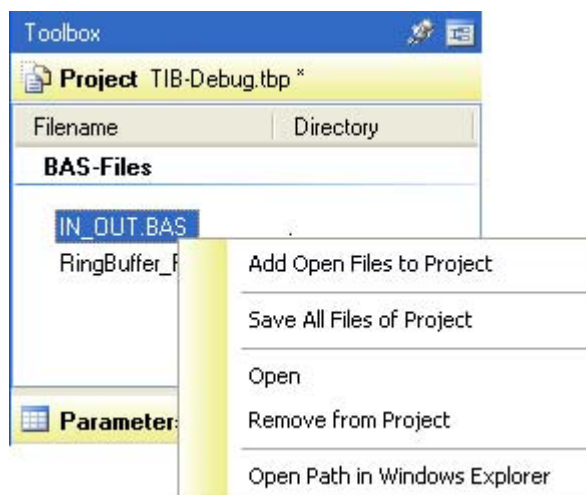



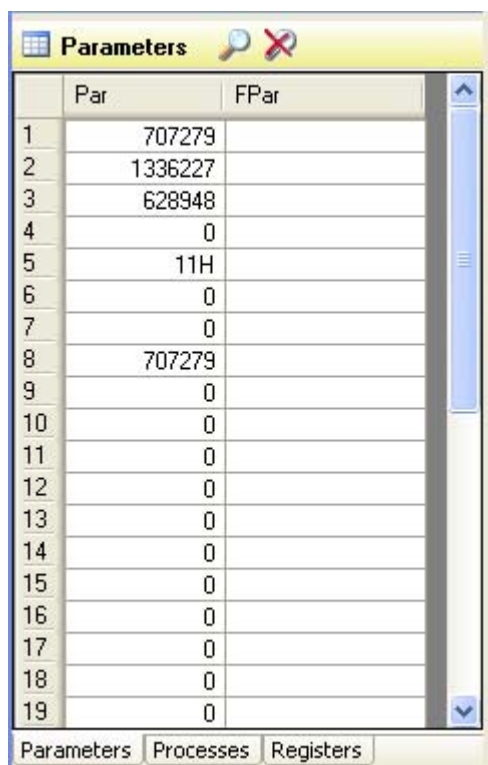
Abb. 5 – Das Projekt-Fenster mit Kontextmenü

### 3.9.3 Parameterfenster

Das Parameterfenster zeigt die globalen Parameter `Par_1...Par_80` an. Mit dem Schieberegler am rechten Rand können Sie den angezeigten Parameter-Bereich auswählen.

Das Parameterfenster ist Teil der Toolbox (siehe Seite 59).


Wenn die Kommunikation zwischen PC und ADwin-System aktiv ist (Schaltfläche `Enable Cyclic Update`  in der Werkzeugleiste), sind die Tabellenfelder weiß hinterlegt und zeigen die Werte der globalen Parameter an. Die Werte werden kontinuierlich vom System ausgelesen und angezeigt. Grau hinterlegte Felder zeigen an, dass keine Kommunikation stattfindet.



	Par	FPar
1	707279	
2	1336227	
3	628948	
4	0	
5	11H	
6	0	
7	0	
8	707279	
9	0	
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	
16	0	
17	0	
18	0	
19	0	


Abb. 6 – Das Parameter-Fenster

Sie können die Werteanzeige zwischen dezimal und hexadezimal umstellen (siehe **Par\_5** in Abb. 6), indem Sie mit der Maus auf die Nummer der betreffenden Variable klicken (links vom Tabellenfeld). Mit einem Klick auf den Spaltentitel **Par** wird die Anzeige aller Parameter **Par\_1...Par\_80** auf einmal geändert.

Mit der Schaltfläche **Scan Global Variables**  können Sie Verwendete globale Variablen und Felder anzeigen (siehe Seite 39).

### 3.9.4 Prozessfenster

Das Prozessfenster zeigt Informationen über die Prozesse auf dem *TiCo*-Prozessor an, wenn die Kommunikation zwischen PC und System aktiv ist

(Schaltfläche  in der Werkzeugleiste). Anderenfalls ist das Fenster grau hinterlegt.



Das Prozessfenster ist Teil der Toolbox (siehe Seite 59). Sie öffnen das Prozessfenster mit einem Klick auf den Reiter `Processes`.



Abb. 7 – Das Prozess-Fenster

Für jeden Prozess werden folgende Informationen angezeigt:

- Prozess-Status:
  - `running`: Prozess läuft.
  - `stopped`: Prozess wurde angehalten.
  - `---`: Prozess ist nicht vorhanden.

Sie können einen Prozess mit der Schaltfläche `Stop`  beenden und mit `Start`  wieder starten. Die Schaltflächen in der Werkzeugleiste haben die gleiche Funktion; sie beziehen sich auf den zum aktiven Quelltext gehörigen Prozess.

- Processdelay (Prozess-Zykluszeit)

Das zum aktiven Quelltext gehörige Processdelay ist auch oben in der Werkzeugleiste zu sehen.


Sie können die Zykluszeit ändern, indem Sie einen neuen Wert in das Eingabefeld schreiben. Sobald Sie das Feld verlassen, wird der Wert

zum *TiCo*-Prozessor übertragen. Achten Sie darauf, den *TiCo*-Prozessor nicht durch zu kleine Werte überlasten.

- Prozess-Priorität; die Farbe der Prozessnummer gibt die Priorität an.
  - rot = hohe Priorität
  - blau = niedrige Priorität.

Die Bedeutung und die Zeiteinheiten des Processdelay sind in Kapitel 6.2.1, Seite 109 erläutert.

## 3.9.5 Registerfenster

Das Registerfenster zeigt die Registerinhalte des TiCo-Prozessors an, wenn die Kommunikation zwischen PC und System aktiv ist (Schaltfläche  in der Werkzeugleiste). Anderenfalls sind die Felder grau hinterlegt.

Das Registerfenster ist Teil der Toolbox (siehe Seite 59). Sie öffnen das Registerfenster mit einem Klick auf den Reiter `Registers`.

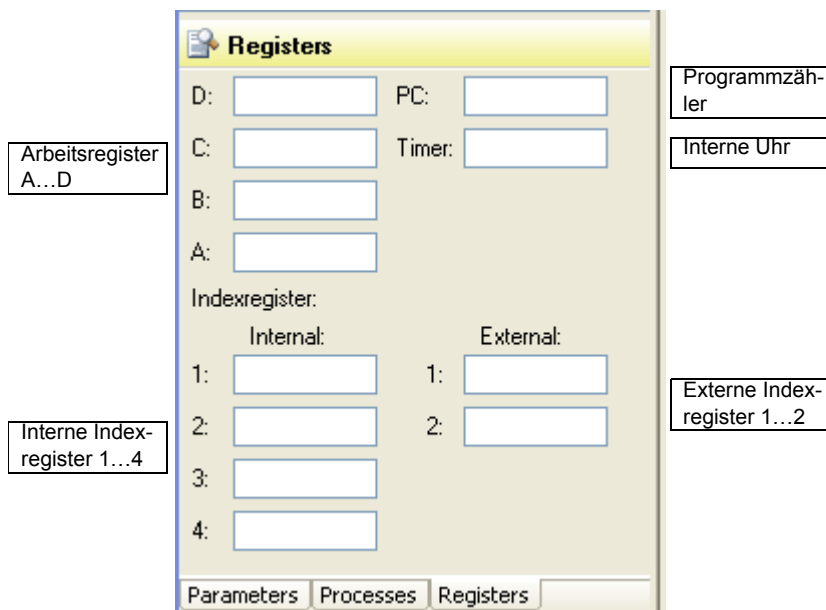
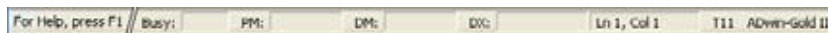


Abb. 8 – Das Register-Fenster

Die Registerinhalte sind hilfreich, wenn Sie in Ihrem Programm Assembler-Befehle verwenden. Eine Dokumentation der Assembler-Befehle ist derzeit noch nicht verfügbar.

### 3.9.6 Statusleiste

Die Statusleiste befindet sich am unteren Rand der Bedienoberfläche.



- Links: Informationen zur zuletzt ausgeführten Aktion.
- Mitte: Die aktuelle Auslastung (bei aktiver Verbindung zwischen PC und *TiCo*) und die Speichergröße des *TiCo*-Prozessors.aktiver
- Rechts: Die aktuelle Cursor-Position im Quelltextfenster (Zeile und Spalte); daneben die Einstellungen für den Compiler (Debug-Modus, Timing-Modus, Device no., Prozessor, *ADwin*-Hardware).

Die Angaben zu Prozessorauslastung und Speicherplatz bedeuten:

**Busy:** Zeitliche Auslastung des Prozessors in Prozent, berechnet als:  $\text{Rechenzeit} / (\text{Rechenzeit} + \text{Leerlaufzeit})$ .

**PM:** Verfügbarer Programmspeicher in Bytes.

**DM:** Verfügbarer interner Datenspeicher in Bytes.

**DX / SX:** Verfügbarer externer Datenspeicher in Bytes.

### 3.10 Info-Bereich

Der Info-Bereich ist der Bereich in der Oberfläche unten, in dem folgende Fenster angezeigt werden:

- Infofenster
- ToDo-Liste
- Fenster „Global Variables“
- Fenster „Declarations“



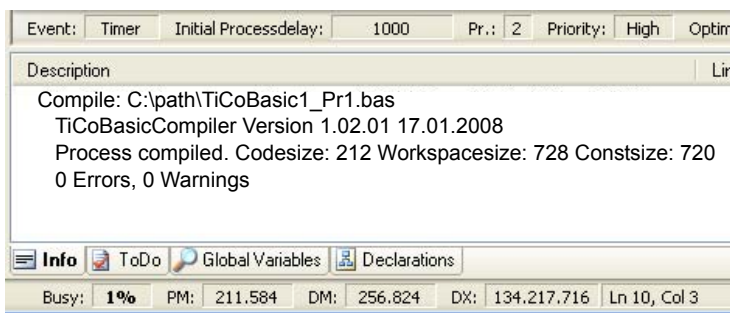
## 3.10.1 Infofenster

Im Info-Fenster werden Meldungen des Compilers zum jeweils aktiven Quelltext dargestellt:

- Statusmeldung nach dem Kompilieren
- Fehlermeldungen
- Warnungen

Warnungen und Fehlermeldungen werden mit dem Ort des Auftretens (Zeile, Dateiname und Pfad) angegeben. Ein Doppelklick auf die Meldung färbt die betreffende Quelltextzeile rot und setzt den Cursor in die Zeile.

Eine erfolgreiche Statusmeldung nach dem Kompilieren sieht z.B. folgendermaßen aus:



Die Werte der Statusmeldung geben Hinweise, wieviel Speicherplatz der Prozess benötigen wird:

- **Codesize:** Größe der erzeugten Binärdatei in Bytes; die Datei wird als Prozess im Programmspeicher (PM) abgelegt.
- **Workspacesize:** Benötigter Speicherplatz in Bytes im lokalen Datenspeicher (DM) für
  - lokale Variablen und Felder
  - interne Zwecke (2 × 4 Byte)

Darüber hinaus wird weiterer Platz im Datenspeicher benötigt, der manuell berechnet werden kann:

- Jedes globale Feld benötigt etwa vierzig Byte im lokalen Datenspeicher (für interne Zwecke).
  - Jedes Element eines globalen Felds benötigt 4 Byte (im externen Datenspeicher; nur wenn das Feld `At DM_Local` deklariert ist, liegen die Daten im lokalen Datenspeicher).
- `Constsize`: Benötigter Speicherplatz für Konstanten in Bytes.
  - `Stacksize`: Größe des internen Stapels, der für Libraries verwendet wird.

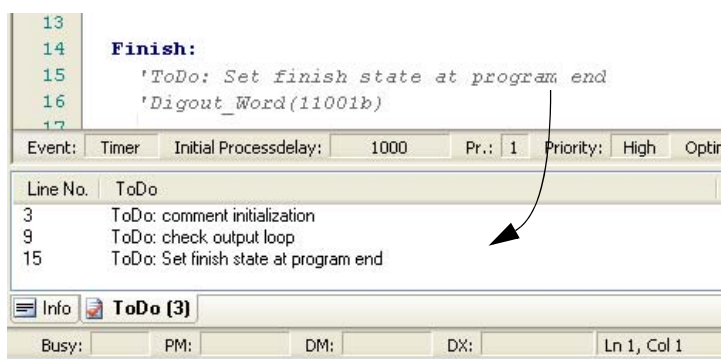
Es wird nicht angezeigt, wieviel Speicherplatz im externen Datenspeicher (DX) benötigt wird.

### 3.10.2 ToDo-Liste

Das Fenster `ToDo` dient als einfache ToDo-Liste: es werden Zeilen der aktuellen Quelltextdatei angezeigt, in denen der Text „ToDo:“ als Kommentar enthalten ist. Durch entsprechende Kommentarzeilen können Sie noch nicht erledigte Arbeiten am Quelltext markieren und übersichtlich anzeigen.

Wenn eine Aufgabe erledigt ist, löschen Sie einfach die entsprechende Kommentarzeile.


Das Fenster `ToDo` ist Teil des Info-Bereichs (siehe Seite 66).



Ein Doppelklick auf einen ToDo-Eintrag setzt den Cursor in die betreffende Zeile im Quelltext.

## 3.10.3 Fenster „Global Variables“

Das Fenster Global Variables zeigt an, welche globalen Variablen (Par\_1 ... Par\_80) und Felder (Data\_1 ... Data\_16) in einem Quelltext oder in einem Projekt verwendet werden.

Um die Anzeige zu starten oder zu aktualisieren, müssen Sie die Schaltfläche  im Parameterfenster drücken (siehe Verwendete globale Variablen und Felder anzeigen, Seite 39).

Das Fenster ist Teil des Info-Bereichs (siehe Seite 66).

Global Variable	Processfile	Line No.	Comment
Par_1	ADB-SCR-WIN-GlobalVars1.bas	4	ADB-SCR-WIN-GLOBALVARS.INC
Par_1	ADB-SCR-WIN-GlobalVars1.bas	10	
Par_1	ADB-SCR-WIN-GlobalVars1.bas	14	
Par_1	ADB-SCR-WIN-GlobalVars1.bas	16	used 2 times
Par_1	ADB-SCR-WIN-GlobalVars1.bas	17	used 2 times
Par_1	ADB-SCR-WIN-GlobalVars2.bas	8	
Par_2	ADB-SCR-WIN-GlobalVars1.bas	15	
Par_3	ADB-SCR-WIN-GlobalVars2.bas	5	
Par_3	ADB-SCR-WIN-GlobalVars2.bas	7	
Par_4	ADB-SCR-WIN-GlobalVars1.bas	2	ADB-SCR-WIN-GLOBALVARS.INC
Par_4	ADB-SCR-WIN-GlobalVars1.bas	3	ADB-SCR-WIN-GLOBALVARS.INC
Par_4	ADB-SCR-WIN-GlobalVars2.bas	6	
Par_4	ADB-SCR-WIN-GlobalVars2.bas	7	
Par_10	ADB-SCR-WIN-GlobalVars1.bas	3	ADB-SCR-WIN-GLOBALVARS.INC
Par_10	ADB-SCR-WIN-GlobalVars2.bas	7	
Data_5	ADB-SCR-WIN-GlobalVars1.bas	6	
Data_5	ADB-SCR-WIN-GlobalVars1.bas	16	
Data_5	ADB-SCR-WIN-GlobalVars2.bas	2	
Data_8	ADB-SCR-WIN-GlobalVars1.bas	5	


Sie können die Zeilen im Fenster mit einem Klick auf einen Spaltentitel sortieren.

Die Liste zeigt an:

- Name der verwendeten globalen Variablen oder des globalen Felds.
- Name der durchsuchten Datei
- Zeilennummer, wo die Variable genutzt oder aufgerufen wird.

Wenn im Kommentar ein Dateiname angegeben ist, bezieht sich die Zeilennummer auf diese Datei, sonst auf die durchsuchte Datei.

- Anmerkungen, wenn
  - die Variable mehrfach in der Zeile aufgerufen wird
  - die Variable nur indirekt aufgerufen wird.  
Dieser Fall tritt auf, wenn z. B. eine Funktion in einer Include- oder Library-Datei eine globale Variable verwendet. Der Funktionsaufruf im Quelltext führt dann indirekt zur Verwendung der Variablen, auch wenn sie in der aufrufenden Zeile nicht auftaucht.

Wenn Sie den Quelltext ändern, wird das Fenster nicht automatisch aktualisiert. Verwenden Sie dazu Schaltfläche `Scan Global Variables`  im Parameterfenster.

### 3.10.4 Fenster „Declarations“

Das Fenster `Declarations` zeigt alle zu einer Quelltextdatei gehörenden Deklarationen, Include- und Library-Dateien an. Zum Aktualisieren der Anzeige drücken Sie die Schaltfläche `Update`.

Deklarationen aus anderen als der aktiven Quelltextdatei – auch innerhalb eines Projekts – werden nicht berücksichtigt.

Das Fenster `Declarations` ist Teil des Info-Bereichs (siehe Seite 66).

Die Deklarationen werden unter verschiedenen Reitern, die den Ort der Deklaration repräsentieren, angezeigt:

- `[file].bas`: In der Quelltextdatei erstellte Deklarationen: lokale Variablen, Felder, Befehle (`Sub`, `Function`) und symbolische Namen (`#Define`).
- `System`: Systemvariablen und Befehle, die in *TiCoBasic* implementiert sind und die zu den aktuellen Compiler-Einstellungen passen.

Die globalen Variablen **PAR** werden nicht angezeigt. Beachten Sie hierzu auch das Fenster „Global Variables“ (Seite 69) und die Funktion „Verwendete globale Variablen und Felder anzeigen“ (Seite 39).

- **ADwin-Gold, ADwin-light-16:** Befehle für Hardware-Zugriffe, die in *TiCoBasic* implementiert sind und die zu den aktuellen Compiler-Einstellungen passen.
- **[file].inc:** Variablen und Befehle, die in dieser Include-Datei deklariert sind. Ein solcher Reiter taucht nur auf, wenn im Quelltext eine Include-Datei mit **#Include** eingebunden ist.
- **[file].lib:** Variablen und Befehle, die in dieser Bibliotheksdatei deklariert sind. Ein solcher Reiter taucht nur auf, wenn im Quelltext eine Bibliotheksdatei mit **Import** eingebunden ist.
- **All:** Alle gültigen Deklarationen aus den oben aufgeführten Quellen.

Sie können die Deklarationen mit einem Klick auf einen Spaltentitel sortieren. Wenn Sie die Option **Show Groups** aktivieren, werden die Deklarationen nach Gruppen sortiert.

Wenn Sie den Quelltext ändern, wird das Fenster nicht automatisch aktualisiert. Verwenden Sie dazu Schaltfläche **Update**.

Die Deklarationen können nur angezeigt werden, wenn die Option **Parse Declarations** unter **Editor - General** (siehe Seite 53) aktiv ist.

## 3.11 ADtools

*ADtools* sind kleine Hilfsprogramme, die Daten und Betriebszustände eines *ADwin*-Systems oder eines *TiCo*-Prozessors anzeigen. Sie starten *ADtools* mit einem Klick auf eine Schaltfläche in der senkrechten Leiste am rechten Rand.

Für den *TiCo*-Prozessor gibt es derzeit nur das Programm **TGraphTiCo**, mit dem Sie die Werte globaler Felder (**Data**) grafisch anzeigen können.

Jedes *ADtool* ist ein eigenständiges Windows-Programm, das Sie auch mehrfach starten können: Lassen Sie sich alle interessanten Parameter auf dem Bildschirm anzeigen. Wenn Sie eine passende Bildschirmanzeige zusammengestellt haben, können Sie die Gesamt-Konfiguration speichern und später erneut verwenden. Folgende *ADtools* stehen Ihnen zur Verfügung:



TButton

löst beim Druck auf die Schaltfläche eine definierte Aktion aus, wie z.B. Booten, Variable ändern, Prozess laden oder starten.



TProcess

zeigt Informationen der laufenden Prozesse, kann diese starten, stoppen und das Timing verändern.



ADtools

speichert und lädt eine von Ihnen erstellte Gesamt-Konfigurationen aus mehreren *ADtools*.



TGraphTiCo

stellt den Inhalt globaler Felder eines *TiCo*-Prozessors in Kurvenform dar.

Alle weiteren Informationen zu den Hilfsprogrammen entnehmen Sie bitte der Online-Hilfe, die Sie im jeweiligen Hilfsprogramm aufrufen.

## 4 Prozesse programmieren


In diesem Kapitel zeigen wir Ihnen, wie Sie ein *TiCoBasic*-Programm aufbauen, strukturieren und welche Variablen Ihnen dabei zur Verfügung stehen.

### 4.1 Programmaufbau

Sie geben ein *TiCoBasic*-Programm als ASCII-Text mit dem Editor der Entwicklungsumgebung ein; dabei verwenden Sie eine erweiterte Basic-Syntax. Diesen Quelltext übersetzt der Compiler in einen ausführbaren Prozess für den *Tico*-Prozessor.

Ein Quelltext besteht aus einer beliebigen Anzahl von Befehlszeilen, die jeweils einen Befehl oder eine Zuweisung enthalten; Ausnahme siehe . Eine Befehlszeile darf bis zu 255 (ASCII-) Zeichen enthalten; Ausnahme siehe .

*TiCoBasic* akzeptiert bei Befehlen und Variablennamen Groß- und Kleinschreibung. In unseren Beispielen verwenden wir allerdings zur besseren Unterscheidung eine einheitliche Schreibweise.

Ein Programm besteht aus bis zu 3 Abschnitten, die bei der Ausführung auf dem *Tico*-Prozessor unterschiedliche Aufgaben übernehmen, sowie aus den erforderlichen Deklarationen. Halten Sie die Reihenfolge der Abb. 9 beim Programmaufbau ein. 

Jedes Programm muss zumindest den Abschnitt **Event:** enthalten.

Die Ausnahme im Programmaufbau ist der Prozess ohne Ansteuerung (None), in dem es keine fest definierten Abschnitte gibt. Näheres siehe Seite 109.

Optional können Sie Funktionen und Unterprogramme definieren, und „Include“-Dateien und Libraries einbinden.

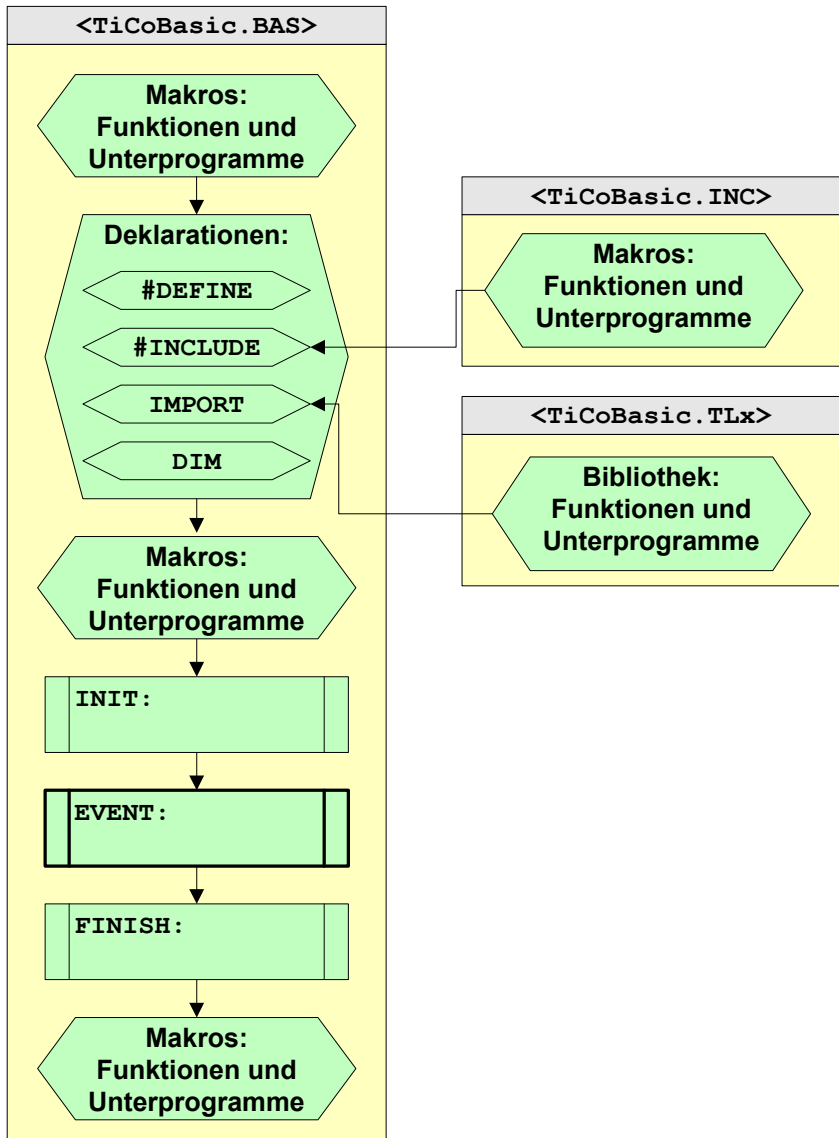


Abb. 9 – Aufbau eines *TiCoBasic*-Programms



### 4.1.1 Die Programmabschnitte

Die Programmabschnitte (siehe Abb. 9) beginnen jeweils mit einem Kennwort. Alle Abschnitte haben die für den Prozess eingestellte Priorität (Dialogfenster „Process Options“, Seite 49).

- **Init:** ist der Programmabschnitt, der bei jedem Start des Prozesses einmalig durchlaufen wird. Er dient zur Initialisierung, z.B. von Variablen oder Datenleitungen.
- **Event:** ist der zentrale Funktionsabschnitt, der (typischerweise) in regelmäßigen Abständen aufgerufen wird, bis er gestoppt wird. Je nach Einstellung wird der Aufruf durch einen zyklischen Timer-Event oder durch einen externen Event ausgelöst.
- **Finish:** wird nach dem Stoppen eines Prozesses einmalig durchlaufen und ist daher das „Gegenstück“ zur Initialisierung.

Die Abschnitte **Init:** und **Finish:** sind optional, der Abschnitt **Event:** muss immer vorhanden sein. Im Unterschied zu *ADbasic* existiert kein Abschnitt **LowInit:**.

### 4.1.2 Benutzerdefinierte Befehle und Variablen

#### Symbolische Namen

Mit der Anweisung **#Define** können Sie symbolische Namen definieren (siehe). Gruppieren Sie alle diese Definitionen am Beginn der Datei und vor dem Start der Programmabschnitte.

Symbolische Namen werden häufig für Konstanten, globale Variablen und globale Felder verwendet, aber auch für Berechnungsausdrücke.

#### Felder und lokale Variablen

In *TiCoBasic* müssen Sie nur lokale Variablen und alle Felder vor der Benutzung mit **Dim** deklarieren (siehe Seite 135). Die globalen Variablen **Par\_n** sind bereits vordefiniert und müssen nicht deklariert werden. Nach der Deklaration haben Variablen und Felder keinen definierten Inhalt, sollten also von Ihnen initialisiert werden.



Alle diese Variablen und Felder sind innerhalb des Prozesses in allen Programmabschnitten verfügbar. Auf die globalen Variablen und Felder können Sie darüber hinaus auch aus anderen Prozessen und von der *ADwin* CPU aus zugreifen, z.B. um Daten auszutauschen.

**Makros**

Makro-Funktionen `Function ... EndFunction` und -Unterprogramme `Sub ... EndSub` werden bei einem Aufruf im Programmtext an der aufrufenden Stelle eingefügt (siehe auch Kapitel 4.5.1 auf Seite 95). Makros müssen in jedem Fall außerhalb der Programmabschnitte definiert werden (siehe Abb. 9 auf Seite 74).

**Libraries**

Das Einbinden von Libraries muss vor dem Beginn der Programmabschnitte erfolgen. Library-Funktionen `Lib_Function ... Lib_EndFunction` und -Unterprogramme `Lib_Sub ... Lib_EndSub` sind Programm-Module mit geringerem Speicherbedarf (bei mehrfachem Aufruf) als die o.g. Makro-Funktionen und -Unterprogramme (siehe auch Kapitel 4.5.3 auf Seite 96).

## 4.2 Variablen und Felder

### 4.2.1 Übersicht

Datenstruktur	Name	Datentyp	Bemerkung
Globale Variablen und Felder			
Variable (Skalar)	<b>Par_1...Par_80</b>	Long	Vordefiniert, nicht deklarierbar
System-Variable	<b>Processdelay</b>	Long	
	<b>Processn_running</b>	Long	
Eindimensiona- les Feld (Vektor)	<b>Data_1[]...</b> <b>Data_16[]</b>	Long, RingBuffer	Name <b>Data_</b> nicht änderbar, nur Deklara- tion von Feldnummer und Dimension.
Lokale Variablen und Felder			
Variable (Skalar)	frei wählbar	Long	muss deklariert wer- den
Eindimensiona- les Feld (Vektor)	frei wählbar	Long	muss deklariert wer- den

Wenn Sie bei der Deklaration den Speicherbereich nicht explizit festlegen, werden Variablen und Felder standardmäßig im internen Speicher DM angelegt (Speicherbelegung siehe Kapitel 4.3.1).

Der Datentyp **Long** hat eine Länge von 32 Bit.

### 4.2.2 Datenstrukturen

In *TiCoBasic* stehen Ihnen vor allem 2 Datenstrukturen zur Verfügung:

- Variablen (Skalare)

**VAR**

Eine Variable kann einen einzelnen Wert enthalten.

- Eindimensionale Felder.

**ARRAY**

Ein Feld besteht aus einer frei definierbaren Zahl von Feldelementen, die je einen Wert enthalten können.

Sie können eindimensionale globale Felder **Data\_n** auch als FIFO verwenden (Ringspeicher nach dem Prinzip: First in, first out, siehe Kapitel 4.3.3 auf Seite 85).

Die maximale Anzahl an Variablen bzw. die Größe eines Felds ist nur durch die Speichergröße des *TiCo*-Prozessors begrenzt.

Der Compiler unterscheidet

- Globale Variablen (Parameter) und Globale Felder (Arrays):

Auf globale Datenstrukturen können sowohl alle *TiCo*-Prozesse als auch die *ADwin* CPU zugreifen, z. B. zum Austausch von Daten.

System-Variablen zählen zu den globalen Variablen (siehe Seite 81).

- Lokale Variablen und Felder (siehe Seite 82):

Lokale Datenstrukturen sind nur innerhalb des Prozesses verfügbar, in dem sie deklariert wurden. In gleicher Weise können Sie lokale Datenstrukturen innerhalb einer Funktion oder eines Unterprogramms deklarieren.

Sie deklarieren Variablen und Felder mit der Anweisung `Dim`; dadurch wird der Datentyp bestimmt, der erforderliche Platz im Speicher belegt und der Speicherplatz dem Variablennamen fest zugeordnet.

Zur Vereinfachung für Sie sind die globalen Variablen `Par_1 ... Par_80` bereits vordefiniert; Sie müssen (und können) diese Variablen also nicht deklarieren.

Die Deklaration globaler Felder erkennt der Compiler am Namen „`Data_n`“, wobei „`Data_`“ ein festgelegter Text ist und „`n`“ die von Ihnen festgelegte Nummer des Felds (1...16).



Variablen und Feldelemente haben nach der Deklaration keinen definierten Wert und sollten deshalb mit einem sinnvollen Wert (z. B. Null) initialisiert werden. Ausnahme: Mit dem Übertragen eines Prozesses auf den *TiCo*-Prozessor werden die globalen Variablen `Par_1 ... Par_80` automatisch mit Null initialisiert.

### 4.2.3 Datentypen

Bei der Deklaration von Variablen und Feldern muss der Datentyp angegeben werden.

Der Compiler verarbeitet nur den Datentyp `Long`. Das sind ganzzahlige 32 Bit-Werte mit dem Wertebereich:

$$-2147483648 \dots +2147483647 = -2^{31} \dots +2^{31}-1$$

Im nächsten Abschnitt ist dargestellt, mit welchen Schreibweisen Sie einen Zahlenwert eingeben können.

### 4.2.4 Zahlenwerte eingeben

Sie können 4 verschiedene Schreibweisen benutzen, wenn Sie einen Zahlenwert angegeben möchten. Die folgenden Beispiele weisen einer Variablen `x` den Wert 930 zu.

1. Dezimale Schreibweise: `x = 930`
2. Exponential-Schreibweise: `x = 93E1`

Hierbei steht „93E1“ für  $93 \times 10^1$ , d.h. nach dem „E“ folgt der (max. 2-stellige) Exponent zur Basis 10.

3. Binäre Schreibweise (angehängtes „b“): `x = 111010001b`.
4. Hexadezimale Schreibweise (angehängtes „h“): `x = 3A2h`.

Wenn der hexadezimale Wert mit einem Buchstaben (**A - F**) beginnt, müssen Sie eine Null voranstellen: Anstelle von „F6h“ also „0F6h“. Anderenfalls interpretiert der Compiler ihren Wert als den Namen einer lokalen Variablen.

### 4.2.5 Globale Variablen (Parameter)

Auf globale Variablen (und Felder) können alle laufenden *TiCo*-Prozesse und die *ADwin* CPU zugreifen; daher eignen sie sich gut zum Datenaustausch zwischen den Prozessen oder zwischen den Prozessen und der *ADwin* CPU (siehe auch Kapitel 6.3.1 „Datenaustausch zwischen Prozessen“). Ihnen stehen 80 ganzzahlige Variablen sowie bis zu 16 Felder (Arrays) vom Datentyp *Long* zur Verfügung. Alle Variablen und Feldelemente haben eine Länge von 32 Bit.

Die ebenfalls global verfügbaren System-Variablen sind auf Seite 81 beschrieben.

Sie können die globalen Variablen in Ihren Programmen an beliebiger Stelle verwenden, ohne sie zu deklarieren. Die Variablen haben jedoch keinen definierten Wert und sollten deshalb mit einem sinnvollen Wert (z.B. Null) initialisiert werden. Ausnahme: Mit dem Übertragen eines Prozesses auf den *TiCo*-Prozessor werden die globalen Variablen `Par_1` ... `Par_80` automatisch mit Null initialisiert.

Die globalen Variablen werden auch als Parameter bezeichnet und haben die Namen `Par_1`, `Par_2`, ..., `Par_80` mit dem Datentyp *Long* für ganzzahlige 32Bit-Werte.

**.Beispiel**

```

Par_5 = 700
Par_72 = ADC(1)

```

'Parameter 5 erhält den  
'Wert 700.  
'Die Spannung am analogen  
'Eingang 1  
'wird gemessen und in  
'Parameter 72  
'abgelegt.



Im Gegensatz zu den sonstigen Variablen dürfen Sie die globalen Variablen **Par\_n** nicht deklarieren, da sie vordefiniert und dem Compiler bereits bekannt sind.

**4.2.6 Globale Felder (Arrays)**

Die globalen Felder ermöglichen Ihnen, große Datenmengen zwischen den Prozessen auf dem ADwin-System oder der ADwin CPU auszutauschen (siehe auch Kapitel 6.3.1 „Datenaustausch zwischen Prozessen“). Ihnen stehen bis zu 16 globale Felder (Arrays) vom Datentyp **Long** zur Verfügung.



Da Größe und Datentyp wählbar sind, müssen Sie globale Felder am Anfang Ihres Programms deklarieren (siehe **Dim**) und möglichst auch initialisieren (die Feldelemente haben sonst keinen definierten Wert).

Die Deklaration eines globalen Felds erkennt der Compiler am Namen „**Data\_n**“, wobei „**Data\_**“ ein festgelegter Text ist und „**n**“ die von Ihnen festgelegte Nummer des Felds (1...16). Die Namen für **Data**-Felder sind also:

**Data\_1, Data\_2, ..., Data\_16.**

Andere Feldnummern sind unzulässig. Sie können jedoch die Feldnummern frei wählen, auch die Deklaration von z. B. **Data\_5** (ohne **Data\_1 ... Data\_4**) ist gültig. In Ihrem Programm werden die Felder vom Compiler anhand ihrer Nummer unterschieden.

**Beispiel**

```

REM Feld 5 mit 20000 Elementen vom Typ Long deklarieren.
Dim Data_5[20000] As Long

```

Die maximale Größe der Felder richtet sich nur nach dem verfügbaren Speicherplatz. Beispielsweise kann auf einem *TiCo*-Prozessor mit 256 MiB Speicher ein Feld mit über 67 Millionen Elementen vom Typ **Long** deklariert werden.

Nachdem das Feld deklariert ist, können Sie auf jedes einzelne Element zugreifen. Das erste Element eines Felds besitzt den Index 1.

Weisen Sie *auf keinen Fall* dem Element 0 eines Felds einen Wert zu, z.B. mit `Data_1[0] = ...`



### Beispiel



```
Rem Der globalen, ganzzahligen Variablen Par_1 wird der
Wert
Rem des 200. Elements aus dem Feld 5 zugewiesen.
Par_1 = Data_5[200]

Rem Durch diese Anweisung erhält das 345. Element aus
dem Feld
Rem Data_5 den Wert 4000.
Data_5[345] = 4000
```

Sie können den Index eines *Feldelements* auch über eine Variable übergeben:

```
Rem Auch hier wird, wie im Beispiel davor, dem 345.
Element des
Rem Felds Data_5 der Wert 4000 zugewiesen.
nummer1 = 345
Data_5[nummer1] = 4000
```

Dagegen darf die Nummer eines *Felds* nicht durch eine Variable übergeben werden. Die folgende Anweisung führt zu einer Fehlermeldung des *TiCo-Basic*-Compilers:



```
num = 2
Data_num[300] = 20      'FALSCH !!
Data_2[300] = 20      'RICHTIG
```

Der Compiler interpretiert `Data_num` als Namen eines lokalen Felds, das (wahrscheinlich) nicht deklariert wurde und daher nicht verfügbar ist. Verwenden Sie statt dessen die Schreibweise `Data_2`. Beachten Sie die unterschiedliche Syntax-Hervorhebung bei den Variablen.

### 4.2.7 System-Variablen

Um Informationen über den Status des *TiCo*-Prozessors zu erhalten, stehen Ihnen die folgenden System-Variablen zur Verfügung. Diese Variablen sind global, d.h. für alle *TiCo*-Prozesse und von der *ADwin* CPU aus verfügbar. Weitere Informationen finden Sie bei den Befehlsbeschreibungen.

**Process<sub>n</sub>\_Running**

Zeigt den Status des Prozesses `n` an (mit `n = 1...10`), d. h. ob der Prozess läuft, gerade angehalten wird oder gestoppt ist (siehe Seite 175). Die Variable kann nur gelesen werden.

, siehe Seite 237 **Processdelay**

Der Soll-Zeitabstand, in dem zeitgesteuerte Prozesse vom Zähler aufgerufen werden, ist das **Processdelay**, auch Zykluszeit genannt. Mit der Systemvariablen **Processdelay** (siehe auch Seite 173) können Sie die Zykluszeit abfragen oder einstellen. Die Zykluszeit wird in Taktzyklen des Zählers gemessen.

Sie können die Variable **Processdelay** nur innerhalb der Abschnitte **Init:** und **Event:** lesen und beschreiben. Das Beschreiben der Variablen ist jedoch nur 1mal pro Abschnitt erlaubt.



Achten Sie darauf, dass die Auslastung des Prozessors möglichst weniger als 90% beträgt, keinesfalls aber 100% übersteigen darf.

#### 4.2.8 Lokale Variablen und Felder



Alle lokalen Variablen und Felder, die Sie für Ihren Prozess benötigen, müssen Sie vor dem Beginn des ersten Abschnitts in Ihrem *TiCoBasic*-Programm deklarieren und möglichst auch initialisieren (sie haben sonst keinen definierten Wert).

Namen müssen mit einem Buchstaben beginnen und dürfen nur aus Buchstaben (a-z, A-Z), Ziffern (0-9) und dem Zeichen „\_“ (Underscore) bestehen. Umlaute (ä, ö, ü) sind nicht erlaubt, Groß- und Kleinschreibung wird nicht unterschieden. Die Länge von Variablennamen ist nur begrenzt durch die max. Zeilenlänge (255 Zeichen).

Bei (skalaren) Variablen sind als Datentypen ganzzahlige Werte (**Long**) verfügbar, in 32 Bit.



#### Beispiel

```
Dim wert As Long      'Definiert die Variable
                        ''wert'
                        'mit dem Datentyp Long
```

Variablen können Sie nicht nur als skalare Größe, sondern auch als eindimensionales Feld deklarieren, das heißt, Sie können Felder von Variablen erzeugen und verarbeiten. Die Anzahl der zu dimensionierenden Elemente im Feld wird in eckigen Klammern nach dem Namen eingegeben.



### Beispiel

```
Dim wert[100] As Long 'Definiert ein Feld der
                      'Länge 100 mit Namen 'wert'
                      'und dem Datentyp Long
```



Das erste Element eines Felds besitzt den Index 1, im Beispiel: `wert[1]`. Auf das Element mit dem Index 0 dürfen Sie nicht zugreifen.



## 4.3 Variablen und Felder – Details

### 4.3.1 Variablen und Felder im Datenspeicher

Sie können für Felder und lokale Variablen explizit festlegen, in welchem Speicherbereich (siehe unten) sie angelegt werden. Diese Festlegung geschieht bei der Deklaration mit `Dim` im Quelltext durch die Zusätze `At Dm_Local` oder `At DAm_Extern`.

Wenn Sie bei der Deklaration keinen Zusatz verwenden, werden Variablen und Felder im internen Speicher DM angelegt

Wie empfehlen Ihnen die Verwendung des internen Speichers für Variablen und (kleine) Felder, auf die Sie sehr schnell zugreifen möchten. Der langsamere externe Speicher ist – falls vorhanden – wegen seiner Größe vorwiegend für Felder geeignet.

In Abb. 10 sehen Sie Beispiele für Deklarationen, um Variablen und Felder in den verschiedenen Speicherbereichen anzulegen.

Variable / Feld	Speicherbereich	Deklaration im Quelltext
Lokale Variable	intern (DM)	<code>Dim var As Long</code> oder <code>Dim var As Long At DM_Local</code>
	extern (DX)	<code>Dim var As Long At DAm_Extern</code>
Feld (global oder lokal)	intern (DM)	<code>Dim array[5] As Long At DM_Local</code>
	extern (DX)	<code>Dim array[5] As Long</code> oder <code>Dim array[5] As Long At DAm_Extern</code>

Abb. 10 – Festlegung des Speicherbereichs bei Deklarationen

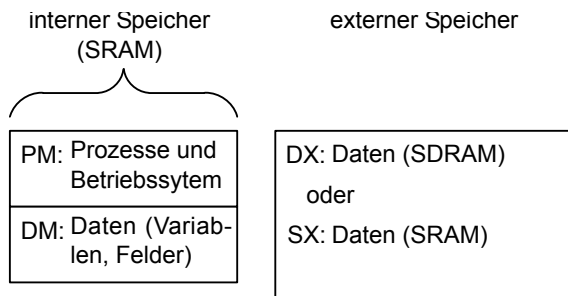


Die globalen Variablen **Par\_1...Par\_80** sind vordefiniert und stehen immer im internen Speicher DM zur Verfügung. Sie brauchen und können diese daher nicht neu (z.B. für den externen Speicher) deklarieren.

### 4.3.2 Speicherbereiche

Der *TiCo*-Prozessor verwendet einen schnellen internen Speicher (SRAM) und – falls vorhanden – einen großen externen Speicher (SDRAM). Manche Pro II-Module besitzen anstelle des externen SDRAM ein schnelleres SRAM (SX).

Je die Hälfte des internen Speichers steht als Programmspeicher PM und als Datenspeicher DM zur Verfügung.



#### – Programmspeicher (PM)

Der Programmspeicher belegt die Hälfte des internen SRAM und nimmt das Betriebssystem und Ihre Prozesse auf.

#### – Datenspeicher intern (DM)

Der interne Datenspeicher belegt die Hälfte des internen SRAM und nimmt die globalen und lokalen Variablen und Felder auf ).

#### – Externer Speicher (DX, SX)

Der externe Speicher belegt das externe SDRAM.

Bei manchen Pro II-Modulen (z.B. Pro II-MIO-TiCo) ist anstelle des SDRAM ein SRAM-Speicher eingebaut.



Der Zugriff auf den externen Speicher ist immer mit einer – noch dazu schwankenden – Wartezeit (Jitter) verbunden; ausgenommen davon ist der Zugriff über Die Datenstruktur RingBuffer (siehe Seite 85). Beachten Sie hierzu Kapitel 5.2.6 auf Seite 103.

Sie können auf Daten im internen Speicher DM deutlich schneller zugreifen als auf Daten im externen Speicher DX. Bei externem SRAM und internem Speicher ist der Zugriff etwa gleich schnell.

Die Speichergröße (SRAM, SDRAM) ist eine Bestelloption und kann nicht nachträglich vergrößert werden.

Die Größe der Speicherbereiche ist der einzige Faktor, der die Größe von Prozessen und die Zahl der deklarierbaren Variablen und Felder begrenzt (indirekt auch die Größe der Quelltext-Dateien). Sie sehen in der Statusleiste der Entwicklungsumgebung, wieviel Speicher Ihnen in PM, DM und DX insgesamt zur Verfügung steht (angegeben in Bytes).

### 4.3.3 Die Datenstruktur RingBuffer

Um große Datenmengen kontinuierlich und schnell zu übertragen, empfehlen wir globale Felder `Data_n` mit der Datenstruktur `RingBuffer`: Ein Ringspeicher, der nach dem Prinzip „First In, First Out“ verwaltet wird.

Ein `Ringbuffer` ist nicht zu verwechseln mit der Datenstruktur FIFO der ADwin CPU. Der FIFO-Ringspeicher ist im Handbuch *ADbasic* beschrieben.



Ringspeicher sind für verschiedene Anwendungen sinnvoll; allerdings schließen sich die Anwendungen gegenseitig aus:

- Der *TiCo*-Prozess greift auf Daten im externen DRAM zu und zwar lesend wie schreibend über den gleichen Ringspeicher.
- *ADbasic*-Prozesse (auf der ADwin CPU) und *TiCoBasic*-Prozesse tauschen über einen Ringspeicher miteinander Daten aus. Für jede Übertragungsrichtung ist ein separater Ringspeicher erforderlich.
- Mehrere Prozesse auf dem *TiCo*-Prozessor tauschen über einen Ringspeicher miteinander Daten aus. Es sind 2 Ringspeicher erforderlich, einer zum Schreiben und einer zum Lesen.

Der Umgang mit der Datenstruktur `RingBuffer` ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung der Datenstruktur `RingBuffer` ist daher erfahrenen Benutzern von *ADbasic* und *TiCoBasic* vorbehalten.



### Wie arbeitet der Ringspeicher?

In einem Ringspeicher werden die Daten auf besondere Weise verwaltet. Sie können sich die Daten als eine Kette vorstellen, an deren Ende Sie neue Daten einzeln anhängen und an deren Spitze Sie einzelne Daten abholen können. Sie greifen also – im Unterschied zu einem „einfachen“ Feld – nicht auf beliebige Feldelemente zu, sondern immer nur auf das erste oder letzte

(über je einen Datenzeiger). Dadurch lesen Sie die Daten in der gleichen Reihenfolge aus, wie sie in das Feld geschrieben wurden (=First In, First Out).

Da ein RingBuffer-Feld eine endliche (von Ihnen deklarierte) Zahl von Elementen besitzt, bildet die Kette aus benutzten und unbenutzten Feldelementen einen Ring, den Ringspeicher. Die Datenzeiger auf das erste und das letzte benutzte Feldelement werden automatisch verwaltet, wenn Sie dem Feld einen neuen Wert zuweisen oder einen Wert auslesen.



Aus der Ringstruktur des RingBuffer-Felds ist ersichtlich, dass die Spitze der Datenkette das Datenende „überholen“ kann. Dies ist möglich, wenn Sie Daten schneller in den RingBuffer schreiben als Sie sie auslesen. Dadurch werden alte gespeicherte Daten überschrieben und gehen somit verloren.

### Ringspeicher deklarieren und anwenden

Ein Ringspeicher wird mit `Dim` deklariert:

```

Rem Lese-Ringspeicher mit 103 Elementen im externen
  Speicher
Dim Data_1[103] As Long As RingBuffer_For_Read At
  DRAM_Extern
Rem Schreib-Ringspeicher mit 1000 Elementen
Rem im internen Speicher
Dim Data_2[1000] As Long As RingBuffer_For_Write At
  DM_Local
Rem Lese- und Schreib-Ringspeicher mit 199 Elementen im
Rem externen Speicher
Dim Data_3[199] As Long As RingBuffer_For_Read At
  DRAM_Extern
Dim Data_3[199] As Long As RingBuffer_For_Write At
  DRAM_Extern

```

Zu Feldgrößen im externen Speicher beachten Sie bitte Seite 178.

Wenn kein Speicherbereich angegeben wird, verwendet der Compiler die Voreinstellung `DM_Local`. Wir empfehlen als Erinnerungstütze, den Speicherbereich bei der Deklaration immer anzugeben.



Beachten Sie bitte, dass Sie ein globales Feld `Data` nicht gleichzeitig als „einfaches“ Feld und als Ringspeicher verwenden können.

Auf ein RingBuffer-Feld greifen Sie zu, indem Sie dessen Feldnamen (mit der entsprechenden Feldnummer) angeben.

## Beispiel



```
Dim Data_5[1000] As Long As RingBuffer_For_Read At
    DM_Local
Dim Data_5[1000] As Long As RingBuffer_For_Write At
    DM_Local
Data_5 = 95                                     'Schreibt in den
                                                'RingBuffer mit der
                                                'Nummer 5 den Wert 95.
Par_7 = Data_5                                 'Liest einen Wert aus dem
                                                'RingBuffer
                                                'und speichert ihn in der
                                                'globalen
                                                'Variablen Par_7
```

Um sicherzustellen, dass noch Platz im RingBuffer ist, sollten Sie vor dem Schreiben die Funktion `RingBuffer_Empty` verwenden. In gleicher Weise prüfen Sie mit der Funktion `RingBuffer_Full` vor dem Lesen, ob noch nicht gelesene Werte vorhanden sind (siehe Beispiel unter „Auf externes DRAM zugreifen“).

In Bezug auf die folgenden Regeln bildet der externe Speicher `SRAM_Extern` (SX) mit dem internen Speicher `DM_Local` einen gemeinsamen Speicherbereich. Auf bestimmten Pro II-Modulen ersetzt `SRAM_Extern` den externen Speicher `DRAM_Extern`.

Allgemeine Regeln zur Deklaration von Ringspeichern:

- Für jeden Speicherbereich sind 2 Ringspeicher-Deklarationen erlaubt.
- Im externen Speicher `DRAM_Extern` ist je ein Ringspeicher zum Lesen und ein Ringspeicher zum Schreiben erlaubt.

Im Speicherbereich `DM_Local` + `SRAM_Extern` sind Kombinationen von Lese- und Schreib-Ringspeichern möglich. Sie können also auch 2 Lese-Ringspeicher oder 2 Schreib-Ringspeicher deklarieren.

- Es ist nicht erlaubt, im externen Speicher `DRAM_Extern` neben einem Ringspeicher auch normale Felder zu deklarieren.

**Beispiel**

```
Rem 2 Ringspeicher im externen Speicher; damit sind
normale
Rem Felder im externen Speicher nicht mehr erlaubt!
Dim Data_5[199] as long as Ringbuffer_For_Read at
  dram_extern
Dim Data_5[199] as long as Ringbuffer_For_Write at
  dram_extern

Rem 2 Ringspeicher im internen Speicher
Rem Normale Felder sind außerdem möglich
Dim Data_1[200] as long as Ringbuffer_For_Read at
  dm_local
Dim Data_2[200] as long as Ringbuffer_For_Read at
  dm_local
Dim Data_3[200] as long at dm_local
```

**Auf externes DRAM zugreifen**

Der Zugriff auf globale und lokale Felder im externen Speicher ist relativ langsam. Ein schneller Datenaustausch ist dagegen mit einem Ringspeicher möglich. Hierbei schreibt und liest der *TiCo*-Prozess die Daten über den gleichen Ringspeicher.

### Beispiel



```
Rem Schreib- und Lese-Ringspeicher im externen Speicher  
Rem Damit sind normale Felder im DRAM_Extern nicht mehr  
Rem erlaubt!
```

```
Dim Data_5[199] as long as ringbuffer_for_read at  
    dram_external  
Dim Data_5[199] as long as ringbuffer_for_write at  
    dram_external  
Dim free,used,value1 As Long
```

#### Init:

```
Rem Lese-Ringspeicher Data_5 initialisieren  
RingBuffer_Clear(5)
```

#### Event:

```
Rem Sind noch Elemente zum Beschreiben frei?  
free = ringbuffer_Empty(5,0)  
If (free > 0) Then  
    Data_5 = value1  
EndIf  
Rem Können noch benutzte Elemente gelesen werden?  
used = ringbuffer_Full(5,0)  
If (used > 0) Then  
    Par_7 = Data_5  
EndIf
```

Nach der Deklaration eines Lese-Ringspeichers im externen Speicher sollten Sie den Ringspeicher mit dem Befehl **RingBuffer\_Clear** initialisieren.

### Datenaustausch zwischen *TiCo*-Prozessen

Zwei *TiCo*-Prozesse in einem Projekt (siehe auch Kapitel 6.3.1 auf Seite 112) können über einen Ringspeicher kontinuierlich und schnell miteinander Daten austauschen. Der Ringspeicher kann dazu im (kleineren) internen Speicher oder im (langsameren) externen Speicher liegen.

Der Datenaustausch arbeitet nur korrekt, wenn der Datenfluss eindeutig ist, also nur der eine Prozess in den Ringspeicher schreibt und nur der andere Prozess daraus liest. Dabei ist auch eine Umkehr des Datenflusses möglich, solange der Datenfluss eindeutige bleibt.

Beachten Sie: Die Deklaration eines Ringspeichers gilt für das gesamte Projekt und darf daher nur in einem der Quelltexte stehen. Dennoch können alle Prozesse des Projekts auf den deklarierten Ringspeicher zugreifen.



### Beispiel

Prozess 1, der Daten schreibt:

```

Rem Schreib- und Lese-Ringspeicher im internen Speicher
Dim Data_5[500] as long as ringbuffer_for_read at
  dm_local
Dim Data_5[500] as long as ringbuffer_for_write at
  dm_local
Dim free,value1 As Long

```

#### Init:

```

Rem Lese-Ringspeicher Data_5 initialisieren
RingBuffer_Clear(5)

```

#### Event:

```

Rem Sind noch Elemente zum Beschreiben frei?
free = ringbuffer_Empty(5,0)
If (free > 0) Then
  Data_5 = value1
EndIf

```

Prozess 2, der Daten liest:

```

Rem Ringspeicher dürfen nicht mehr deklariert werden!
Dim used As Long

```

#### Event:

```

Rem Ringspeicher wird verwendet, auch wenn er im
Rem Quelltext von Prozess 2 nicht deklariert ist:
Rem Können noch benutzte Elemente gelesen werden?
used = ringbuffer_Full(5,0)
If (used > 0) Then
  Par_7 = Data_5
EndIf

```

### Datenaustausch mit ADbasic-Prozessen

ADbasic-Prozesse (auf der ADwin CPU) und TiCoBasic-Prozesse können über einen Ringspeicher kontinuierlich und schnell miteinander Daten austauschen. Der Ringspeicher kann dazu im (kleineren) internen Speicher oder im (langsameren) externen Speicher liegen.

Für jede Übertragungsrichtung ist jeweils ein eigener Ringspeicher erforderlich, der nur in TiCoBasic deklariert wird. Der Datenaustausch arbeitet nur korrekt, wenn der Datenfluss eindeutig ist, also nur der eine Prozess in den



Ringspeicher schreibt und nur der andere Prozess daraus liest. Eine Umkehr des Datenflusses ist nicht möglich.

Die Datenübertragung verwendet eine globale Variable **Par\_n** des *TiCo*-Prozessors zur Synchronisation. Hierzu trägt der *ADbasic*-Befehl den aktuellen Wert des Schreib- oder Lesezeigers – je nach Richtung des Datenflusses – in die Variable ein, so dass in *TiCoBasic* bekannt ist, wieviele Daten gelesen oder geschrieben werden können.

### Beispiel



*TiCoBasic*-Prozess, der Daten schreibt

```
Rem Schreib-Ringspeicher im internen Speicher
Dim Data_1[500] as long as ringbuffer_for_Write at
    dm_local
Dim free,used,value1 As Long

Init:
    Rem Schreib-Ringspeicher Data_1 initialisieren
    RingBuffer_Clear(1)

Event:
    Rem Sind noch Elemente im Data_1 zum Beschreiben frei?
    Rem Par_5 enthält die Anzahl der freien Elemente und
    wird von
    Rem ADbasic aus gesetzt
    free = ringbuffer_Empty(1,5)
    If (free > 0) Then
        Data_5 = wert1
    EndIf
```

ADbasic-Prozess, der Daten liest (hier für ADwin-Gold II)

```
#Include ADwinGoldII.inc
Dim Data_10[300] As Long 'array for read data
REM define settings array for TiCo
Dim tset[150] As Long 'settings array for TiCo
Dim val As Long 'error code

Init:
    Rem Datenübertragung zum TiCo-Prozessor 1
    initialisieren
    TDrv_Init(1, tset)

Event:
    Rem Bis zu 250 Werte aus dem TiCo-Feld Data_1 lesen und
    in
    Rem Data_10 speichern. Die Anzahl der freien Elemente
    Rem wird in den TiCo-Parameter Par_5 geschrieben.
    val = Get_TiCo_RingBuffer(tset, 1, Data_10, 1, 250, 1,
    5, 0)
    If (val > 0)
        Rem Daten wurden gelesen, Daten weiter verarbeiten
    EndIf
```

## 4.4 Berechnungsausdrücke

Ein Berechnungsausdruck ist das, was Sie einer Variablen zuweisen oder einem Befehl als Argument übergeben. Er besteht aus einer beliebigen Kombination von:

- einfachen Daten: Konstante, Variable oder Feldelement
- Operatoren, die auf Argumente angewendet werden, die wieder Berechnungsausdrücke sind.

### 4.4.1 Auswertung von Operatoren

Für die Auswertung eines Berechnungsausdrucks (Definition siehe Kapitel 4.4) ist es wesentlich, in welcher Reihenfolge die Operatoren angewendet werden. Hierzu werden die Operatoren in Kategorien eingeteilt, die nach Prioritäten geordnet sind: Eine Kategorie höherer Priorität wird vor einer Kategorie niedriger Priorität bearbeitet (siehe Abb. 11).

Operator	Kategorie
" "	Begrenzer von Zeichenketten
Kennwort in <i>TiCoBasic</i>	Befehl, Funktion, Variable, etc.
=	Zuweisung
( )	Klammern
-	Vorzeichen einer <i>Konstanten</i>
* /	Punkt-Operatoren
+ -	Strich-Operatoren
And Or XOr	Binär-Operatoren
< > =	Vergleichs-Operatoren
And Or	Boolesche Operatoren

Abb. 11 – Prioritäten von Operatoren-Kategorien  
(von oben nach unten absteigende Priorität)

### Beispiel

```
var = Par_1 + Par_2 * Par_1^3 / 4
```

entspricht

```
var = Par_1 + (Par_2 * (Par_1^3) / 4)
```



Wenn sich 2 Operatoren in der gleichen Kategorie befinden (oder gleiche Operatoren vorhanden sind), dann verarbeitet der Compiler diese wie sie erscheinen, von links nach rechts.



Wenn Sie Variablen mit negativem Vorzeichen verwenden, kann dies in manchen Fällen zu unerwarteten Ergebnissen führen, die Sie durch Klammersetzung vermeiden.



### Beispiel

```
var = 1/-x           'nicht empfohlene
                    'Schreibweise
var = 1 / (-x)       'Korrekt: Negativer
                    'Umkehrwert
```

## 4.5 Bedingungen, Schleifen und Module

Wenn Sie Programme schreiben, strukturieren Sie diese in *TiCoBasic* mit folgenden Elementen:

- Kontrollstrukturen verkürzen aufwändige Abschnitte.
  - Schleifen für oft wiederholte Abschnitte:  
   Do ... Until oder  
   For ... Next.
  - Abfragen für fallweise Unterscheidungen:  
   If ... EndIf oder  
   SelectCase ... EndSelect.
- Unterprogramm- und Funktions-Makros ermöglichen Ihnen, häufig benutzte Programmabschnitte zu definieren als
  - Unterprogramm-Makros mit Sub ... EndSub
  - Funktions-Makros mit Function ... EndFunction
- Bibliotheken (Libraries) von kompilierten Funktionen und Unterprogrammen, die Sie mit Import in den Quelltext einbinden:
  - Library-Unterprogramme: Lib\_Sub ... Lib\_EndSub
  - Library-Funktionen: Lib\_Function ... Lib\_EndFunction
- Sammlungen von Quelltext-Abschnitten und Programm-Makros in Include-Dateien, die Sie komplett in den Quelltext einbinden mit  
 #Include filename.Inc

Sie finden nähere Erklärungen und Beispiele der Befehle in Kapitel 7 „Befehlsreferenz“.

### 4.5.1 Unterprogramm- und Funktions-Makros

Unterprogramme und Funktionen definieren Makros, d. h. deren vollständiger Anweisungsblock wird (noch vor dem Kompilieren) an der aufrufenden Stelle in den Quelltext eingefügt.

Die Syntax von Unterprogramm- und Funktions-Makros ist sehr einfach, Sie müssen lediglich die Begriffe `Sub ... EndSub` und `Function ... EndFunction` wie eine Klammer um die jeweiligen Programmabschnitte legen. Funktionen geben – im Unterschied zu Unterprogrammen – einen Wert zurück.

Makros erhöhen die Übersichtlichkeit Ihres Quelltextes. Beachten Sie aber auch, dass jeder Aufruf die erzeugte Binärdatei vergrößert. Sie können alternativ auch Library-Funktionen oder -Unterprogramme verwenden (siehe unten).

Sie finden nähere Informationen zum Aufbau von Makros in der Befehlsreferenz (Seite 143: `Function ... EndFunction`; Seite 194: `Sub ... EndSub`).

### 4.5.2 Include-Dateien

Sie können eine Sammlung von Quelltext-Abschnitten erstellen und in einer sogenannten „Include-Datei“ speichern. Solche Dateien (bzw. den darin enthaltenen Quelltext) können Sie sehr einfach mit dem Befehl `#Include` in Ihren aktuellen Quelltext einbinden.

Der Inhalt von Include-Dateien unterliegt den gleichen Regeln wie der von normalen Quelltext-Dateien, vorwiegend enthalten sie jedoch nur Unterprogramm- und Funktions-Makros.

Zum Erstellen einer Include-Datei geben Sie, wie bei einer „normalen“ *TiCoBasic*-Datei, den gewünschten Quelltext ein und speichern diesen mit „File / Save as“ als Dateityp „Include file \*.Inc“.

Je nach enthaltenem Quelltext müssen Sie darauf achten, an welcher Stelle Sie die Include-Datei in Ihren aktuellen Quelltext einbinden, damit die korrekte Programmstruktur gewahrt bleibt. Wenn die Include-Datei Unterprogramm- und Funktions-Makros enthält, muss sie beispielsweise vor dem Abschnitt `Init:` oder nach dem Abschnitt `Finish:` eingebunden werden.

Sie können Include-Dateien auch in Quelltexte von Library-Dateien oder anderen Include-Dateien einbinden.

Die Include-Dateien, die mit *TiCoBasic* geliefert werden, enthalten nur Unterprogramm- und Funktions-Makros, die Befehle für den Hardware-Zugriff definieren. Aus diesem Grund ist die korrekte Stelle für das Einbinden dieser Dateien der Anfang des Quelltexts (siehe Seite 74).



#### 4.5.3 Bibliotheken (Libraries)

In einer Bibliothek können Sie kompilierte Library-Unterprogramme und -Funktionen (Module) zusammenfassen. Mit dem Befehl `Import` binden Sie diejenigen Module einer Library in einen Prozess ein, die dort tatsächlich aufgerufen werden.

Die Library-Module sind den Funktions- und Unterprogramm-Makros ähnlich. Sie erstellen diese in einem Quelltext mit den Befehlen `Lib_Sub ... Lib_EndSub` und `Lib_Function ... Lib_EndFunction` und kompilieren daraus die Library-Datei mit „Build / Make lib file“.

Wenn Sie einen Quelltext kompilieren, in dem eine Library importiert wird, werden nur die im Quelltext aufgerufenen Library-Module zu der Binärdatei hinzugefügt. Ein mehrfacher Aufruf im Quelltext vergrößert die Binärdatei nicht (im Gegensatz dazu siehe auch Kapitel 4.5.1 „Unterprogramm- und Funktions-Makros“), jedoch benötigt jeder einzelne Aufruf auch zusätzliche Ausführungszeit.



Beachten Sie bitte, dass ein Library-Modul kein Library-Modul innerhalb der gleichen Library-Datei aufrufen kann. Wir empfehlen Ihnen, statt dessen Funktions- und Unterprogramm-Makros zu verwenden. Alternativ können Sie auch eine zusätzliche Library erstellen (oder mehrere).

Wenn Sie Libraries verschachtelt einbinden (d.h. in einer Library eine weitere Library einbinden), müssen Sie im aufrufenden Quelltext die Libraries aller Schachtelungsstufen einbinden (siehe Abb. 12), sonst erhalten Sie eine Fehlermeldung des Compilers.



Rekursive Aufrufe von Library-Funktionen oder Unterprogrammen sind nicht erlaubt.

Sie finden nähere Informationen zum Aufbau von Library-Modulen in der Befehlsreferenz (Seite 157: `Lib_Function ... Lib_EndFunction`; Seite 161: `Lib_Sub ... Lib_EndSub`).

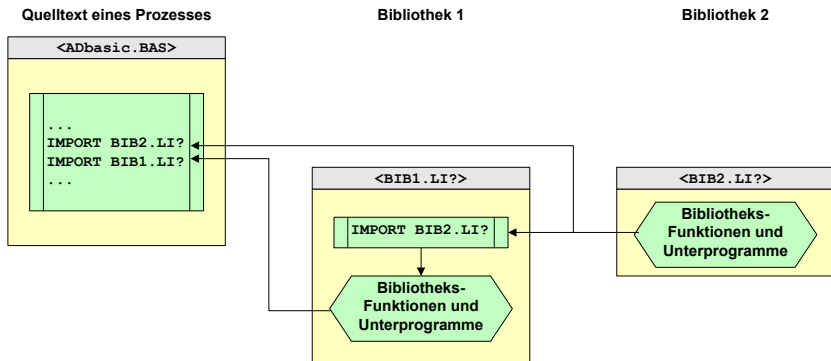


Abb. 12 – Verschachteltes Einbinden von Bibliotheken





## 5 Prozesse optimieren


Der *Tico*-Prozessor ist dafür ausgelegt, Regel-, Steuer- und Messaufgabe schnell und präzise auszuführen. Je nach Anforderung kann es erforderlich werden, dass Sie Ihr *TiCoBasic*-Programm für eine schnellere Bearbeitungszeit optimieren.

Im folgenden zeigen wir beispielhaft, mit welchen Mitteln und an welchen Stellen Sie bei einer Optimierung ansetzen können. Die Vorgehensweise hängt von vielen Faktoren ab und ist daher auf den Einzelfall abzustimmen.

### 5.1 Bearbeitungszeit messen

Als Grundlage für eine Optimierung ist es wichtig, die Bearbeitungszeit eines Prozesszyklus oder von Programmabschnitten zu messen. Sie verwenden hierzu den internen Zähler des *Tico*-Prozessors.

Der *Tico*-Prozessor verfügt über einen internen Zähler, der in Zeittakten von 20ns hochgezählt wird. Mit dem Befehl **Read\_Timer** () können Sie den aktuellen Zählerstand feststellen.

Nach dem Einschalten der Spannungsversorgung wird der Zähler auf den Wert 0 (Null) gesetzt und anschließend in festen Zeittakten kontinuierlich hochgezählt. 

Sie messen die Bearbeitungszeit von Programmen als Zeitdifferenz. Im folgenden Beispiel wird die Bearbeitungszeit eines zeitkritischen Abschnitts (abzüglich eines Offsets) in der globalen Variablen **Par\_1** gespeichert.

Sie erhalten den Offset, wenn Sie die beiden **Read\_Timer** () -Zeilen nacheinander – ohne dazwischen liegende Programmzeilen – ausführen und die Differenz dieser Werte bilden. Der Offset muss für das betrachtete Programm nur ein Mal ermittelt werden.

#### Beispiel

```
Dim t1, t2 As Long
```

#### Event:

```
...
t1 = Read_Timer ()
...
t2 = Read_Timer ()
Par_1 = t2 - t1 - 4
```

*'zeitkritischer Abschnitt*

*'Bearb.zeit des zeitkritischen*  
*'Abschnitts in Zeittakten*  
*' (Offset = 4 Zeittakte)*

Wenn `Par_1` im obigen Beispiel den Wert 37 erhält, hat der zeitkritische Abschnitt  $37 \times 20\text{ns} = 740\text{ns}$  benötigt.

Sie können die Zeitmessung auch benutzen, um beispielsweise die Zeitdifferenz zwischen zwei externen Event-Signalen zu messen. Im Beispiel wird diese bei jedem Aufruf in der globalen Variablen `Par_1` gespeichert.



### Beispiel

```
Dim oldtime, time As Long
```

#### Init:

```
oldtime = Read_Timer()
```

#### Event:

```
time = Read_Timer()
Par_1 = time - oldtime
oldtime = time
```

## 5.2 Verschiedene Tipps

### 5.2.1 Zugriff auf Hardware-Adressen

Viele Funktionen des *Tico*-Prozessors werden über dessen Steuer- und Datenregister kontrolliert. Diese Funktionen können Sie sehr schnell ausführen lassen, wenn Sie mit den Befehlen **In** und **Out** *direkt* auf die entsprechenden Register zugreifen. Direkt bedeutet, dass Sie im Prozesszyklus die Adressen nicht berechnen, sondern als konstante Werte übergeben: Sie sparen die Berechnungszeit ein.

Die Adressen der Steuer- und Datenregister finden Sie im entsprechenden Hardware-Handbuch.

### 5.2.2 Konstanten anstelle von Variablen

Eine Berechnung kann deutlich schneller ausgeführt werden, wenn Sie Werte als Konstanten und nicht mit Variablen angeben.



### Beispiel

```
Par_1 = Par_2*Par_2      'mit Par_2=17
Par_1 = 17*17
```

Für die erste Berechnung muss zur Laufzeit der Wert der Variablen `Par_2` ermittelt, das Quadrat berechnet und `Par_1` zugewiesen werden.

In der zweiten Berechnung kann schon der Compiler den Wert ermitteln. Zur Laufzeit wird der Wert nur noch zugewiesen.

### 5.2.3 Schnellere Messfunktion

Mit dem Befehl `ADC` wird eine A/D-Wandlung für einen Kanal mit bestimmter Verstärkung vorgenommen. Der Befehl ist sehr einfach gehalten, um Ihnen die Anwendung zu erleichtern, denn er fasst mehrere Ablaufschritte zusammen (siehe Hardware-Handbuch zum *ADwin*-System).

Es gibt verschiedene Situationen, in denen Sie mit den einzelnen Ablaufschritten einen schnelleren Ablauf erzielen können als mit dem Befehl `ADC`.

Beispielsweise wird mit dem Befehl `ADC` nicht ausgenutzt, dass sich auf einem *ADwin-Gold II*-System zwei ADC befinden, die gleichzeitig zwei verschiedene Kanäle konvertieren können. Dies geschieht im folgenden Beispiel:

#### Beispiel



```

Rem Beispiel für Gold II
Rem Multiplexer der ADC auf Kanäle 1 und 2 setzen
Set_Mux1(000b)
Set_Mux2(000b)
...
Start_Conv(11b)      'Einschwingzeit abwarten
                     'Wandlung an beiden ADC
                     'starten
Wait_EOC(11b)        'Wandlungsende abwarten
Par_1 = ReadADC(1)    'Auslesen von ADC1
Par_2 = ReadADC(2)    'Auslesen von ADC2
    
```

### 5.2.4 Wartezeit genau einstellen

Mit einer Wartezeit kann man leicht einen genauen Zeitabstand zwischen 2 Befehlen einstellen, z. B. um eine feste Bearbeitungszeit eines Hardware-Bausteins zu überbrücken.

Der Befehl `Sleep` stellt die genaue Wartezeit ein: Der Prozessor stoppt für die eingestellte Zeit, so dass der folgende Befehl entsprechend später startet.

### 5.2.5 Wartezeiten nutzen

Manche Befehle erfordern nach ihrem Aufruf eine bestimmte Wartezeit, ohne dabei den Prozessor zu nutzen. Diese Zeit können Sie für andere Berechnungen nutzen.

Solche Befehle sind `Set_Mux1/2` und `Start_Conv`, nach denen Wartezeiten nötig sind, um das Einschwingen des Multiplexers und die Konvertierung

der ADC abzuwarten. Während dieser Wartezeit ist aber der Prozessor nicht beschäftigt, könnte also andere Aufgaben übernehmen.

Genauere Angaben über die erforderlichen Wartezeiten bei der Datenwandlung finden Sie in Ihrem Hardware-Handbuch.

Als praktische Anwendung wird das Beispiel aus dem Abschnitt „Schnellere Messfunktion“ nun so erweitert, dass in einem Prozesszyklus 2 Messungen an je 2 ADC durchgeführt werden. Dadurch können Sie im Vergleich zum Befehl ADC in der gleichen Zeit die 4fache Zahl an Messungen durchführen.

Der wesentliche Effekt beruht darauf, dass die einzelnen Schritte der 2 Messungen nicht nacheinander folgen, sondern das Setzen des Multiplexers in die Wartezeit der jeweils anderen Messung verschoben wird. Die Abläufe der beiden Messungen überlagern sich: Auf den Wandlungsstart für die Kanäle 1+2 folgt das Setzen des Multiplexers für die Kanäle 3+4.



### Beispiel

*Rem Beispiel für Gold Rev. B*

#### Init:

```
Set_Mux(000000b)    'Mux für 1. Messung
                    'setzen, Kanäle 1+2
Sleep(140)           '14 µs warten
```

#### Event:

```
Start_Conv(11b)      'Wandlung starten (Kanäle
                    '1+2)
Set_Mux(001001b)     'Mux setzen, Kanäle 3+4
Wait_EOC(11b)        'Wandlungsende abwarten
                    '(Kanäle 1+2)
Par_1 = ReadADC(1)    'Auslesen von ADC1, Kanal
                    '1
Par_2 = ReadADC(2)    'Auslesen von ADC2, Kanal
                    '2

Start_Conv(11b)      'Wandlung starten (Kanäle
                    '3+4)
Set_Mux(000000b)     'Mux setzen, Kanäle 1+2
Wait_EOC(11b)        'Wandlungsende abwarten
                    '(Kanäle 3+4)
Par_3 = ReadADC(1)    'Auslesen von ADC1, Kanal
                    '3
Par_4 = ReadADC(2)    'Auslesen von ADC2, Kanal
                    '4
```

Für die erste Messung im Abschnitt **Event**: ist nun erforderlich, den Multiplexer bereits im Abschnitt **Init**: zu setzen, damit der erste Start der Wandlung definiert ist.

Achten Sie streng darauf, dass Sie die Mindest-Wartezeiten für das Einschwingen des Multiplexers und für die Konvertierung der ADC nicht unterschreiten, da sonst die A/D-Wandlung nicht funktioniert und falsche Ergebnisse liefert. Hinweise bietet das Kapitel 5.2.4 „Wartezeit genau einstellen“.



### 5.2.6 Optimierung des Speicherzugriffs

Der Zugriff auf den externen Speicher ist relativ langsam, insbesondere beim Zugriff auf Einzeldaten. Bei einem Prozess mit niedriger Priorität kann ein Einzelzugriff auf den externen Speicher sogar dazu führen, dass die Reaktionszeit eines Prozesses mit hoher Priorität verlangsamt wird.

Zusätzlich ergibt sich bei jedem Zugriff auf den externen Speicher des *TiCo*-Prozessors eine Wartezeit, die variieren kann („Jitter“). Der Grund ist, dass der *TiCo*-Prozessor davon ausgeht, dass man auf zufällige Speicherstellen zugreift, und deswegen jeden Zugriff neu organisiert – mit entsprechender Wartezeit.

Vermeiden Sie die oben genannten Nachteile, indem Sie die Datenstruktur **RingBuffer** für den Zugriff auf Daten im externen Speicher verwenden. Beachten Sie: Die Nachteile werden erst nach der Initialisierung und dem ersten Zugriff mit der Datenstruktur **RingBuffer** vermieden.

Informationen zur Anwendung der Datenstruktur **RingBuffer** finden Sie in Kapitel 4.3.3 auf Seite 85.



## 6 Prozesse im Betriebssystem



Das *ADwin*-System stellt alle Möglichkeiten zur Verfügung, um komplexe Anlagen zu regeln, zu steuern und Messungen durchzuführen. Mit *TiCoBasic* programmieren Sie einen Prozess, der diese Möglichkeiten nutzt: Sie legen darin fest, wie und wann der *Tico*-Prozessor analoge und digitale Daten verarbeitet und nach außen gibt, entweder um der *ADwin* CPU zuzuarbeiten oder auch ganz eigenständig.

Nach dem Start des Prozesses wird das Programm<sup>1</sup> im *Tico*-Prozessor (typischerweise) zyklisch, also in regelmäßigen Abständen neu aufgerufen und abgearbeitet. Ein solcher Aufruf eines Prozesszyklus wird durch eines der folgenden Startsignale, sogenannte „Events“, ausgelöst:

1. Timer-Event: Ein Impuls des internen Zählers. Sie können für jeden Prozess separat festlegen, in welchem Zeitabstand (Processdelay) ein neuer Event ausgelöst wird: das ist ein zeitgesteuerter Prozess.
2. Externer Event: Ein externes Signal, das am „Event“-Eingang Ihres *ADwin*-Systems eingeht. Dies könnte beispielsweise ein Impuls eines Inkrementalgebers sein: das ist ein extern gesteuerter Prozess.
3. Der Prozessstyp None (ohne Event-Signal) wird nur für – meist in Assembler programmierte – Spezialanwendungen benötigt und schließt andere Prozessstypen aus. Wenn nicht anders programmiert, reagiert der Prozess nicht auf Event-Signale und wird nur einmal durchlaufen.

Die exakte Funktion eines Prozesses definieren Sie im *TiCoBasic*-Quelltext:

- Die Initialisierung im Abschnitt **Init:**.
- Die eigentliche Funktion des Prozesszyklus im zentralen Abschnitt **Event:**.
- Die Schlussbearbeitung im Abschnitt **Finish:**.

Von der *ADwin* CPU aus können Sie die Prozesse eines Systems steuern, d.h. Sie können die Prozesse starten, stoppen oder deren Processdelay ändern. Vom PC aus ist dies nur mit der Entwicklungsumgebung *TiCoBasic* möglich. Mit der Bootloader-Option können Prozesse außerdem beim Starten der *ADwin*-Hardware automatisch gestartet werden. Näheres zum Programmieren des Bootloaders siehe Kapitel 3.6.2 „TiCo-Bootloader programmieren“, Seite 40.

---

1. genauer: der Programmabschnitt **Event:**.





### 6.1 Prozessverwaltung

Auf dem *TiCo*-Prozessor sollte nur ein einziger Prozess (mit hoher Priorität) laufen. Je nach Aufgabenstellung wählen Sie dafür einen der folgenden Prozessstypen:

- Zeitgesteuerter Prozess (Timer)

Neben dem hochprioren zeitgesteuerten Prozess ist zusätzlich auch ein niederpriorer zeitgesteuerter Prozess möglich. Der Prozess mit niedriger Priorität kann nicht alleine ausgeführt werden.

- Extern gesteuerter Prozess (External)

Der extern gesteuerte Prozess hat immer hohe Priorität.

- Prozess ohne Ansteuerung (None)

Bei dem Prozess ohne Ansteuerung spielt die Priorität keine Rolle.

Es ist auch möglich, einen zeitgesteuerten und einen extern gesteuerten Prozess miteinander zu kombinieren. Wenden Sie sich in diesem Fall bitte an unseren Support ([support@adwin.de](mailto:support@adwin.de)), damit wir Sie über die erforderlichen Vorkehrungen informieren können.

Wenn Sie mehrere Prozesse gleichzeitig nutzen wollen, müssen Sie die Quelldateien in ein Projekt einbinden (siehe Kapitel 3.9.2 auf Seite 59).

#### 6.1.1 Zeitgesteuerter Prozess (Timer)

Beim zeitgesteuerten Prozess startet ein regelmäßiger Impuls des internen Zählers einen Prozesszyklus. Der Zeitabstand zwischen zwei Impulsen wird auch als Zykluszeit (Processdelay) bezeichnet und kann in Schritten zu 20ns eingestellt werden (siehe auch Kapitel 4.2.7 auf Seite 81).

Neben dem zeitgesteuerten Prozess mit hoher Priorität ist zusätzlich auch ein zeitgesteuerter Prozess mit niedriger Priorität möglich. Weisen Sie einem Prozess seine Priorität über das Menü „Options \ Process Options“ zu.

Der Prozess mit hoher Priorität wird bevorzugt behandelt:

- Vom Aufruf des Prozesszyklus durch den internen Zähler bis zur Bearbeitung des ersten Befehls vergehen maximal 300ns.
- Der hochpriore Prozesszyklus ist nicht unterbrechbar und wird immer vollständig abgearbeitet.

Auch ein Stopp-Befehl kann einen laufenden, hochprioren Prozesszyklus nicht unterbrechen: es muss auf das Ende der Bearbeitung gewartet werden.

Ein Prozesszyklus mit niedriger Priorität wird sofort unterbrochen, wenn ein Prozesszyklus mit hoher Priorität aufgerufen wird und zwar so lange, bis dieser fertig bearbeitet ist.



Programmieren Sie zeitkritische Vorgänge in den Prozess mit hoher Priorität und andere Vorgänge mit niedriger Priorität, damit der Prozessor die zeitkritischen Prozesszyklen ungestört bearbeiten kann.

### 6.1.2 Extern gesteuerter Prozess (External)

Beim extern gesteuerten Prozess startet ein bestimmtes externes Signal einen Prozesszyklus.

Der Prozess läuft immer mit hoher Priorität:

- Vom Aufruf des Prozesszyklus durch ein externes Signal bis zur Bearbeitung des ersten Befehls vergehen maximal 300ns.
- Der hochpriore Prozesszyklus ist nicht unterbrechbar und wird immer vollständig abgearbeitet.

Auch ein Stopp-Befehl kann einen laufenden, hochprioren Prozesszyklus nicht unterbrechen: es muss auf das Ende der Bearbeitung gewartet werden.

Das aufrufende externe Signal wird sehr flexibel über eine Hardware-Adresse festgelegt: Der Wert der Hardware-Adresse wird mit einer Bitmaske Und-verknüpft und anschließend mit einem festen Wert über einen Operator (<, >, =) verglichen. Wenn der Vergleich wahr ist, wird ein Event-Signal ausgelöst. Zum Einstellen der Werte siehe Dialogfenster „Process Options“ auf Seite 49.

Die Hardware-Adressen sind für jede ADwin-Hardware unterschiedlich.

Beispiel: Die Einstellung oben maskiert den Wert der Adresse 70h mit 12h, so dass nur die Bits 2 und 5 erhalten bleiben. Wenn das Ergebnis > 0 ist (operation und value), wenn also eines der beiden Bits gesetzt ist, wird ein Event-Signal aus gelöst und damit ein Prozesszyklus gestartet. Gesetzt den

Fall, die Hardware-Adresse ist ein Register für Digitaleingänge, löst also jedes Setzen eines der beiden – den Bits 2 und 5 zugeordneten – Digitalkanäle einen Prozesszyklus aus.

### 6.1.3 Prozess ohne Ansteuerung (None)

Der Prozesstyp None (ohne Event-Signal) wird nur für – meist in Assembler programmierte – Spezialanwendungen benötigt und schließt alle anderen Prozesstypen aus. Wir empfehlen die Anwendung nur für sehr erfahrene Nutzer.

Der Prozess reagiert nicht auf Event-Signale, sondern startet, sobald der Prozess auf den *TiCo*-Prozessor übertragen ist oder durch die Bootloader-Funktion. Das Programm wird nur einmal durchlaufen, es beginnt *nicht automatisch* von vorne.

Um das Programm mehr als einmal zu durchlaufen, können beispielsweise Schleifen eingesetzt werden.

Der Prozesstyp `None` beeinflusst das Betriebssystem: Prozesse können von außerhalb weder gestartet noch gestoppt werden, noch kann die Zykluszeit von Prozessen verändert werden.

Beachten Sie die Besonderheiten bei der Programmierung:



- Im Programm gibt es keine Abschnitte, die Kennwörter `Init:`, `Event:` und `Finish:` sind daher ungültig und erzeugen eine Fehlermeldung.
- Der Befehl `End` hat keine Funktion.
- Programmieren Sie an das Ende des Programms eine Endlosschleife. Anderenfalls können unvorhergesehene Probleme auftreten. Eine Endlosschleife kann so aussehen:

```
Do
Until (1 = 2)
```

Die Programmierung mit Assembler ist in einem separaten Handbuch beschrieben.

## 6.2 Zeitverhalten von Prozessen

### 6.2.1 Processdelay

Der Zeitabstand, in dem *zeitgesteuerte* Prozesszyklen vom Zähler aufgerufen werden, ist die Zykluszeit. Sie wird in Taktzyklen des Zählers gemessen und dann als *Processdelay* bezeichnet. Sie können das *Processdelay* jedes Pro-

zesses über den Wert der Systemvariablen **Processdelay** festlegen (siehe auch Seite 173).

Ein Zähler-Taktzyklus im *TiCo*-Prozessor dauert 20ns.

Ein **Processdelay** mit dem Wert 1250 beispielsweise bedeutet einen regelmäßigen Aufruf im Zeitabstand von  $1250 \times 20\text{ns} = 25\mu\text{s}$ . Dies kann z.B. eingestellt werden mit der Programmzeile:

```
Processdelay = 1250
```

Damit jeder (zeitgesteuerte) Prozesszyklus zu der (mit **Processdelay**) festgelegten Zeit aufgerufen wird, darf die Bearbeitungszeit eines Prozesszyklus die Zykluszeit auch im ungünstigsten Fall nicht überschreiten. Unterschiede bei der Berechnungszeit ergeben sich beispielsweise bei fallweisen Unterscheidungen (If, Case).

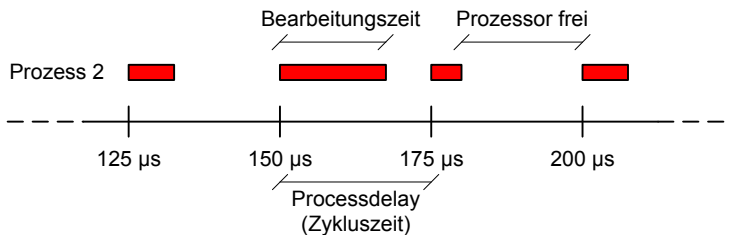


Abb. 13 – Processdelay und Bearbeitungszeit



### Beispiel

Wenn eine umfangreiche Berechnung nur alle 1000 Messungen auftritt, dann muss auch die lange Bearbeitungszeit dieses Prozesszyklus kürzer sein als die Zykluszeit. Um dennoch kurze Prozesszyklen zu erreichen, ist es eine gute Alternative, die Berechnung in kleine Schritte aufzuteilen und in jedem Prozesszyklus jeweils einen Schritt zu bearbeiten. Die Prozesszyklen erhalten dadurch eine durchweg gleichmäßige und kurze Bearbeitungszeit.

## 6.2.2 Auslastung des *TiCo*-Prozessors

Die Auslastung des *TiCo*-Prozessors ist das Verhältnis von genutzter Rechenzeit zur insgesamt verfügbaren Rechenzeit, angegeben in Prozent.

Sie können die Auslastung des Prozessors an der Anzeige „Busy“ in der Statusleiste der Entwicklungsumgebung beobachten (siehe Kapitel 3.9.6). Die-

ser Wert gibt Ihnen einen Anhaltspunkt, ob der Prozessor noch genügend Rechenzeit frei hat, um alle Prozesszyklen abarbeiten zu können.

Die Auslastung des Prozessors sollte 90% nur in Ausnahmefällen überschreiten, den Wert 100% jedoch niemals.

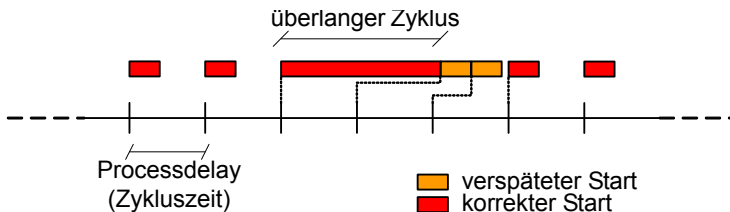
### 6.2.3 Verschiedene Betriebszustände im Betriebssystem

Das Betriebssystem behandelt den zeitgesteuerten und den extern gesteuerten Prozess beim Zeitverhalten unterschiedlich: Bei dem extern gesteuerten Prozess können Event-Signale verloren gehen, bei zeitgesteuerten Prozessen in der Regel nicht.

#### Zeitgesteuerter Prozess

Bei einem zeitgesteuerten Prozess wird in der Regel jeder Prozesszyklus zu der (mit `Processdelay`, Seite 109) festgelegten Zeit aufgerufen. Manchmal ist dies nicht möglich, z.B. weil ein Prozesszyklus länger dauerte als die Zykluszeit; dann laufen Event-Signale des internen Zählers auf.

Das Betriebssystem holt aufgelaufene Event-Signale nach, d.h. Prozesszyklen werden nacheinander ohne Pause aufgerufen, bis das ursprüngliche Zeitraster wieder erreicht ist. Beim niederpriorigen Prozess gilt dies auch, soweit der hochpriorige Prozess nicht aktiv ist.



Wenn aufgelaufene Event-Signale länger als 42,9 Sekunden auf die Abarbeitung warten müssen, werden die Event-Signale nicht mehr nachgeholt. Wenn eine derart große Verzögerung auftritt, müssen Sie das Zeitverhalten des Prozesses überprüfen: Es ist wahrscheinlich, dass der Prozesszyklus regelmäßig länger dauert als die Zykluszeit. In diesem Fall vergrößern Sie die Zykluszeit oder verkürzen die Bearbeitungszeit für den Prozesszyklus durch geeignete Programmierung.



#### Extern gesteuerter Prozess

Bei dem externen Prozess werden eintreffende Event-Signale sehr schnell bearbeitet, jedoch können auch Event-Signale verloren gehen.

Das Betriebssystem verwendet ein Hardware-Register, um externe Event-Signale zu verarbeiten. Ist ein Event-Signal eingetroffen, startet das Betriebssystem sofort einen Prozesszyklus, wenn nicht gerade ein hochpriorer Prozesszyklus bearbeitet wird. In diesem Fall dient das Register als Zwischenspeicher für das Event-Signal, und das Betriebssystem startet den nächsten Prozesszyklus sofort nach dem aktuell bearbeiteten Prozesszyklus.



Wenn mehrere Event-Signale während eines Prozesszyklus eintreffen, werden anschließend nicht entsprechend viele Prozesszyklen aufgerufen, sondern nur ein einziger; es gehen also Event-Signale verloren.

Ein externes Event-Signal ist eine besonders wichtige Information – schon weil sie vom ADwin-System nicht vorherbestimmbar ist – und darf auf keinen Fall verloren gehen. Achten Sie deshalb in diesem Prozess ganz besonders auf kurze Prozesszyklen (im Abschnitt **Event:**).

## 6.3 Kommunikation

### 6.3.1 Datenaustausch zwischen Prozessen

Sie können Daten zwischen *TiCoBasic*-Prozessen über globale Variablen (**Par\_1** ... **Par\_80**) oder über globale Felder (**Data\_n**) austauschen.



Wenn Sie globale Felder in mehreren Prozessen verwenden, müssen Sie diese in jedem Prozess in absolut gleicher Weise deklarieren. In diesem Fall ist es praktisch, wenn Sie die Deklaration der globalen Felder in einer Include-Datei speichern und diese in allen Prozessen einbinden (siehe auch Kapitel 4.5.2 „Include-Dateien“).

Dies gilt nicht für Die Datenstruktur RingBuffer (siehe Kapitel 4.3.3 auf Seite 85); hier darf die Deklaration innerhalb eines Projekts nur ein einziges Mal vorkommen.

Je nach Programmierung können Sie (jeweils gleiche) globale Variablen verwenden, um aus einem Prozess heraus einen anderen, gleichzeitig laufenden Prozess zu steuern.



### Beispiel

Prozess 1 arbeitet als Funktionsgenerator und Prozess 2 als Regler. Der Funktionsgenerator schreibt regelmäßig den jeweils erzeugten Wert in die globale Variable **Par\_10**. Der Regler liest bei jedem Prozesszyklus diese globale Variable **Par\_10** aus und verwendet deren Inhalt als Sollwert des Regelkreises.

Damit steuert der Funktionsgenerator auf einfache Weise den Sollwertverlauf des Reglers. Alle *lokalen* Variablen und Felder des Prozesses 1 bleiben hierbei dem Prozess 2 verborgen (und umgekehrt). Beachten Sie bitte, dass bei der Zusammenarbeit der beiden Prozesse auch deren Zeitverhalten von Bedeutung ist.

### 6.3.2 Kommunikation zwischen PC und TiCo-Prozessor

Vom PC aus haben Sie keinen direkten Zugriff auf den *TiCo*-Prozessor, der Zugriff ist nur von der *ADwin* CPU aus möglich. Die *ADwin* CPU kann Prozesse im *TiCo*-Prozessor steuern, Daten von dort lesen oder dorthin schreiben (siehe Kapitel 6.3.3 auf Seite 114). Der *TiCo*-Prozessor selbst kommuniziert nicht aktiv.

Um Daten zwischen PC und *TiCo*-Prozessor austauschen zu können, muss die *ADwin*-CPU als Zwischenstation eingerichtet werden. Hierbei überträgt ein Prozess – den Sie selbst erstellen müssen – auf der *ADwin*-CPU die Daten und Steuersignale. Auf diese Weise arbeitet auch der Datenfluss bei der Entwicklungsumgebung *TiCoBasic*.

Daten werden immer über globale Variablen (**Par\_n**, **FPar\_n**) oder globale Felder (**Data\_n**) ausgetauscht (auch beim Datenaustausch zwischen Prozessen, siehe oben).

### Kommunikation zwischen PC und ADwin CPU

Die Kommunikation zur *ADwin* CPU läuft unter Windows über die sogenannte „ADwin32.dll“ (dynamic-link library), in der *ADwin* CPU übernimmt diese Aufgabe ein Kommunikationsprozess.

Wenn Sie mit der ActiveX-Schnittstelle arbeiten, übernimmt diese – auf den ersten Blick ähnlich zentral – jede Kommunikation mit der *ADwin* CPU. Intern gibt die ActiveX-Schnittstelle die kommunizierten Daten weiter an die „ADwin32.dll“ oder erhält sie von dieser.

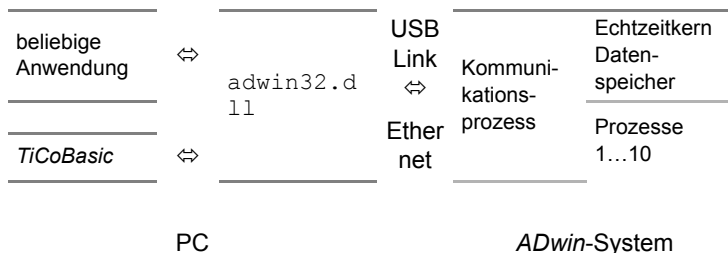


Abb. 14 – Kommunikation zwischen PC und *ADwin*-System

Die „ADwin32.dll“ übernimmt folgende Aufgaben:

- Kommunikation mit dem angesprochenen *ADwin*-System über Ethernet (TCP/IP).
- Erkennen und Behandeln von Fehlern.
- Verriegeln mehrerer PC-Anwendungen gegeneinander, wenn diese gleichzeitig auf dasselbe System zugreifen möchten.

Durch die Verriegelung können mehrere Anwendungen unabhängig voneinander und quasi gleichzeitig auf ein oder mehrere *ADwin*-Systeme zugreifen.

Wenn eine PC-Anwendung die Kommunikation zu einem bestimmten System aufnimmt, übergibt es neben der gewünschten Anweisung auch eine Geräte-Nummer, die sogenannte „Device No“. Anhand dieser „Device No“ unterscheidet die „ADwin32.dll“ die verschiedenen *ADwin*-Systeme und ordnet die entsprechenden Einstellungen zu.

### 6.3.3 Kommunikation zwischen ADwin CPU und TiCo-Prozessor

Die *ADwin* CPU kann Prozesse im *TiCo*-Prozessor steuern, Daten von dort lesen oder dorthin schreiben. Der *TiCo*-Prozessor selbst kommuniziert nicht aktiv.

Mit *ADbasic*-Befehlen kann die *ADwin* CPU auf den *TiCo*-Prozessor zugreifen und folgende Aktionen ausführen:

- Datenzugriff Initialisieren.
- Globale Variablen Par\_1...Par\_80 lesen und schreiben.
- Globale Felder Data\_1...Data\_16 lesen und schreiben.
- Ringspeicher lesen und schreiben sowie Zustand abfragen.
- Processdelay eines *TiCo*-Prozesses einstellen und abfragen.
- *TiCo*-Prozesse starten und stoppen.
- Prozessor starten, stoppen und zurücksetzen.
- Systeminformation abfragen.
- Binärdatei übertragen.

Eine detaillierte Beschreibung der Befehle finden Sie hier:

- *ADwin-Gold II*: Kapitel 7.3 auf Seite 197.
- *ADwin-Pro II*: Kapitel 7.4 auf Seite 245.



### 6.3.4 Die Device No

Jedes *ADwin*-System, das an einen PC angeschlossen ist, wird über eine (in diesem PC) eindeutige Gerätenummer, die *Device No* angesprochen.

Sie stellen die Device-No. mit dem Programm *ADconfig* ein: Programs ► *ADwin* ► *ADconfig*.

In *ADconfig* verknüpfen Sie eine „Device No“ mit den Verbindungsparametern, mit denen ein System erreichbar ist (über TCP/IP). Auf diese Informationen greift die „*ADwin32.dll*“ zurück, um mit dem System kommunizieren zu können.



## 7 Befehlsreferenz

Im folgenden sind die in *TiCoBasic* verfügbaren Befehle für *TiCo*-Prozessoren aufgeführt. Befehle zur Steuerung der Ein- und Ausgänge finden Sie in der Hardware-Dokumentation.

Die Befehle sind alphabetisch sortiert aufgeführt. Im Anhang gibt es eine Befehlsübersicht.

In Kapitel 7.3 und Kapitel 7.4 sind die *TiCoBasic*-Befehle aufgeführt, mit denen die *ADwin* CPU auf den *TiCo*-Prozessor zugreifen kann; die Befehle sind für *ADwin-Gold II* und für *ADwin-Pro II* separat aufgeführt.

### 7.1 Befehlssyntax

Beachten Sie bitte:

- Als Argument ist ein beliebiger Berechnungsausdruck möglich.
- Bei einigen Argumenten ist die Datenstruktur vorgeschrieben, die wie folgt gekennzeichnet ist:

**CONST** Konstante Zahlen wie **35** sowie Berechnungsausdrücke ohne Variablen.

**VAR** Variable oder Feldelement.

**ARRAY** Feld, in der Syntaxzeile auch erkennbar an den Klammern [ ] nach dem Feldnamen.

- Neben einem Argument oder dem Rückgabewert einer Funktion ist der erwartete Datentyp angegeben:

**LONG** ganze Zahl

**LOGIC** logischer Ausdruck in einer Bedingung

- Manche Befehle können nur benutzt werden, wenn eine bestimmte Library- oder Include-Datei eingebunden wird. Unter **Syntax** ist der jeweilige Befehl zum Einbinden angegeben (setzen Sie diese Befehlszeile bitte an den Anfang des Quelltextes).

Es wird davon ausgegangen, dass die erforderliche Library- oder Include-Datei in dem Verzeichnis liegt, das unter „Options ▶ Settings“ unter dem Reiter „Directory“ eingestellt ist (siehe auch die Befehle **#Include** oder **Import**).

## **7.2 Basis-Befehlssatz *TiCoBasic***

Die Befehle in diesem Abschnitt sind für alle *TiCo*-Prozessoren gültig.

### + (Addition)

Der Operator „+“ addiert je zwei Werte.

#### Syntax

```
val = val_1 + val_2
```

#### Parameter

val\_1                      Summand 1

LONG

val\_2                      Summand 2

LONG

#### Bemerkungen

- / -

#### Siehe auch

- (Subtraktion), \* (Multiplikation), / (Division), ^ (Potenz)

#### Beispiel

```
Par_1 = 9 + 4                      'Par_1 = 13
```

## - (Subtraktion)

Der Operator „-“ subtrahiert je zwei Werte.

### Syntax

```
val = val_1 - val_2
```

### Parameter

<code>val_1</code>	Minuend
<code>val_2</code>	Subtrahend

LONG

LONG

### Bemerkungen

- / -

### Siehe auch

+ (Addition), \* (Multiplikation), / (Division), ^ (Potenz)

### Beispiel

```
Par_1 = 9 - 4          'Par_1 = 5
```

## \* (Multiplikation)

Der Operator „\*“ multipliziert je zwei Werte.

### Syntax

```
val = val_1 * val_2
```

### Parameter

val\_1                      Multiplikator 1

LONG

val\_2                      Multiplikator 2

LONG

### Bemerkungen

- / -

### Siehe auch

+ (Addition), - (Subtraktion), / (Division), ^ (Potenz)

### Beispiel

```
Par_1 = 9 * 4                      'Par_1 = 36
```

## / (Division)

Der Operator „/“ dividiert je zwei Werte.

### Syntax

```
val = val_1 / val_2
```

### Parameter

`val_1` Dividend

LONG

`val_2` Divisor

LONG

### Bemerkungen

Beachten Sie, dass eine Division immer ohne Rest durchgeführt wird.

Wenn Sie durch eine Variable mit negativem Vorzeichen teilen, müssen Sie diese in Klammern setzen, damit das erwartete Ergebnis berechnet wird (siehe auch Kapitel 4.4.1 „Auswertung von Operatoren“).

### Siehe auch

+ (Addition), - (Subtraktion), \* (Multiplikation), ^ (Potenz)

### Beispiel

```
Par_1 = 36 / 4          'Par_1 = 9
Par_2 = 2 / 4 * 5       'Par_2 = 0 -> ganzzahlige
                        'Rechnung
Par_3 = 27 / (-Par_1)   'Par_3 = -3
Rem Beachten Sie die Klammersetzung in der letzten Zeile
```



## ^ (Potenz)

Der Operator „^“ berechnet eine beliebige Potenz eines Wertes.

### Syntax

```
val = val_1 ^ val_2
```

### Parameter

val\_1                      Basis

LONG

val\_2                      Exponent

LONG

### Bemerkungen

Wenn die Basis und/oder der Exponent eine Variable mit negativem Vorzeichen ist, müssen Sie diese in Klammern setzen, damit das Vorzeichen bei der Potenzierung berücksichtigt wird (siehe auch Kapitel 4.4.1 „Auswertung von Operatoren“). Bei Konstanten ist dies nicht der Fall.

```
var1 = -2^2           'var1 = 4
var2 = -var1^2        'var2 = -16
var3 = (-var1)^2      'var3 = 16
```

Polynome werden schneller berechnet, wenn Sie die Potenzen mittels Ausklammerung durch wenige Multiplikationen ersetzen:

```
y = a + b*x + c*x^2 + d*x^3 + e*x^4 'langsame Variante
y = a + x*(b + x*(c + x*(d + x*e))) 'schnelle Variante
```

### Siehe auch

+ (Addition), - (Subtraktion), \* (Multiplikation), / (Division)

### Beispiel

```
Par_1 = 9 ^ 4           'Par_1 = 6561
```

## #... (Präprozessor-Anweisung)

Ein *TiCoBasic*-Befehl, der mit dem Zeichen „#“ beginnt, ist eine Anweisung für den sogenannten „Präprozessor“, den Quelltext auf eine bestimmte Weise zu bearbeiten. Das Ergebnis der Bearbeitung wird vom Compiler verarbeitet.

Folgende Präprozessor-Anweisungen stehen Ihnen zur Verfügung:

<b>#Define</b>	Definition symbolischer Konstanten: Zeichenfolgen im Quelltext werden durch andere Zeichenfolgen ersetzt.
<b>#Include</b>	Datei einfügen: Eine Datei (mit Quelltext) wird in den Quelltext eingefügt.
<b>#If...#EndIf</b>	Bedingte Kompilierung: Bei erfüllter Bedingung werden die entsprechenden Quelltextzeilen kompiliert, anderenfalls gelöscht

## : (Doppelpunkt)

Das Zeichen „:“ trennt Programmschritte innerhalb einer einzelnen Programmzeile.

### Syntax

```
[Schritt_1] : [Schritt_2] {: [Schritt_3] ...}
```

### Bemerkungen

[Schritt\_n] bezeichnet einen beliebigen Programmschritt, wie er sonst in einer einzelnen Programmzeile angegeben wird.

Eine Programmzeile darf nicht mehr als 255 Zeichen beinhalten (Ausnahme siehe **#Include** auf Seite 154).

Verwenden Sie den Befehl nur, wenn dadurch der Quelltext übersichtlicher wird.

### Beispiel

```
Inc Par_1 : Inc Par_2  
Rem Par_1 und Par_2 erhöhen in *einer* Zeile
```

## = (Zuweisung)

Der Operator „=“ weist der Variablen oder dem Feldelement links vom Operator das Ergebnis des Ausdrucks rechts vom Operator zu.

### Syntax

```
var = expr
```

### Parameter

`var` Variable oder Feld

VAR

LONG

`expr` Berechnungsausdruck

LONG

### Bemerkungen

- / -

### Beispiel

```
Dim val_1, val_2 As Long 'Deklaration
```

#### Init:

```
val_1 = 69 'Zuweisung einer  
           'Konstanten
```

#### Event:

```
val_2 = val_1 * 2 'Zuweisung eines  
                  'Ausdrucks
```

## < = > (Vergleich)

Die Operatoren „<“, „=“ und „>“ dienen zum Vergleich zweier Werte. In *TiCo-Basic* kommen diese Operatoren nur in Bedingungen vor.

### Syntax

```
If (val_1 > val_2) Then
```

### Parameter

val\_1            Operand

LONG

val\_2            Operand

LONG

### Bemerkungen

Folgende Vergleiche sind möglich:

Operator	Bedeutung
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich
=	gleich
<>	ungleich

### Siehe auch

If ... Then ... {Else ... } EndIf, #If ... Then ... {#Else ... } #EndIf

### Beispiel

```
Dim value As Long
```

```
Event:
```

```
value = -5
```

```
If (value < 0) Then value = 0
```

```
Rem Ergebnis: value = 0
```

## AbsI

**AbsI** liefert den Betrag einer Long-Variablen.

### Syntax

```
ret_val = AbsI(value)
```

### Parameter

**value** Argument:  $-(2^{31}-1) \dots +2^{31}-1$ .

LONG
------

**ret\_val** Betrag des Arguments ( $0 \dots +2^{31}-1$ ).

LONG
------

### Bemerkungen

Für den kleinsten negativen, ganzzahligen Wert  $-2^{31}$  gibt es in *TiCo-Basic* keine positive Entsprechung; der Betrag dieses Werts ist daher undefiniert.

### Siehe auch

- / -

### Beispiel

```
Dim val_1, val_2 As Long
```

#### Event:

```
val_1 = -5
```

```
val_2 = AbsI(val_1) 'Ergebnis: val_2 = 5
```

## And

Der Operator **And** verknüpft zwei ganzzahlige Werte bitweise oder zwei Boolesche Ausdrücke als Boolescher Operator.

### Syntax

```
ret_val = val_1 And val_2      'Bitweiser
                               Operator

If ((expr1) And (expr2)) Then  'Boolescher
                               Operator
```

### Parameter

<code>val_1, val_2</code>	Ganzzahliger Wert	<div style="border: 1px solid black; padding: 2px; display: inline-block;">LONG</div>
<code>expr1, expr2</code>	Boolescher Ausdruck mit dem Wert „wahr“ oder „falsch“	<div style="border: 1px solid black; padding: 2px; display: inline-block;">LOGIC</div>

### Bemerkungen

Sie können mit **And** nur gleichartige Ausdrücke verknüpfen (ganzzahlige *oder* Boolesche), ein Mischen ist nicht möglich.

Sie können Boolesche Ausdrücke nur mit den Anweisungen **If ... Then ... Else** oder **Do ... Until** verwenden (Variablen können keine Booleschen Werte annehmen).

Wenn Sie in einer Zeile mehrere Boolesche Operatoren verwenden, müssen Sie jede Verknüpfung separat in Klammern setzen. Bei der Verknüpfung ganzzahliger Werte ist dies nicht erforderlich.

### Siehe auch

Not, Or, XOr

### Beispiel

```
Rem Bitweise Verknüpfung von Long-Variablen
Dim val_1, val_2, val3 As Long
val_1 = 0100b      '= 4
val_2 = 0110b      '= 6
val3 = val_1 And val_2 'Bitweise Verknüpfung
Rem Ergebnis: val3 = 0100b = 4
```

Oder:

*Rem Boolesche Verknüpfung von Booleschen Ausdrücken*

```
Dim val_1, val4 As Long
```

```
val_1 = 314
```

*Rem Boolesche Verknüpfung: (wahr) And (wahr) = wahr*

```
If ((val_1 < 910) And (val_1 > 310)) Then
```

```
    val4 = 1
```

```
Else
```

```
    val4 = 0
```

```
EndIf
```

*'Ergebnis: val4 = 1*



## Data\_n

Mit `Dim Data_n[...]` `As ...` wird ein globales **DATA**-Feld dimensioniert.  
Weitere Informationen zur Dimensionierung siehe [Dim](#) auf Seite 135.

### Syntax

```
Dim Data_n[dim1] As Long

{At <Mem_Type>}
```

### Parameter

<b>Data_n</b>	Name des deklarierten <b>DATA</b> -Felds mit <b>n</b> : 1...16.	
<b>dim1, dim2</b>	Feldgröße: Anzahl (≥1) der Elemente vom Typ <code>ARR_</code> <b>TYPE</b> im Feld.	<b>CONST</b>
<b>&lt;Mem_Type&gt;</b>	Speicher, in dem die Variablen abgelegt werden: <code>DRAM_Extern</code> : externer Datenspeicher. <code>SRAM_Extern</code> : externer SRAM-Speicher bei Pro II-Modulen (anstelle des DRAM). <code>DM_Local</code> : interner Datenspeicher (Default).	<b>LONG</b>

### Bemerkungen

Bei einem Feld können Sie auf die Elemente 1...**dim** zugreifen. Das Feldelement [0] dürfen Sie nicht verwenden, weil es für interne Zwecke benutzt wird.  
Die maximale Feldgröße ist abhängig vom verfügbaren physikalischen Speicher auf dem *TiCo*-Prozessor.

### Siehe auch

`Dim`, `RingBuffer`, „Globale Felder (Arrays)“ auf Seite 80, „Variablen und Felder im Datenspeicher“ auf Seite 83

### Beispiel

```
Rem Dimensioniere das globale Feld Data_15 mit
Rem 1000 Long-Elementen
Dim Data_15[1000] As Long
```

## Dec

**Dec** verringert den Wert einer Long-Variablen um 1.

### Syntax

```
Dec (var)
```

### Parameter

var

Name einer lokalen oder globalen Long-Variablen

**VAR**

**CONST**

LONG

### Bemerkungen

Die Anweisung **Dec** (var) führt zum gleichen Ergebnis wie die Programmzeile: `val=val-1`. Außerdem kann die Anweisung **Dec** eine geringere Ausführungszeit haben.

### Siehe auch

Inc, - (Subtraktion)

### Beispiel

```
Dim index As Long
Dim Data_1[1000] As Long

Init:
index=1000

Event:
  DAC(1,Data_1[index]) 'Wert auf DAC1 ausgeben
  Dec(index)           'index um 1 verringern
  If (index<1) Then
    index=1000         'Nach 1000 Ausgaben von
  EndIf               'vorne beginnen
```

## #Define

**#Define** ersetzt im Quelltext einen symbolischen Namen durch einen frei definierbaren Ausdruck, z. B. eine Konstante.

### Syntax

```
#Define name expression
```

### Parameter

name	Symbolischer Name, <i>ohne</i> Hochkommata. Sonderzeichen sind nicht erlaubt, nur alphanumerische Zeichen (a...z, A...Z, 0...9) und der Unterstrich (_).	CONST STRING
expression	Ausdruck, für den der symbolische Name steht; <i>ohne</i> Hochkommata. Alle Zeichen sind erlaubt.	CONST STRING

### Bemerkungen

Stellen Sie diese Anweisung an den Beginn eines Quelltextes.

Die Funktion **#Define** ist ein Präprozessor-Befehl, d. h. die Ersetzung findet statt, wenn Sie den Quelltext kompilieren lassen (noch bevor der Compiler das lauffähige Programm erzeugt). Verwenden Sie die Funktion, um im Quelltext aussagekräftige Namen anstelle von Konstanten, Parametern oder Berechnungsausdrücken zu verwenden.

Die erste Zeichenfolge bis zu einem Leerzeichen wird als symbolischer Name interpretiert, der nachfolgende Zeileninhalt bis zum Zeilenumbruch als einzufügender Ausdruck<sup>1</sup>. Der Ausdruck wird exakt so eingefügt, wie Sie ihn definiert haben; Variablennamen im Ausdruck werden also nicht durch deren aktuellen Wert, sondern als Zeichenfolge ersetzt.

Groß- und Kleinschreibung wird beim Suchen und Ersetzen nicht unterschieden.

Wenn Sie für **expression** einen Rechenausdruck einsetzen, empfehlen wir, diesen mit Klammern zu umgeben. Sie vermeiden damit

---

1. Text hinter einem Kommentarzeichen „*/\**“ wird vom Compiler ignoriert.

eventuelle Fehler im Zusammenhang mit weiteren Rechenausdrücken.

### Siehe auch

#Include

### Beispiel

```
#Define setpoint Par_1 'Kommentar, wird nicht  
                          'ersetzt
```

```
#Define measured Data_1  
#Define const 12441223
```

Mit diesen Anweisungen können Sie im Quelltext anstelle von `Par_1`, `Data_1` und der Ziffernfolge die Namen `setpoint`, `measured` und `const` verwenden.

```
#Define Sollwert (13 + 4^3)  
Par_1 = 2 * Sollwert    '= 2 * (13 + 4^3)
```

Ohne die Klammern im `#Define`-Ausdruck würden Sie statt des erwarteten Ergebnisses „154“ den Wert „90“ erhalten.

## Dim

`Dim` deklariert ein oder mehrere

- *lokale* Variablen
- *lokale* eindimensionale Felder
- *globale* eindimensionale Felder `Data_n[n]` (auch RingBuffer-Felder)

`Data_n[n][m]` Grundlagen zu Variablen und Datentypen finden Sie in Kapitel 4.2.3 auf Seite 78 sowie Informationen zu RingBuffer-Feldern unter dem Stichwort RingBuffer auf Seite 178.

### Syntax

```
Dim var1 {, var2, ...} As Long
```

```
Dim array1[dim1] As Long
```

```
{At <Mem_Type>}
```

```
Dim Data_n[dim1] As Long
```

```
{As RingBuffer_For_Read /
RingBuffer_For_Write} {At <Mem_Type>}
```

### Parameter

<code>var1, var2</code>	Namen der deklarierten Variablen	
<code>array1,</code> <code>array2,</code> <code>Data_n</code>	Namen der deklarierten Felder. Für <code>Data_n</code> kann <code>n</code> aus 1...16 gewählt werden.	
<code>dim1, dim2</code>	Feldgröße: Anzahl ( $\geq 1$ ) der Feldelemente vom Typ <code>Long</code> .	<b>CONST</b> <b>LONG</b>
<code>&lt;Mem_Type&gt;</code>	Speicher, in dem die Variablen abgelegt werden: <code>DRAM_Extern</code> : externer Datenspeicher <code>SRAM_Extern</code> : externer SRAM-Speicher bei Pro II-Modulen (anstelle von DRAM) <code>DM_Local</code> : interner Datenspeicher (Default)	

### Bemerkungen

Die globalen Variablen `Par_n` dürfen nicht deklariert werden, weil sie vordefiniert sind.

Wenn Sie von der ADwin CPU oder aus mehreren Prozessen auf Daten zugreifen wollen, ist dies nur über *globale* Variablen und Felder möglich.

Die Datenstruktur `RingBuffer` ist geeignet, um große Datenmengen kontinuierlich und schnell zu übertragen, zwischen ADwin CPU und TiCo-Prozessor, zwischen TiCo-Prozessor und externem Speicher oder zwischen zwei TiCo-Prozessen.



Der Umgang mit Ringspeichern ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung des Datentyps `RingBuffer` ist daher erfahrenen Benutzern von *TiCoBasic* und *ADbasic* vorbehalten.

Bitte beachten Sie die Hinweise in Kapitel 4.3.3 auf Seite 85.

Bei einem Feld können Sie auf die Elemente 1...`Dim` zugreifen. Das Feldelement [0] dürfen Sie nicht verwenden, weil es für interne Zwecke benutzt wird.

Die maximale Feldgröße ist abhängig vom verfügbaren physikalischen Speicher auf dem TiCo-Prozessor.

## Siehe auch

Data\_n, Event:, RingBuffer, Finish:, Init:, „Variablen und Felder im Datenspeicher“ auf Seite 83

## Beispiel

```
Rem Dimensioniere var1 als Long-Variable  
Dim var1 As Long  
  
Rem Dimensioniere das Feld array1 mit 1000  
Long-Elementen  
Dim array1[1000] As Long  
  
Rem Dimensioniere das globale Feld Data_15 mit  
Rem 1007 Long-Elementen als Lese-Ringspeicher  
Dim DATA_15[1007] As Long As RingBuffer_For_Read
```

## Do ... Until

**Do...Until** definiert eine Schleife, deren Anweisungsblock mindestens einmal durchlaufen werden. Die Schleife wird abgebrochen, wenn die Abbruchbedingung den Wert „Wahr“ hat.

### Syntax

```
Do
    ...
Until (condition)
```

*'Anweisungsblock*

### Parameter

**condition**      Boolesche Abbruchbedingung mit den Operatoren <, >, =, **And** und **Or**.

LOGIC

### Siehe auch

< = > (Vergleich), And, Or, For ... To ... {Step ... } Next, SelectCase

### Bemerkungen

Sie können **Do...Until**-Schleifen beliebig tief verschachteln; nur die Speichergröße kann Sie hierbei begrenzen.

Vermeiden Sie Schleifen mit langer Ausführungszeit in hochprioren Prozessen, weil diese nicht unterbrochen werden können.

### Beispiel

```
Dim count As Long
Dim DATA_1[103] As Long As RingBuffer_For_Write

Init:
    count = 1

Event:
    Do
        Data_1 = ADC(1,4)
        Inc count
    Until (count > 100)
```

*'Schleife beginnen  
'Messwert auslesen  
'Zählvariable erhöhen  
'100 Messungen  
'durchgeführt?*



## End

`End` beendet einen Prozess.

### Syntax

`End`

### Bemerkungen

Der Befehl `End` beendet die Ausführung des Abschnitts sofort. `End` ist in allen Programmabschnitten gültig.

Wenn der Befehl im Abschnitt **Event:** benutzt wird, beginnt danach die Ausführung des Abschnitts **Finish:** (sofern vorhanden). Wenn im Abschnitt **Event:** nach der Anweisung `End` weitere Programmzeilen folgen, werden sie nicht mehr ausgeführt.

### Siehe auch

`ProcessN_Running`,

### Beispiel

```
Event:  
  If (ADC(1) > 3000) Then 'Messen und Vergleichen  
    End                  'Prozess beenden, aber  
                        'Finish:  
  EndIf                 'noch ausführen  
  
Finish:  
  Set_Digout(1)         'Dig. Ausgang 1 setzen
```

## Event:

Das Kennwort **Event:** bezeichnet den Anfang des Haupt-Programmabschnitts, der bei jedem Event-Signal aufgerufen wird.

### Syntax

**Event:**

### Parameter

- / -

### Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 75.

Der Programmabschnitt **Event:** ist der zentrale Funktionsabschnitt, der im Prozess (typischerweise) in regelmäßigen Abständen aufgerufen wird, bis er gestoppt wird. Je nach Einstellung wird der Aufruf durch einen zyklischen Timer-Event oder durch einen externen Event ausgelöst. Näheres ist in Kapitel 6 „Prozesse im Betriebssystem“ beschrieben.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

### Siehe auch

Dim, Init:, Finish:

### Beispiel

```
Dim val_1 As Long
```

**Event:**

```
val_1 = -5
```

der Abschnitt

## Finish:

Das Kennwort **Finish:** bezeichnet den Anfang des Programmabschnitts zur Schlussbearbeitung.

### Syntax

**Finish:**

**Parameter-** / -

### Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 75.

Der Programmabschnitt **Finish:** wird einmalig durchlaufen, sobald der Prozess gestoppt wird.

Wenn der letzte Befehl im Abschnitt **Finish:** abgearbeitet ist, vergeht noch eine bestimmte Zeit, bis der Prozessstatus „gestoppt“ erreicht ist.

Im Unterschied zu *ADbasic* hat der Abschnitt **Finish:** die Priorität, die für den Prozess gewählt ist.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

### Siehe auch

Dim, Init:, Event:, ProcessN\_Running

### Beispiel

```
Dim val_1 As Long
```

**Finish:**

```
val_1 = -5
```

## For ... To ... {Step ... } Next

**For...Next** definiert eine Schleife, die eine bestimmte Zahl an Durchläufen haben sollen.

### Syntax

```
For i = X To Y {Step Z}
    ...
Next i
```

*'Anweisungsblock*

### Parameter

<b>i</b>	lokale Zählvariable	LONG
<b>X</b>	Startwert der Laufvariablen	LONG
<b>Y</b>	Endwert der Laufvariablen	LONG
<b>Z</b>	Schrittweite ( $\geq 1$ ) der Laufvariablen; Vorgabewert: 1	LONG

### Bemerkungen

Der Anweisungsblock wird in jedem Fall einmal ausgeführt, auch wenn der Startwert **X** größer als der Endwert **Y** ist.

Deklarieren Sie die Zählvariable als lokale Variable (Datentyp **Long**).

Ein hochpriorer Prozess kann von keinem anderen Prozess unterbrochen werden, auch wenn gerade eine zeitaufwändige Schleife bearbeitet wird. Während dieser Zeit kann der *TiCo*-Prozessor nicht auf andere Events reagieren. Die Schleife darf deshalb in hochpriorien Prozessen nur verwendet werden, wenn die Anzahl der Schleifendurchläufe niedrig gehalten wird.

### Siehe auch

Do ... Until, If ... Then ... {Else ... } EndIf, SelectCase

### Beispiel

- / -

## Function ... EndFunction

`Function...EndFunction` definiert ein Funktions-Makro mit Übergabeparametern und einem Rückgabewert.

### Syntax

```
Function macro_name ({val_1, val_2, ...}) As Long

    {Dim var As Long}

    ...                               'Anweisungsblock

    macro_name = ... 'Rückgabewert zuweisen
EndFunction
```

### Parameter

`macro_name` Name der Funktion und Rückgabewert, Datentyp `Long`

`val_1, val_2` Namen der Übergabeparameter; für Felder ist die Syntax mit Dimensionsklammern erforderlich: `array[]` oder `Data_n[]`.

LONG
------

### Bemerkungen

Allgemeine Informationen über Makros finden Sie in Kapitel 4.5.1 auf Seite 95.

Diese Anweisung definiert ein Funktions-Makro, d.h. der vollständige Anweisungsblock zwischen `Function` und `EndFunction` wird an der aufrufenden Stelle eingefügt.

Funktionen erhöhen die Übersichtlichkeit Ihres Quelltextes. Beachten Sie aber, dass jeder Funktionsaufruf die kompilierte Datei vergrößert.

Sie können Funktionen an 3 Stellen einfügen:

1. Vor dem Abschnitt `Init:`
2. Nach dem Abschnitt `Finish:`
3. In einer separaten Datei, die Sie mit `#Include` einbinden (aber nur an einer der Stellen, die unter 1. und 2. angegeben sind).

Beachten Sie bitte, dass Sie in Funktionen:

- keine Prozess-Abschnitte wie **Init:**, **Event:**, oder **Finish:** definieren.
- am Anfang lokale Variablen definieren können, die nur innerhalb der Funktion und für die Dauer der Abarbeitung verfügbar sind.  
Eine lokale Variable kann den gleichen Namen haben wie eine Variable, die außerhalb der Funktion definiert wurde.
- dem Funktionsnamen einen Wert zuweisen, damit dieser zum Rückgabewert an die aufrufende Stelle wird.

Eine Funktion wird mit ihrem Namen und allen definierten Argumenten aufgerufen; die Funktion muss in der aufrufenden Programmzeile als Argument verwendet werden, z. B. in einer Zuweisung (siehe Beispiel). Als Argument ist jeder Berechnungsausdruck (auch Felder) zulässig, solange er den passenden Datentyp hat.

Wenn Sie keine Argumente definieren, müssen Sie dennoch beim Aufruf der Funktion die Leerklammern verwenden: `name()`.

Wenn ein Feld als Übergabeparameter einer Funktion verwendet wird, ist die Syntax für Aufruf und Definition unterschiedlich:

- Funktions-Aufruf *ohne* Dimensions-Klammern:  
`ret_val = name(array_pass)`
- Funktions-Definition *mit* Dimensions-Klammern:  
`Function name(array_def[])`

Werte werden an Feldelemente (eines Felds als Übergabeparameter) zugewiesen wie gewöhnlich:

```
array_def[2] = value
```

Wenn Sie einem Übergabeparameter `x` in der Funktion einen Wert zuweisen, darf beim Funktionsaufruf für `x` keine Konstante angegeben werden, sondern nur eine Variable oder ein einzelnes Feld-Element. Auf diese Weise können Übergabeparameter auch einen Rückgabewert enthalten.

Bei Berechnungsausdrücken in einer Funktion sollten die Übergabeparameter in Klammern stehen. Auf diese Weise vermeiden Sie Probleme mit der Rangfolge von Operatoren (z. B. Punkt- vor Strich-Rechnung).

## Siehe auch

#Include, Sub ... EndSub

### Beispiel

```
Function sumsquare(w1, w2, w3) As Long
    Rem Die Funktion berechnet das Summenquadrat aus den
        Werten
    Rem w1, w2 und w3
    Dim sum As Long
    sum = w1 + w2 + w3
    sumsquare = sum * sum
EndFunction
```

Ein Aufruf der Funktion erfolgt z.B. mit den Programmzeilen:

```
x = sumsquare(x1, x2, x3)
DAC(1, sumsquare(x1, x2, x3))
```

Die gleiche Funktion mit einem Feld als Übergabe-Parameter:

```
Function sumsquare_array(array[]) As Long
    sum = array[1] + array[2] + array[3]
    sumsquare_array = sum * sum
EndFunction
```

Der Aufruf dieser Funktion erfolgt wieder in ähnlicher Weise (allerdings *ohne* die Dimensionsklammern):

```
x = sumsquare_array(array)
DAC(1, sumsquare_array(array))
```

Beim Aufruf können Sie für `array` ein globales oder ein lokales Feld angeben. Tragen Sie nur den Feldnamen ein ohne Elementnummer und eckige Klammern.

## If ... Then ... {Else ... } EndIf

Die Kontrollstruktur bewirkt in Abhängigkeit von einer Bedingung die Ausführung einer Anweisung (**If...Then...**) oder eines Anweisungsblocks (**If...Then...Else...EndIf**).

### Syntax

```
If (condition) Then
    ...                               'Anweisungsblock
{Else                               'Der else-Teil ist optional
    ...                               'Anweisungsblock }
EndIf
oder
If (condition) Then instr
```

### Parameter

<b>condition</b>	Boolesche Bedingung mit den Operatoren <, >, =, <b>And</b> und <b>Or</b> . Wenn die Bedingung „Wahr“ ist, werden die Anweisungen nach <b>Then</b> ausgeführt.	LOGIC
<b>instr</b>	Anweisung (entspricht einer Befehlszeile).	

### Bemerkungen

Sie können **If**-Strukturen beliebig tief verschachteln; nur die Speichergröße begrenzt Sie hierbei.

Der Anweisungsblock nach **Else** wird (falls vorhanden) schneller ausgeführt als der nach **If...Then**. Dies beschleunigt die Gesamt-Ausführungszeit des **Event**:-Abschnitts, weil die Bedingung meistens den Wert „Falsch“ hat, z.B. bei der Prüfung auf Überschreiten von Grenzwerten.

In der einzeiligen Variante darf die Anweisung weder ein Unterprogramm-Makro (**Sub**) noch ein Funktion-Makro (**Function**) aufrufen.

### Siehe auch

< = > (Vergleich), And, Or, Do ... Until, SelectCase



## Beispiel

```
Dim val As Long           'Deklaration

Event:
    val = ADC(1)           'Messwert erfassen

    If (val > 3000) Then    'Grenzwert überschritten:
        Clear_Digout(1)    'Rücksetzen Digout 1
        Set_Digout(0)      'Setzen Digout 0
    Else                   'Grenzwert nicht
                            'überschritten:
        Clear_Digout(0)    'Rücksetzen Digout 0
        Set_Digout(1)      'Setzen Digout 1
    EndIf                  'Kontrollstruktur Ende
```

## #If ... Then ... {#Else ... } #EndIf

Die Präprozessor-Anweisung bewirkt in Abhängigkeit von einer Bedingung die Kompilierung eines Anweisungsblocks (**#If...Then...#Else...#EndIf**).

### Syntax

```
#If condition Then
...
                                'Anweisungsblock
{#Else
                                'Der else-Teil ist optional
...
                                'Anweisungsblock}
#EndIf
```

### Parameter

**condition**      Boolesche Bedingung (ohne Klammern und Hochkommas) in der Form:

**<SYSPAR> = value**

Wenn die Bedingung erfüllt ist, werden die folgenden Anweisungen ausgeführt.

Die Systemparameter **<SYSPAR>** und die zugehörigen Werte **value** sind in der Tabelle unten aufgeführt.

LOGIC

<b>&lt;SYSPAR&gt;</b>	<b>value</b>	<b>Bedeutung</b>
<b>ADwin_ SYSTEM</b>	<b>ADWIN_GOLDII ADWIN_PROII</b>	Einstellung „System“ im Fenster „Compiler Options“.
<b>Processor</b>	<b>Ticol</b>	Einstellung „Processor“ im Fenster „Compiler Options“.

### Bemerkungen

In der Bedingung darf nur der Operator „=“ verwendet werden; weder Boolesche Verknüpfungen mit **And** und **Or** noch Klammerungen sind erlaubt. Sie können **#If**-Strukturen beliebig tief verschachteln; nur die Speichergröße begrenzt Sie hierbei.

Es gibt keine einzeilige Variante wie bei **If...Then**.

Bei einem Kommandozeilen-Aufruf des Compilers (siehe Seite A-9) beziehen sich die Systemparameter auf die beim Aufruf übergebenen Parameter **/Sx** und **/Px**.

### Siehe auch

< = > (Vergleich), If ... Then ... {Else ... } EndIf

### Beispiel

```
Rem Processdelay auf 800µs setzen
#If Processor = Ticol Then
    Rem 800µs = 40000 x 20ns
    Processdelay = 40000
#EndIf
```

## Import

`Import` bindet Funktionen und Unterprogramme aus der angegebenen Library-Datei bei der Kompilierung mit ein.

### Syntax

```
Import {path}file
```

### Parameter

<code>file</code>	Dateiname der Library-Datei <i>ohne</i> Hochkommas. Die Dateiendung ist <code>.TL1</code> für Tico1.	<b>CONST</b> <b>STRING</b>
<code>path</code>	Vollständiger Pfad mit Laufwerk oder relativer Pfad der Library-Datei; <i>ohne</i> Hochkommas	<b>CONST</b> <b>STRING</b>

### Bemerkungen

Fügen Sie `Import`-Anweisungen ganz am Anfang Ihres Quelltextes ein (vor der Variablendeklaration). Wenn Sie im Quelltext einer Library-Datei weitere Library-Dateien importieren, müssen Sie dies zusätzlich auch im aufrufenden Quelltext tun.

Es werden nur diejenigen Funktionen und Unterprogramme aus der Library-Datei eingebunden, die Sie in Ihrem Quelltext aufrufen.

Wenn Sie keinen Pfadnamen angeben, wird die Datei nur im Standard-Verzeichnis (siehe Menü Options `Directories`, Seite 56) gesucht. Verwenden Sie bei der Pfadangabe den Backslash "`\`", um Verzeichnisnamen voneinander zu trennen.

Das Basisverzeichnis für relative Pfadangaben ist – wenn der Quelltext Teil einer Projektdatei ist – das Verzeichnis der Projektdatei, anderenfalls das Verzeichnis der Quelltextdatei.

Die folgenden Bibliotheken sind im Lieferumfang von *TiCoBasic* enthalten:

`math.tl1`                      Mathematik-Befehle.

### Siehe auch

`#Include`

### Beispiel

```
Import math.tl1           'Mathematik-Library
                          'importieren
Rem Benutzerdefinierte Library importieren
Import C:\MyFiles\ADwinLibs\dig2volt.tl1
```

## In

**In** gibt den Inhalt einer bestimmten Speicherstelle im I/O-Adressbereich des *TiCo*-Prozessors zurück.

### Syntax

```
ret_val = In(addr)
```

### Parameter

**addr** Adresse der auszulesenden Speicherstelle.

LONG

**ret\_val** Inhalt der Speicherstelle.

LONG

### Bemerkungen

Sie benötigen den Befehl **In** in aller Regel nicht. Nutzen Sie die Befehle der Include-Dateien, um den *TiCo*-Prozessor zu steuern.

**In** ist nur für spezielle Anwendungen vorgesehen, die in Zusammenarbeit mit unserem Support erstellt werden. Die Dokumentation enthält daher keine Beschreibung von Speicherstellen des *TiCo*-Prozessors.

Wenn in einem Projekt ein extern gesteuerter Prozess enthalten ist, darf **In** nur in einem hochprioren Prozess eingesetzt werden.

### Siehe auch

Out

### Beispiel

*Rem Das Beispiel zeigt nur die Verwendung der Befehle In  
Rem und Out nur symbolisch. Führen Sie das Programm nicht  
aus!*

#### Event:

```
If (In(100h) <> 0) Then 'Speicherstelle 100h auslesen
    Out (0, 255)         'Adresse 0 auf den Wert 255
                        'setzen
Else
    Out (0, 0)           'Adresse 0 auf den Wert 0
                        'setzen
EndIf
```

## Inc

**Inc** erhöht den Wert einer lokalen oder globalen ganzzahligen Variablen um Eins.

### Syntax

```
Inc (var)
```

### Parameter

**var** Name einer lokalen oder globalen Long-Variablen

**VAR****CONST**

LONG

### Bemerkungen

Die Anweisung **Inc** (var) führt zum gleichen Ergebnis wie die Programmzeile: **var** = **var** + 1. Allerdings kann diese Anweisung eine geringere Ausführungszeit haben.

### Siehe auch

Dec, + (Addition)

### Beispiel

```
Dim index As Long
Dim Data_1[1000] As Long

Init:
    index=1

Event:
    Data_1[index] = ADC(1) 'Messwert im Feld ablegen
    Inc(index)           'index um 1 erhöhen
    If (index>1000) Then End 'Nach 1000 Messungen das
                             'Programm
                             'beenden
```

## #Include

**#Include** bindet den vollständigen Inhalt einer Include-Datei in den Quelltext ein.

### Syntax

```
#Include {path}filename
```

### Parameter

<code>filename</code>	Name der einzubindenden Datei (mit Endung „.Inc“) ohne Hochkommas	<b>CONST</b> <b>STRING</b>
<code>path</code>	Vollständiger Pfad mit Laufwerk oder relativer Pfad.	<b>CONST</b> <b>STRING</b>

### Bemerkungen

Allgemeine Informationen über Include-Dateien finden Sie in Kapitel 4.5.2 auf Seite 95.

Fügen Sie **#Include**-Anweisungen ganz am Anfang Ihres Quelltextes ein (vor der Variablendeklaration). Im Quelltext einer Include-Datei können Sie weitere Include-Dateien importieren.

Wenn die eingebundene Include-Datei Library-Funktionen verwendet, müssen Sie mit **Import** auch die entsprechenden Library-Dateien einbinden.

Wenn Sie keinen Pfadnamen angeben, wird die Datei nur im Standard-Verzeichnis (siehe Menü Options **Directories**, Seite 56) gesucht. Verwenden Sie bei der Pfadangabe den Backslash "\", um Verzeichnisnamen voneinander zu trennen.

Das Basisverzeichnis für relative Pfadangaben ist – wenn der Quelltext Teil einer Projekts ist – das Verzeichnis der Projektdatei, anderenfalls das Verzeichnis der Quelltextdatei.

Um eine der im Lieferumfang von *TiCoBasic* enthaltenen Include-Dateien – sie enthalten Befehle zur Ansteuerung der Hardware-I/Os – einzubinden, geben Sie die ersten Zeichen des Befehls **#Include** ein, drücken [CTRL][SPACE] und wählen die passende Include-Datei aus der Liste.



Beachten Sie bitte: Eine Zeile mit **#Include**-Anweisung darf höchstens 136 Zeichen lang sein (Zeilenlänge für alle anderen Zeilen, siehe Seite 125). Alle weiteren Zeichen der Zeile schneidet der Compiler ab.

### Siehe auch

#Define, Import, Function ... EndFunction, Sub ... EndSub

### Beispiel

*Rem Datei im angegebenen Verzeichnis suchen*

```
#Include C:\Test\demofunc.Inc
```

*Rem Datei im Standard-Verzeichnis suchen*

```
#Include demofunc.Inc
```

*Rem Relative Pfadangabe.*

*Rem Der Pfad ist relativ zum Verzeichnis der  
Projektdatei (wenn*

*Rem der Quelltext zu einem Projekt gehört).*

*Rem Gehört der Quelltext nicht zu einem Projekt, ist*

*Rem der Pfad relativ zum Verzeichnis der  
Quelltextdatei.*

```
#Include .\demofunc.Inc
```

## Init:

Das Kennwort **Init:** bezeichnet den Anfang eines Programmabschnitts zur Initialisierung.

### Syntax

**Init:**

### Parameter- / -Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 75.

Der Programmabschnitt **Init:** wird einmal durchlaufen, sobald der Prozess gestartet wird. Die Zeit zwischen dem letzten Befehl im Abschnitt **Init:** und dem Beginn des Abschnitts **Event:** beträgt etwas mehr als  $1 \times \text{Processdelay}$ .

Der Programmabschnitt hat die für den Prozess eingestellte Priorität (Menüpunkt „Options / Process“). Bei hoher Priorität kann der Programmabschnitt nicht unterbrochen werden und sollte dann möglichst kurz sein.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

### Siehe auch

Dim, Event:, Finish:, Processdelay

### Beispiel

```
Dim val_1 As Long
Init:
    val_1 = -5
```

## Lib\_Function ... Lib\_EndFunction

`Lib_Function...Lib_EndFunction` definiert in einer Library-Datei eine Funktion mit Übergabe- und Rückgabeparametern.

### Syntax

```
Lib_Function lib_name(<LIB_PAR1> {,  
    <LIB_PAR2>, ...} )  
As <FCT_TYPE>  
  
    {Dim var As <VAR_TYPE>}  
  
    ...                               'Anweisungsblock  
  
    lib_name = ...  
Lib_EndFunction
```

Syntax der Übergabeparameter `<LIB_PAR>`:  
`<BY_TYPE> var_name As <VAR_TYPE>`

### Parameter

<code>lib_name</code>	Name der Library-Funktion und des Rückgabewerts; Datentyp <code>&lt;FCT_TYPE&gt;</code> .
<code>&lt;FCT_TYPE&gt;</code>	Datentyp: <code>Float</code> , <code>LONG</code> .
<code>var_name</code>	Name eines Übergabeparameters innerhalb der Library-Funktion; für Felder ist die Syntax mit Dimensionsklammern erforderlich: <code>array[]</code> oder <code>Data_n[]</code> .
<code>&lt;BY_TYPE&gt;</code>	Methode zur Übergabe der Parameter: <code>BYREF</code> : Zeiger auf Variable oder Feld übergeben. <code>BYVAL</code> : Wert übergeben.
<code>&lt;VAR_TYPE&gt;</code>	Datentyp: <code>Float</code> , <code>Long</code> , <code>String</code> .

### Bemerkungen

Allgemeine Informationen über Library-Dateien finden Sie in Kapitel 4.5.3 auf Seite 96.

Erstellen Sie Library-Funktionen (und -Unterprogramme) in einer eigenen Quelltext-Datei. Nach der Kompilierung mit „Build / Make lib file“ können Sie mit dem Befehl `Import` diejenigen Module einer Library in einen Prozess einbinden, die dort tatsächlich aufgerufen werden.

Innerhalb einer Library-Funktion können Sie

- lokale Variablen und Felder deklarieren und verwenden. Deklarieren Sie Variablen immer am Anfang, keinesfalls außerhalb des Unterprogramms.
- globale Variablen und Felder verwenden, wenn sie als Parameter übergeben werden.
- dem Funktionsnamen einen Wert zuweisen, damit dieser zum Rückgabewert an die aufrufende Stelle wird.

Innerhalb einer Library-Funktion ist *nicht* erlaubt:

- Prozess-Abschnitte wie `Init:`, `Event:`, oder `Finish:` definieren.
- Library-Funktion oder -Unterprogramm aus der gleichen Library-Datei aufrufen. Gegebenenfalls müssen Sie die aufzurufende Funktion in eine neue Library-Datei auslagern und von dort importieren.
- Anweisung `SelectCase` benutzen.
- Symbolische Namen mit `#Define` deklarieren.
- Auf Hardware zugreifen wie analoge oder digitale Ein- oder Ausgänge.

Die Methoden zur Übergabe von Parametern unterscheiden sich folgendermaßen:

- `BYREF`: Die Library-Funktion kann den Parameter verändern, so dass der geänderte Wert anschließend im aufrufenden Programm vorliegt (es wird die Adresse des Parameters übergeben).
- `BYVAL`: Die Library-Funktion kann nur auf den Wert des Parameters zugreifen, diesen aber selbst nicht ändern. Der Parameter bleibt also für das aufrufende Programm gleich.

Wenn ein Feld als Übergabeparameter einer Library verwendet wird, ist die Syntax für Definition und Aufruf unterschiedlich:

- Definition des Funktionsparameters *mit* Klammern:  
`Lib_Function name(array[]) ...`
- Funktionsaufruf mit dem Parameter *ohne* Klammern:  
`ret_val = name(array)`

Definieren Sie Felder als Übergabeparameter immer **BYREF** und ohne Feldgröße. Sie können keine FIFO-Felder als Übergabeparameter verwenden.

### Siehe auch

Lib\_Sub ... Lib\_EndSub, Import, Function ... EndFunction, Sub ... EndSub

### Beispiel

```
Rem ----- Mittelwertbildung -----  
Lib_Function average(BYREF array[] As Long, BYVAL ptr  
    As Long,  
        BYVAL cnt As Long) As Long  
    Dim i As Long  
    average = 0  
    If (cnt > 0) Then  
        For i = ptr To (ptr + cnt)  
            average = average + array[i]  
        Next i  
        average = average / cnt  
    EndIf  
Lib_EndFunction
```

Den Aufruf der Library-Funktion **average** sehen Sie im folgenden Beispiel, einem sogenannten „moving average filter“:

```

Rem Library 'MEAN' importieren
Import C:\MyFiles\ADwinLibs\MEAN.tll
#Define cnt 10          'Anzahl der Summanden
                        '(Samples)

#Define samples Data_1 'Anzahl Messwerte
#Define filtered Data_2 'Anzahl gefilterte Messwerte
#Define length 1000    'Feldlänge
Dim samples[length] As Long 'Quell-Feld
Dim filtered[length] As Long 'Ziel-Feld
Dim i As Long          'Zählvariable

Init:
    i = 1               'Zählvariable
                        'initialisieren
    Processdelay = 40000 'Messung mit 1 kHz

Event:
    samples[i] = ADC(1) 'Analogwerte messen und
                        'speichern
    Inc i               'Zählvariable erhöhen
    If (i > length) Then End '1000 Messungen durchgeführt?
                        'Wenn ja: Finish
                        'abarbeiten

Finish:
    For i = 1 To (length - cnt) 'Alle Messwerte aufrufen
        Rem Library-Funktion "average" aufrufen
        filtered[i + cnt] = average(samples,i,cnt)
        Rem Beachten Sie die Syntax beim Feld 'samples'
        Rem als Übergabeparameter ohne eckige Klammern
    Next i

```

## Lib\_Sub ... Lib\_EndSub

`Lib_Sub...Lib_EndSub` definiert in einer Library-Datei ein Unterprogramm mit Übergabeparametern.

### Syntax

```
Lib_Sub lib_name(<LIB_PAR1> {, <LIB_PAR2>, ...})
    {Dim var As <VAR_TYPE>}
    {#Define name expression}
    ...                               'Anweisungsblock
Lib_EndSub
```

Syntax der Übergabeparameter `<LIB_PAR>`:

```
<BY_TYPE> var_name As <VAR_TYPE>
```

### Parameter

<code>lib_name</code>	Name des Library-Unterprogramms.
<code>var_name</code>	Name eines Übergabeparameters innerhalb des Library-Unterprogramms; für Felder ist die Syntax mit Dimensionsklammern erforderlich: <code>array[]</code> oder <code>Data_n[]</code> .
<code>&lt;BY_TYPE&gt;</code>	Methode zur Übergabe eines Parameters: <code>BYREF</code> : Zeiger auf Variable oder Feld übergeben. <code>BYVAL</code> : Wert übergeben.
<code>&lt;VAR_TYPE&gt;</code>	Datentyp: <code>Float</code> , <code>Long</code> , <code>String</code> .

### Bemerkungen

Allgemeine Informationen über Bibliotheken (Libraries) finden Sie in Kapitel 4.5.3 auf Seite 96.

Erstellen Sie Library-Unterprogramme (und -Funktionen) in einer eigenen Quelltext-Datei. Mit „Build / Make lib file“ kompilieren sie diese und erzeugen die Library-Datei. Der Befehl `Import` bindet diejenigen Module einer Library in einen Prozess ein, die dort tatsächlich aufgerufen werden.

Innerhalb eines Library-Unterprogramms können Sie

- lokale Variablen und Felder deklarieren und verwenden. Deklarieren Sie Variablen immer am Anfang, keinesfalls außerhalb des Unterprogramms.
- globale Variablen und Felder verwenden, wenn sie als Parameter übergeben werden.

Innerhalb eines Library-Unterprogramms ist *nicht* erlaubt:

- Prozess-Abschnitte wie **Init:**, **Event:**, oder **Finish:** definieren.
- Library-Funktion oder -Unterprogramm aus der gleichen Library-Datei aufrufen.  
Gegebenenfalls müssen Sie die aufzurufende Funktion in eine neue Library-Datei auslagern und von dort importieren.
- die Anweisung **SelectCase** benutzen.
- Symbolische Namen mit **#Define** deklarieren.
- Auf Hardware zugreifen wie analoge oder digitale Ein- oder Ausgänge.

Die Methoden zur Übergabe von Parametern unterscheiden sich folgendermaßen:

- **BYREF**: Das Library-Unterprogramm kann den Parameter verändern, so dass der geänderte Wert anschließend im aufrufenden Programm vorliegt (es wird die Adresse des Parameters übergeben).
- **BYVAL**: Das Library-Unterprogramm kann nur auf den Wert des Parameters zugreifen, diesen aber selbst nicht ändern. Der Parameter bleibt also für das aufrufende Programm gleich.

Wenn ein Feld als Übergabeparameter einer Library verwendet wird, ist die Syntax für Definition und Aufruf unterschiedlich:

- Definition des Unterprogrammparameters *mit* Klammern: **LIB\_Sub** **subname**(**array**[]) ...
- Aufruf mit dem Parameter *ohne* Klammern:  
**subname**(**array**)

Definieren Sie Felder als Übergabeparameter immer **BYREF** und ohne Feldgröße. Sie können keine FIFO-Felder als Übergabeparameter verwenden.

### Siehe auch

Lib\_Function ... Lib\_EndFunction, Import, Function ... EndFunction,  
Sub ... EndSub



**Beispiel:**

```
Rem Messwertumrechnung von Digit(0...65535) nach
Volt(±10V)
Lib_Sub dig2volt(BYREF digit[] As Long, BYVAL ptr As
Long,
    BYVAL cnt As Long, BYVAL gain As Long, BYREF volt[]
As Float)
    Dim i As Long
    For i = ptr To (ptr + cnt)
        volt[i] = ((digit[i] * 20 / 65536) - 10) / gain
    Next i
Lib_EndSub
```

Den Aufruf der Library-Funktion **dig2volt** sehen Sie im folgenden Beispiel, einer Messwert-Umwandlung:

```

Rem Die Library 'DIG2VOLT' wird importiert
Import C:\MyFiles\ADwinLibs\DIG2VOLT.tl1
#Define cnt 1000           'Anzahl der Samples
#Define ptr 1              'Erstes zu
                           'konvertierendes Sample
#Define gain 1             'Verstärkungsfaktor des
                           'PGA
#Define samples Data_1    'Speicher für Messwerte
#Define scaled Data_2     'Speicher für
                           'konvertierte Messwerte
#Define length 1000       'Feldlänge
Dim samples[length] As Long 'Quell-Feld
Dim i As Long             'Zählvariable

Init:
    i = 1                  'Zählvariable
                           'initialisieren
    Processdelay = 40000   'Messung mit 1 kHz

Event:
samples[i] = ADC(1)        'Analogwerte messen und
                           'speichern
    Inc i                  'Zählvariable erhöhen
    If (i > length) Then End '1000 Messungen durchgeführt?
                           'Wenn ja: Finish
                           'abarbeiten

Finish:
    Rem Die gewünschten Messwerte konvertieren durch
    Rem Aufruf des Library-Unterprogramms 'dig2volt'
    dig2volt(samples,ptr,cnt,gain,scaled)
    Rem Beachten Sie die Syntax beim Feld 'samples'
    Rem als Übergabeparameter ohne eckige Klammern []

```

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

## Max\_Long

**Max\_Long** liefert den größeren von 2 ganzzahligen Werten.

### Syntax

```
ret_val = Max_Long(val1, val2)
```

### Parameter

val\_1            Vergleichswert 1

LONG

val\_2            Vergleichswert 2

LONG

ret\_val          Der größere der beiden Vergleichswerte.

LONG

### Bemerkungen

- / -

### Siehe auch

Absl, Min\_Long

### Beispiel

**Event:**

```
Par_10 = Max_Long(Par_1, Par_2)
```

## Min\_Long

**Min\_Long** liefert den kleineren von 2 ganzzahligen Werten.

### Syntax

```
ret_val = Min_Long(val1, val2)
```

### Parameter

<code>val_1</code>	Vergleichswert 1	LONG
<code>val_2</code>	Vergleichswert 2	LONG
<code>ret_val</code>	Der kleinere der beiden Vergleichswerte.	LONG

### Bemerkungen

- / -

### Siehe auch

Absl, Max\_Long

### Beispiel

```
Event:  
Par_10 = Min_Long(Par_1, Par_2)
```

### Siehe auch

Dim

## NOP

**NOP** (No OPeration) lässt den Prozessor für die Zeit von einem Taktzyklus warten.

### Syntax

**NOP**

### Bemerkungen

Sie können mit dieser Anweisung erforderliche Wartezeiten überbrücken (beispielsweise nach **Set\_Mux**), wenn es keine sinnvolle andere Verwendung der Prozessorzeit gibt.

### Siehe auch

NOPs, Sleep

## NOPs

**NOPS** lässt den Prozessor für mehrere Taktzyklen warten. Die Wartezeit entspricht der Anzahl mehrerer **NOP**-Befehle.

### Syntax

**NOPS** (*number*)

### Parameter

*number*

Anzahl ( $\geq 1$ ) der einzufügenden NOP-Befehle.  
Für Konstanten ist 32767 der größte mögliche Wert.

LONG


### Bemerkungen

Verwenden Sie **NOPS** nur in hochprioren Prozessen. In niederprioren Prozessen ist **Sleep** die bessere Wahl, vor allem weil bei hohen Werten für *value* unerwartet lange Wartezeiten auftreten können.

Der Befehl **NOPS** benötigt weniger Speicherplatz als die entsprechende Anzahl an **NOP**-Befehlen.

Wenn *number* eine Konstante ist, ersetzt **NOPS** die angegebene Zahl von **NOP**-Befehlen exakt. Bei Verwendung einer Variablen werden 2...4 Taktzyklen zusätzlich benötigt; ebenso werden beim Zugriff auf externen Speicher oder bei Berechnungen weitere Taktzyklen benötigt.

Unter bestimmten Voraussetzungen fügt der Compiler nach **NOPS** automatisch einen zusätzlichen **NOP**-Befehl ein. Wenn stattdessen eine exakte Wartezeit benötigt wird, verringern Sie *number* um 1 und fügen den **NOP**-Befehl (hinter **NOPS**) von Hand ein.

Wenn die Ausführung von **NOPS** unerwartet lange dauert, könnte die Variable *number* einen negativen Wert enthalten. Verwenden Sie stattdessen eine positiven Konstante. 

### Siehe auch

NOP, Sleep

### Beispiel

- / -



## Or

Der Operator `Or` verknüpft zwei ganzzahlige Werte bitweise oder zwei Boolesche Ausdrücke als Boolescher Operator.

### Syntax

```
ret_val = val_1 Or val_2           'Bitweiser Operator
If ((expr1) Or (expr2)) Then      'Boolescher Operator
```

### Parameter

`val_1, val_2` Ganzzahliger Wert

LONG

`expr1, expr2` Boolescher Ausdruck mit dem Wert „wahr“ oder „falsch“

LOGIC

### Bemerkungen

Sie können mit `Or` nur gleichartige Ausdrücke verknüpfen (ganzzahlige oder Boolesche), ein Mischen ist nicht möglich.

Sie können Boolesche Ausdrücke nur mit den Anweisungen `If ... Then ... Else` oder `Do ... Until` verwenden (Variablen können keine Booleschen Werte annehmen).

Wenn Sie in einer Zeile mehrere Boolesche Operatoren verwenden, müssen Sie jede Verknüpfung separat in Klammern setzen. Bei der Verknüpfung ganzzahliger Werte ist dies nicht erforderlich.

### Siehe auch

And, If ... Then ... {Else ... } EndIf, Not, XOr

### Beispiel

Bitweise Operator:

```
Dim val_1, val_2, val3 As Long
```

```
val_1 = 0100b
```

```
val_2 = 0110b
```

```
val3 = val_1 Or val_2 'Ergebnis: val3 = 0110b
```



Boolescher Operator:

```
Dim x As Long
Dim val4 As Long
```

**Init:**

```
x = 15
```

**Event:**

```
If ((x < 3) Or (x > 9)) Then
    val4 = 1
Else
    val4 = 0
EndIf
```

*'Ergebnis: val4 = 1*

## Out

**Out** schreibt einen Wert in eine bestimmte Speicherstelle im I/O-Adressbereich des *TiCo*-Prozessors.

### Syntax

```
Out(addr, value)
```

### Parameter

**addr** Adresse der zu beschreibenden Speicherstelle.

LONG

**value** Zu schreibender Wert.

LONG

### Bemerkungen

Sie benötigen den Befehl **Out** in aller Regel nicht. Nutzen Sie die Befehle der Include-Dateien, um den *TiCo*-Prozessor zu steuern.

**Out** ist nur für spezielle Anwendungen vorgesehen, die in Zusammenarbeit mit unserem Support erstellt werden. Die Dokumentation enthält daher keine Beschreibung von Speicherstellen des *TiCo*-Prozessors.

### Siehe auch

In

### Beispiel

*Rem Das Beispiel zeigt die Verwendung der Befehle IN und Rem OUT nur symbolisch. Führen Sie das Programm nicht aus!*

**Init:**

```
If (In(100h)=0) Then      'Speicherstelle 100h = 0?
  Out(1,12)               'Wert 12 in Adresse 1 schreiben
EndIf
```

## Processdelay

Die Systemvariable **Processdelay** definiert das Processdelay (die Zykluszeit) eines Prozesses.

**Processdelay** ersetzt die Systemvariable **Globaldelay**, die aus Kompatibilitätsgründen weiterhin gültig ist.

### Syntax

```
ret_val = Processdelay
```

oder

```
Processdelay = expr
```

### Parameter

**ret\_val**      Aktuell eingestellte Zykluszeit in Taktzyklen.

LONG

**expr**          Einzustellende Zykluszeit: Anzahl ( $\geq 1$ ) der Taktzyklen.

LONG

### Bemerkungen

Bei einem zeitgesteuerten Prozess wird der Abschnitt **Event**: vom internen Zähler zyklisch und in festen Zeitabständen aufgerufen. Der Zeitabstand zwischen zwei Aufrufen, Zykluszeit oder Processdelay genannt, wird in Zähler-Taktzyklen gezählt.

Die Zeiteinheit des Processdelay ist 20 ns.

Wählen Sie bei hochprioren Prozessen eine ausreichend großes Processdelay, um eine Überlastung des *TiCo*-Prozessors zu vermeiden. Als Faustregel sollte die Auslastung des Prozessors (Anzeige: „Busy x%“ in der Statusleiste) möglichst unter 90% bleiben und darf keinesfalls 100% übersteigen.

Wenn die Bearbeitungszeit des Abschnitts **Event**: größer ist als das Processdelay, kommen der nächste Zähler-Aufruf und die folgenden verspätet. Sollte diese Verzögerung nicht innerhalb von 250 ms aufgeholt werden, kann die Kommunikation zwischen *ADwin*-System und PC zusammenbrechen.

Ein konstantes Processdelay können Sie festlegen, indem Sie der Variablen **Processdelay** im Abschnitt **Init**: einen Wert zuweisen. Sie überschreiben damit ggf. die Voreinstellung, die Sie in der Entwicklungsumgebung *TiCoBasic* im Dialogfenster „Options / Process“ unter „Initial Processdelay“ eingegeben haben.

Sie können die Systemvariable innerhalb eines Abschnitts nur einmal beschreiben.

Wenn der Parameter **Processdelay** innerhalb eines Prozesszyklus, d.h. im Abschnitt **Event**: geändert wird, wird die Zykluszeit dadurch sofort verändert. Dies kann insbesondere bei einer Verkürzung der Zykluszeit kritisch sein: Achten Sie immer darauf, dass die Bearbeitungszeit des Prozesszyklus kürzer bleibt als die neu eingestellte Zykluszeit.

### Siehe auch

Read\_Timer, Kapitel 6.2.1 „Processdelay“

### Beispiel

#### Init:

```
Rem Zykluszeit auf 800 µs einstellen
Processdelay = 40000
...
```

Wenn Sie eine längere Zykluszeit benötigen als mit **Processdelay** einstellbar ist, können Sie eine Hilfsvariable verwenden:

#### Init:

```
Rem Max. Zykluszeit einstellen, etwa 42,9 s
Processdelay = 2147483647
Rem Hilfsvariable initialisieren
Par_1 = 0
```

#### Event:

```
Inc Par_1
Rem 100fache Zykluszeit verwenden: 71,5 Min.
If (Par_1 = 100) Then
  Par_1 = 0
  Rem Programm ausführen
...
EndIf
```

## ProcessN\_Running

Die Systemvariable `ProcessN_Running` gibt den aktuellen Status eines bestimmten Prozesses zurück.

### Syntax

```
ret_val = ProcessN_Running
```

### Parameter

`n` Prozessnummer `n` (0...4)

CONST

LONG

`ret_val` Prozess-Status:  
 1 Prozess läuft  
 0 Prozess ist gestoppt  
 -1 Prozess wird gestoppt

LONG

### Bemerkungen

Diese Systemvariable kann nur gelesen werden.

### Siehe auch

End

### Beispiel

```
Event:
  Rem Status von Prozess 2 ermitteln
  Par_2 = Process2_Running
```

## Read\_Timer

**Read\_Timer** gibt den aktuellen Zählerstand des ADwin-Systems zurück.

### Syntax

```
ret_val = Read_Timer()
```

### Parameter

**ret\_val**      Aktueller Zählerstand in Einheiten von 20ns.

LONG
------

### Bemerkungen

Die Systemvariable kann nur gelesen werden.

Sie können eine Zeitdifferenz aus der Differenz von 2 Zählerständen ermitteln. Beachten Sie dabei bitte, dass ein gelesener Zählerstand nach 42,9 Sekunden wieder erneut erreicht wird.

### Siehe auch

Processdelay

### Beispiel

```
Dim timervalue As Long
```

#### Event:

```
timervalue = Read_Timer()
```

## Rem, '

Die Compiler-Anweisungen *Rem* oder „*'*“ ermöglichen das Einfügen von Kommentaren im Quelltext. Sie bewirken, dass der in einer Programmzeile folgende Text vom Compiler ignoriert wird.

### Syntax

```
Rem comment  
  
instr : Rem comment  
  
instr 'comment
```

### Parameter

<i>comment</i>	Beliebige Zeichenfolge
<i>instr</i>	TiCoBasic-Anweisung

### Bemerkungen

Die Anweisung gilt nur für die Zeile, in der sie benutzt wird. Für einen mehrzeiligen Kommentar müssen Sie jede Zeile einzeln mit der Anweisung *Rem* oder „*'*“ beginnen.

Wenn Sie eine *Rem*-Anweisung hinter einer anderen Anweisung in einer Befehlszeile einfügen, dann trennen Sie beide durch einen Doppelpunkt *:*. Bei dem Kommentarzeichen *'* ist dies nicht erforderlich.

### Beispiel

```
Rem Dies ist ein Kommentar, der mehr als eine  
Rem Textzeile benötigt  
'Dies ist auch eine Kommentarzeile  
Dim min As Long : Rem Kommentar nach einer Anweisung  
Dim max As Long      'Kommentar nach einer Anweisung
```

## RingBuffer

Mit `Dim DATA_n As RingBuffer_For_x` wird ein globales **DATA**-Feld als Ringspeicher zum Lesen oder zum Schreiben dimensioniert.

### Syntax

```
Dim DATA_n[length] As Long As RingBuffer_For_Read
    {At <Mem_Type>}

Dim DATA_n[length] As Long As RingBuffer_For_Write
    {At <Mem_Type>}
```

### Parameter

<b>DATA_n</b>	Name des deklarierten <b>DATA</b> -Felds (n: 1...16).
<b>length</b>	Feldgröße (≥1): Anzahl der Elemente im Feld. Im externen Datenspeicher ist der Wertebereich eingeschränkt. <b>CONST</b>
<b>&lt;Mem_Type&gt;</b>	Speicherbereich, in dem die Variablen abgelegt werden: <b>DRAM_Extern</b> : externer Datenspeicher. Hier ist der Wertebereich für <b>length</b> nur in Schritten einstellbar: $length = 8 \times a + 7; a \geq 0$ <b>SRAM_Extern</b> : externer Datenspeicher bei Pro II-Modulen (anstelle von DRAM). <b>DM_Local</b> : interner Datenspeicher (Default).

### Bemerkungen



Der Umgang mit Ringspeichern ist nicht einfach. Bei falscher Anwendung können schwer auffindbare Fehler entstehen. Die Verwendung des Datentyps `RingBuffer` ist daher erfahrenen Benutzern von *AD-basic* und *TiCoBasic* vorbehalten. Beachten Sie in jedem Fall die Hinweise in Kapitel 4.3.3 auf Seite 85.

Sie können nur **DATA**-Felder als Ringspeicher verwenden. Dadurch kann ein `RingBuffer`-Feld nicht mehr gleichzeitig als „normales“ Feld verwendet werden.

Nach der Dimensionierung sollten Sie den Ringspeicher im Abschnitt **Init**: mit `RingBuffer_Clear` initialisieren.



Für einen Ringspeicher im externen Datenspeicher gilt:

- Wenn Sie eine nicht erlaubte Feldgröße `length` angeben, wird der Ringspeicher automatisch mit der nächstgrößeren, erlaubten Feldgröße dimensioniert. Beispielsweise ändert der Compiler die Feldgröße `[1000]` automatisch in `[1007]`.
- In einem Lese-Ringspeicher sollten Sie die Daten nach der Dimensionierung im Abschnitt `Init:` mit dem Befehl `Refresh_RingBuffer` aktualisieren.

Wenn Sie schneller Daten in einen Ringspeicher schreiben als auslesen, werden alte gespeicherte Daten überschrieben und gehen damit verloren. Zur Abhilfe können Sie die Befehle `RingBuffer_Empty` und `RingBuffer_Full` verwenden.

### Siehe auch

`Dim`, `Data_n`, `Refresh_RingBuffer`, `RingBuffer_Clear`, `RingBuffer_Empty`, `RingBuffer_Full`, „Globale Felder (Arrays)“ auf Seite 80, „Die Datenstruktur RingBuffer“ auf Seite 85

### Beispiel

```
REM Dimension the global array DATA_15 with 1007 LONG elements
REM as read ringbuffer
Dim Data_15[1007] As Long As Refresh_RingBuffer_For_Read
```

## Refresh\_RingBuffer

**Refresh\_RingBuffer** aktualisiert die Daten in einem Lese-Ringspeicher im externen Speicher.

### Syntax

```
Refresh_RingBuffer(data_no)
```

### Parameter

**data\_no**      Nummer (1...16) des Ringspeichers **DATA\_x**.

LONG
------

### Bemerkungen

Bei einem Ringspeicher im internen Speicher sowie bei einem Schreib-Ringspeicher im externen Speicher ist die Aktualisierung mit **Refresh\_RingBuffer** nicht erforderlich.

Bei einem Lese-Ringspeicher im externen Speicher müssen Sie die Daten (vor dem Auslesen) in folgenden Fällen aktualisieren:

- Im Lese-Ringspeicher sind schon Daten vorhanden und es wird zum ersten Mal ein Wert gelesen.
- Nach dem vorherigen Auslesen enthielt der Ringspeicher weniger als 8 Werte und anschließend wurden neue Daten in den Ringspeicher geschrieben.  
Anders gesagt: Sie können die regelmäßige Aktualisierung mit **Refresh\_RingBuffer** vermeiden, wenn der Ringspeicher nach jedem Auslesen eines Werts 8 oder mehr Werte enthält.

Die Aktualisierung richtet in keinem Fall Schaden an, d.h. es können keine Werte doppelt gelesen werden oder verloren gehen. Im Zweifelsfall ist es also besser, einmal zuviel mit **Refresh\_RingBuffer** zu aktualisieren als einmal zuwenig.

### Siehe auch

RingBuffer (Deklaration), RingBuffer\_Clear, RingBuffer\_Empty, RingBuffer\_Full, Kapitel 4.3.3 „Die Datenstruktur RingBuffer“

**Beispiel**

```
Rem Use global array DATA_20 as ringbuffer
Dim DATA_12[999] As Long As Ringbuffer_For_Read At DRAM_Extern
Dim i As Long
```

**Init:**

```
Rem initialize ringbuffer
RingBuffer_Clear(12)
Rem wait until the ringbuffer contains more than 7 values
Do
Until (RingBuffer_Full(12,PAR_1) > 7)
Rem refresh ringbuffer data
Refresh_RingBuffer(12)
```

**Event:**

```
Rem read 500 ringbuffer values, but always have more
Rem than 7 values left in it
If (RingBuffer_Full(12,PAR_1) > 507)
For i = 1 To 500
PAR_10 = DATA_12
DAC(2,PAR_10) 'do something with Par_10
Next i
EndIf
```

**Finish:**

```
For i = 1 To RingBuffer_Full(12,PAR_1)
PAR_10 = DATA_12
Next i
```

## RingBuffer\_Clear

**RingBuffer\_Clear** initialisiert den Schreib- und den Lese-Zeiger eines Ringspeichers.

### Syntax

**RingBuffer\_Clear** (*data\_no*, *par\_x*)

### Parameter

<i>data_no</i>	Nummer (1...16) des Ringspeichers <b>DATA_x</b> .	LONG
<i>par_x</i>	Nur bei Datenübertragung <i>ADwin</i> CPU / <i>TiCo</i> : Globale Variable <b>par_x</b> (Par_1...Par_80), die eine Kopie des Schreib- oder Lesezeigers (auf der <i>ADwin</i> CPU) auf den Ringspeicher enthält.	VAR
	Alle anderen Fälle: Eine beliebige Variable, deren Wert geändert werden darf.	LONG

### Bemerkungen

Die Initialisierung eines Ringspeichers ist in 2 Fällen sinnvoll:

- Vor dem ersten Zugriff auf den Ringspeicher.  
Sie sollten dies auf jeden Fall im Abschnitt **Init**: tun, weil die Ringspeicher-Zeiger bei der Dimensionierung nicht initialisiert werden.
- Während des Programmlauf, wenn Sie alle im Feld gesammelten Daten (z. B. wegen eines Messfehlers) verwerfen wollen.

Das Initialisieren des Schreib- und des Lese-Zeigers ändert die im Ringspeicher enthaltenen Daten nicht.

Die globale Variable **par\_x** hat nur eine Bedeutung für die Datenübertragung zwischen der *ADwin* CPU und *TiCo*-Prozessor. In **par\_x** trägt die *ADwin* CPU ein, wie viele Werte sie aus dem Ringspeicher gelesen oder hinein geschrieben hat; daraus kann mit **RingBuffer\_Empty** die Anzahl der freien Elemente oder mit **RingBuffer\_Full** die Anzahl der belegten Elemente im Ringspeicher berechnet werden.

Bei einer Datenübertragung zwischen ADwin CPU und TiCo-Prozessor gilt nach der Initialisierung eines Ringspeichers mit **Ringbuffer\_Clear**:

1. Stellen Sie sicher, dass ab jetzt keine Daten mehr in den Ringspeicher geschrieben oder daraus gelesen werden.
2. Initialisieren Sie die Datenübertragung in ADbasic mit **TDrv\_Init**.
3. Jetzt können Sie wieder mit dem Ringspeicher weiter arbeiten.

### Siehe auch

RingBuffer (Deklaration), Refresh\_RingBuffer, RingBuffer\_Empty, RingBuffer\_Full, Kapitel 4.3.3 „Die Datenstruktur RingBuffer“

### Beispiel

```
REM 1007 LONG elements as write ringbuffer
Dim DATA_11[1007] As Long As Ringbuffer_For_Write

Init:
  Rem initialize read pointer of DATA_11 (using PAR_4)
  Ringbuffer_Clear(11, PAR_4)
```

## RingBuffer\_Empty

**RingBuffer\_Empty** gibt die Anzahl der freien Elemente in einem Schreib-Ringspeicher zurück.

### Syntax

```
ret_val = RingBuffer_Empty (data_no, par_x)
```

### Parameter

<b>data_no</b>	Nummer (1...16) des Ringspeichers <b>DATA_x</b> .	LONG
<b>par_x</b>	Bei Datenübertragung ADwin CPU / TiCo: Globale Variable <b>par_x</b> (Par_1...Par_80), die eine Kopie des Lesezeigers (auf der ADwin CPU) auf den Ringspeicher enthält. Bei Datenübertragung ext. Speicher / TiCo: 0.	VAR LONG
<b>ret_val</b>	Anzahl der freien Ringspeicher-Elemente.	LONG

### Bemerkungen

Initialisieren Sie den Schreibzeiger des Schreib-Ringspeichers mit **RingBuffer\_Clear**, bevor Sie das erste Mal auf den Ringspeicher zugreifen.

Wenn Sie Daten in einen Ringspeicher schreiben wollen, sollten Sie vorher mit **RingBuffer\_Empty** überprüfen, ob noch genügend Platz im Ringspeicher frei ist.

Bitte beachten Sie, dass Ringspeicher im externen Speicher immer in Schritten von 8 Elementen dimensioniert werden (see Seite 178).

Die globale Variable **par\_x** hat hier nur eine Bedeutung für die Datenübertragung zwischen der ADwin CPU (z. B. T11) und TiCo-Prozessor. In diesem Fall liest die ADwin CPU Daten aus dem Ringspeicher. Bei jedem Lesevorgang mit **Get\_TiCo\_RingBuffer** aktualisiert die ADwin CPU den Lesezeiger und kopiert den Wert in die globale Variable **par\_x** auf dem TiCo-Prozessor. Mit dem Wert in **par\_x** kann **RingBuffer\_Empty** die Anzahl der freien Elemente im Ringspeicher berechnen.

### Siehe auch

RingBuffer (Deklaration), RingBuffer\_Full, Get\_TiCo\_RingBuffer  
(ADbasic), Kapitel 4.3.3 „Die Datenstruktur RingBuffer“

### Beispiel

```
REM 1007 LONG elements as write ringbuffer
Dim DATA_11[1007] As Long As Ringbuffer_For_Write

Init:
    Ringbuffer_Clear(11, PAR_4)

Event:
    Rem read number of unused elements in DATA_11 (using PAR_4)
    PAR_1 = RingBuffer_Empty(11, PAR_4)
```

## RingBuffer\_Full

**RingBuffer\_Full** gibt die Anzahl der genutzten Elemente in einem Lese-Ringspeicher zurück.

### Syntax

```
ret_val = RingBuffer_Full (data_no, par_x)
```

### Parameter

<b>data_no</b>	Nummer (1...16) des Ringspeichers <b>DATA_x</b> .	LONG
<b>par_x</b>	Bei Datenübertragung <i>ADwin</i> CPU / <i>TiCo</i> : Globale Variable (Par_1...Par_80), die eine Kopie des Schreibzeigers (auf der <i>ADwin</i> CPU) auf den Ringspeicher enthält.	VAR LONG
	Bei Datenübertragung ext. Speicher / <i>TiCo</i> : 0.	
<b>ret_val</b>	Anzahl der belegten Ringspeicher-Elemente.	LONG

### Bemerkungen

Initialisieren Sie den Schreibzeiger des Schreib-Ringspeichers mit **RingBuffer\_Clear**, bevor Sie das erste Mal auf den Ringspeicher zugreifen.

Wenn Sie Daten aus einem Ringspeicher lesen, sollten Sie vorher mit **RingBuffer\_Full** überprüfen, ob im Ringspeicher noch Daten enthalten sind. Falls keine Daten mehr vorhanden sind, wird aus dem Ringspeicher ein undefinierter Wert gelesen.

Bitte beachten Sie, dass Ringspeicher im externen Speicher immer in Schritten von 8 Elementen dimensioniert werden (see Seite 178).

Die globale Variable **par\_x** hat hier nur eine Bedeutung für die Datenübertragung zwischen *ADwin* CPU (z.B. T11) und *TiCo*-Prozessor. In diesem Fall schreibt die *ADwin* CPU Daten in den Ringspeicher. Bei jedem Schreibvorgang mit **Set\_TiCo\_RingBuffer** aktualisiert die *ADwin* CPU den Schreibzeiger und kopiert den Wert in die globale Variable **par\_x** auf dem *TiCo*-Prozessor. Mit dem Wert in **par\_x** kann **RingBuffer\_Full** die Anzahl der belegten Elemente im Ringspeicher berechnen.



### Siehe auch

(Deklaration), RingBuffer\_Empty, Set\_TiCo\_RingBuffer (*ADbasic*),  
Kapitel 4.3.3 „Die Datenstruktur RingBuffer“

### Beispiel

```
REM 1007 LONG elements as read ringbuffer
Dim Data_12[1007] As Long As RingBuffer_For_Read

Init:
    Ringbuffer_Clear(12, PAR_3)

Event:
    Rem read number of used elements in DATA_12 (using PAR_3)
    Par_1 = RingBuffer_Full(12, PAR_3)
```

## SelectCase

Die Kontrollstruktur **SelectCase** bewirkt in Abhängigkeit von einem Wert die Ausführung eines von mehreren Anweisungsblöcken.

### Syntax

```
SelectCase var
Case const1a{,const1b, ...}
    ...                               'Anweisungsblock
CCase const2a{,const2b, ...}
    ...                               'Anweisungsblock
CaseElse
    ...                               'Anweisungsblock
EndSelect
```

### Parameter

<b>var</b>	Auszuwertendes Argument (kein Berechnungsausdruck).	LONG
<b>const1a,</b> <b>const1b,</b> <b>const2a,</b> <b>const2b</b>	Wert von <b>var</b> (0...255), bei dem der nachfolgende Anweisungsblock ausgeführt wird.	CONST LONG

### Bemerkungen

In einer Library-Funktion oder einem Library-Unterprogramm kann die Kontrollstruktur nicht verwendet werden.

Sie können mehrere **SelectCase**-Strukturen beliebig tief verschachteln; nur die Speichergröße begrenzt Sie hierbei.

Je nach Argument können Sie mit **SelectCase** verschachtelte **If**-Strukturen ersetzen und damit übersichtlicher gestalten; vor allem aber wird diese Struktur schneller ausgeführt als mehrere aufeinander folgende **If**-Strukturen.

Wenn das auszuwertende Argument mit keiner der **Case**-Konstanten übereinstimmt, wird – falls vorhanden – nur der **CaseElse**-Anwei-

sungsblock ausgeführt. Dies geschieht ebenfalls, wenn das auszuwertende Argument außerhalb des Wertebereichs der Konstanten liegt.

**CCase** steht für „Continue Case“: Wenn ein **Case**- oder **CCase**-Anweisungsblock abgearbeitet wurde, dann wird ein direkt folgender **CCase**-Anweisungsblock ebenfalls abgearbeitet.

Im Beispiel unten wird deshalb nicht nur die Anweisung **ADC (5)**, sondern auch **ADC (7)** ausgeführt. Wäre dagegen **Par\_1 = 3**, dann würde nur **ADC (7)** ausgeführt.

Wenn Sie in den Anweisungsblöcken Variablen so verändern, dass sich der Wert des Arguments ändert, wird dies erst bei der nächsten **SelectCase**-Abfrage berücksichtigt.

Die Struktur verwendet intern eine Sprungtabelle im Datenspeicher (DM), deren Speicherbedarf sich nach der größten angegebenen **Case** / **CCase**-Konstanten richtet. Um den Speicherplatz-Bedarf zu beschränken, ist der Wertebereich der Konstanten auf 0...255 eingeschränkt. Es gilt:

$$\text{Speicherbedarf in Bytes} = [ (\text{größter Konstantenwert}) + 1 ] \times 4$$

Beispielsweise wäre der Speicherbedarf bei **Case 200**:

$(200 + 1) \times 4 = 804$  Bytes; der max. Bedarf beträgt genau 1 KiB.

### Siehe auch

Do ... Until, For ... To ... {Step ... } Next, If ... Then ... {Else ... } EndIf

**Beispiel****Event:**

```

Par_1=2
SelectCase Par_1      'Par_1 auswerten
Case 0                'Ist Par_1 = 0?
    Par_10 = ADC(1)    'ADC(1) auslesen
Case 1                'Ist Par_1 = 1?
    Par_10 = ADC(3)    'ADC(3) auslesen
Case 2                'Ist Par_1 = 2?
    Par_10 = ADC(5)    'ADC(5) und auch (durch CCase)
                        ' ADC(7) auslesen
CCase 3               'Ist Par_1 = 3?
    Par_11 = ADC(7)    'ADC(7) auslesen
Case 4,5,6,7,16       'Ist Par_1 = 4, 5, 6, 7 oder 16?
    Par_2 = Digin_Word() 'digitale Eingänge einlesen
CaseElse              'Par_1: sonstige Werte
    Digout_Word(Par_10) 'Wert von Par_10 an digitale
                        'Ausgänge ausgeben
EndSelect              'Abschluss der Auswahl

```

## Shift\_Left

**Shift\_Left** verschiebt alle Bits eines Werts um eine bestimmte Stellenzahl nach links. Rechts frei werdende Bits werden zu 0 gesetzt.

### Syntax

```
ret_val = Shift_Left(val, num)
```

### Parameter

val	Argument	LONG
num	Anzahl der Stellen, um die das Argument geschoben wird (0...31).	LONG
ret_val	Argument mit verschobenen Bits oder 0 für (num < 0) und für (num > 31)	LONG

### Bemerkungen

Das Verschieben um n Stellen nach links entspricht einer Multiplikation mit  $2^n$ . Ein eventuell vorkommender Überlauf wird nicht berücksichtigt, d.h. ein gesetztes Bit ist verloren, wenn es aus dem Argument nach links „heraus geschoben“ wird.

Die Ausführungszeit ist gleich derjenigen bei einem vergleichbaren Multiplikations-Operator.

### Siehe auch

Shift\_Right

### Beispiel

```
Dim val_1, val_2 As Long
```

#### Event:

```
val_1 = 1024
```

```
val_2 = Shift_Left(val_1, 2) 'Ergebnis: val_2=4096
```

## Shift\_Right

**Shift\_Right** verschiebt alle Bits eines Werts um eine bestimmte Stellenzahl nach rechts. Links frei werdende Bits werden zu 0 gesetzt.

### Syntax

```
ret_val = Shift_Right(val, num)
```

### Parameter

<code>val</code>	Argument	LONG
<code>num</code>	Anzahl der Stellen, um die das Argument geschoben wird (0...31).	LONG
<code>ret_val</code>	Argument mit verschobenen Bits oder 0 für ( <code>num &lt; 0</code> ) und für ( <code>num &gt; 31</code> )	LONG

### Bemerkungen

Falls das Argument eine positive Zahl ist, entspricht das Verschieben um  $n$  Stellen nach rechts einer Division durch  $2^n$ . Ein eventuell vorkommender Divisions-Rest wird nicht berücksichtigt, d.h. ein gesetztes Bit, das aus dem Argument nach rechts „heraus geschoben“ wird, ist verloren.

Die Ausführungszeit ist geringer als bei einem vergleichbaren Divisions-Operator, d.h. `val_2 = Shift_Right(val_1, 3)` ist schneller als `val_2 = val_1 / 8`.

### Siehe auch

Shift\_Left

### Beispiel

```
Dim val_1, val_2 As Long
```

**Event:**

```
val_1 = 1024
```

```
val_2 = Shift_Right(val_1, 3) 'Ergebnis: val_2=128
```

## Sleep

**Sleep** lässt den Prozessor für eine bestimmte Zeit warten.

### Syntax

**Sleep**(val)

### Parameter

val                      Anzahl ( $\geq 1$ ) der zu wartenden Zeiteinheiten in 100ns. LONG

### Bemerkungen

Da der Befehl **Sleep** als Zählschleife ausgeführt wird, kann er bei einem hochpriorien Prozess nicht unterbrochen werden.

Verwenden Sie nach Möglichkeit eine Konstante als Argument. Wenn das Argument **val** eine Berechnung erfordert, benötigt dies eine bestimmte Zeitspanne zusätzlich; diese ist jeweils konstant und beträgt einige wenige Taktzyklen.

Folgende Fälle erfordern eine Berechnung:

- Das Argument ist ein Berechnungsausdruck mit Variablen oder Feldelementen.
- Die Variable im Argument ist für den Speicherbereich **DRAM\_Extern** deklariert.
- Das Argument ist ein Feld.

### Siehe auch

NOP, NOPs

### Beispiel

```
Event:
Set_Mux(0)           'Multiplexer setzen
Sleep(25)            '2.5 µs (=25*100ns) warten
                     '= Einschwingzeit des MUX
Start_Conv(1)        'Konvertierung starten
...
```

## Sub ... EndSub

`Sub...EndSub` definiert ein Unterprogramm-Makro mit Übergabeparametern.

### Syntax

```
Sub macro_name ({val_1, val_2, ...})
    {Dim var As <VAR_TYPE>}
    ...
    'Anweisungsblock
EndSub
```

### Parameter

`macro_name` Name des Unterprogramms

`val_1, val_2` Namen der Übergabeparameter;  
für Felder ist die Syntax mit Dimensionsklammern erforderlich: `array[]` oder `Data_n[]`.

FLOAT

LONG

### Bemerkungen

Allgemeine Informationen über Makros finden Sie in Kapitel 4.5.1 auf Seite 95.

Diese Anweisung definiert ein Unterprogramm-Makro, d.h. der vollständige Anweisungsblock zwischen `Sub` und `EndSub` wird an der aufrufenden Stelle eingefügt.

Unterprogramm-Makros erhöhen die Übersichtlichkeit Ihres Quelltextes. Beachten Sie aber, dass auch jeder Aufruf des Unterprogramms die kompilierte Datei vergrößert.

Sie können Unterprogramme an 3 Stellen einfügen:

1. Vor dem Abschnitt `Init:`.
2. Nach dem Abschnitt `Finish:`.
3. In einer separaten Datei, die Sie mit `#Include` einbinden (aber nur an einer der Stellen, die unter 1. und 2. angegeben sind).

Beachten Sie bitte, dass Sie in Unterprogrammen:

- keine Prozess-Abschnitte wie `Init:`, `Event:`, oder `Finish:` definieren dürfen.
- am Anfang lokale Variablen definieren können, die nur innerhalb des Unterprogramms und für die Dauer der Abarbeitung



verfügbar sind.

Dies gilt auch, wenn die Variablen den gleichen Namen haben wie Variablen außerhalb des Unterprogramms.

Bei Berechnungsausdrücken in einem Unterprogramm sollten die Übergabeparameter in Klammern stehen. Auf diese Weise vermeiden Sie Probleme mit der Rangfolge von Operatoren (z.B. Punkt- vor Strich-Rechnung).

Ein Unterprogramm wird mit seinem Namen und allen definierten Argumenten aufgerufen. Als Argument ist jeder Berechnungsausdruck (auch Felder) zulässig, solange er den passenden Datentyp hat. Wenn Sie keine Argumente definieren, müssen Sie dennoch beim Aufruf des Unterprogramms die Leerklammern verwenden: `name()`.

Wenn ein Feld (kein Feldelement) als Übergabeparameter eines Unterprogramms verwendet wird, ist die Syntax für Aufruf und Definition unterschiedlich:

- Unterprogramm-Aufruf *ohne* Dimensions-Klammern:  
`subname(array_pass)`
- Unterprogramm-Definition *mit* Dimensions-Klammern:  
`Sub subname(array_def[]) ...`

Werte werden an Feldelemente (eines Felds als Übergabeparameter) zugewiesen wie gewöhnlich:

```
array_def[2] = value
```

Wenn Sie einem Übergabeparameter `x` im Unterprogramm einen Wert zuweisen, darf beim Unterprogramm-Aufruf für `x` keine Konstante angegeben werden, sondern nur eine Variable oder ein einzelnes Feldelement. Auf diese Weise können Übergabeparameter auch einen Rückgabewert enthalten.

## Siehe auch

#Include, Function ... EndFunction

## Beispiel

- / -

## XOr

Der Operator **XOr** (Exklusiv-Oder) verknüpft zwei ganzzahlige Werte bitweise.

### Syntax

```
... val_1 XOr val_2 ...
```

### Parameter

**val\_1, val\_2** Ganzzahliger Wert

LONG
------

### Siehe auch

And, Not, Or

### Beispiel

```
Dim value As Long
```

```
Event:
```

```
value = 0100b XOr 0110b
```

```
Rem Ergebnis: value = (4 XOr 6) = 0010b = 2
```

## 7.3 Gold II: TiCo-Prozessor

Mit den folgenden *ADbasic*-Befehlen greift die *ADwin* CPU (Prozessor T11) auf den *TiCo*-Prozessor zu:

Datenzugriff Initialisieren	TDrv_Init
Globale Variablen lesen und schreiben	Get_Par, Get_Par_Block Set_Par, Set_Par_Block
Globale Felder lesen und schreiben	GetData_Long, SetData_Long
Ringspeicher lesen, schreiben und Status abfragen	Get_TiCo_RingBuffer, RingBuffer_Empty Set_TiCo_RingBuffer, RingBuffer_Full
Processdelay setzen und abfragen	TiCo_Get_Processdelay, TiCo_Set_Processdelay
Prozesse starten und stoppen, Prozessor steuern	TiCo_Start_Process, TiCo_Stop_Process TiCo_Stop, TiCo_Start, TiCo_Reset, TiCo_Reset_Mode
Systeminformationen abfragen	Get_TiCo_Status, Process_Status, Workload
Binärdatei übertragen	TiCo_Flash, TiCo_Load

Beachten Sie bitte: Bei fast allen Befehlen muss zuerst die Datenübertragung mit **TDrv\_Init** initialisiert werden.

## Get\_Par

**Get\_Par** gibt den Wert einer globalen Variablen **Par\_x** von einem *TiCo*-Prozessor zurück.

### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val = Get_Par(tico_no, Par_no)
```

### Parameter

<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
<code>Par_no</code>	Nummer (1...80) der globalen Variablen.	LONG
<code>ret_val</code>	Wert ( $-2^{31} \dots +2^{31}-1$ ) der globalen Variablen.	LONG

### Bemerkungen

Mehrere Werte werden mit **Get\_Par\_Block** schneller gelesen.

### Siehe auch

[Get\\_Par\\_Block](#), [GetData\\_Long](#), [Set\\_Par](#), [Set\\_Par\\_Block](#), [SetData\\_Long](#), [TDrv\\_Init](#)

### Gültig für

Gold II

### Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Init:
    TDrv_Init(tico_no,tset)

Event:
    REM read Par_1 from TiCo and write value to Par_2 of ADwin
    REM CPU
    Par_2 = Get_Par(tico_no,1)
```

## Get\_Par\_Block

**Get\_Par\_Block** liest eine Anzahl von globalen Variablen **Par\_x** des *TiCo*-Prozessors und schreibt die Werte in ein Feld.

### Syntax

```
#Include ADwinGoldII.inc

Get_Par_Block(tico_no, dest_array[],
              dest_array_idx, Par_no, Par_count)
```

### Parameter

<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
<code>dest_array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>dest_array_idx</code>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte abgelegt werden.	LONG
<code>Par_no</code>	Index (1...80) der ersten globalen Variablen, die gelesen wird.	LONG
<code>Par_count</code>	Anzahl (1...80) der zu lesenden Variablen.	LONG

### Bemerkungen

- / -

### Siehe auch

[Get\\_Par](#), [GetData\\_Long](#), [Set\\_Par](#), [Set\\_Par\\_Block](#), [SetData\\_Long](#), [TDrv\\_Init](#)

### Gültig für

Gold II

### Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim Data_1[80] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no,tset)

Event:
    REM read Par_1...Par_80 from TiCo and write values to
    REM Data_1[1]...Data_1[80] of ADwin CPU
    Get_Par_Block(tico_no,Data_1,1,1,80)
```

## Get\_TiCo\_RingBuffer

**Get\_TiCo\_RingBuffer** liest Werte aus einem Ringspeicher von einem *TiCo*-Prozessor und schreibt die Werte in ein Feld der *ADwin* CPU.

### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val =  
    Get_TiCo_RingBuffer(tdrv_datatable[],  
        src_array_no, dest_array[],
```



```
dest_array_idx,  
maxcount, flowrate, tico_par, struct)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>[]</code>		
<code>src_</code>	Nummer (1...16) des Schreib-Ringspeichers	FLOAT
<code>array_no</code>	<b>Data_n</b> auf dem <i>TiCo</i> -Prozessor.	
<code>dest_</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>array[]</code>		LONG
<code>dest_</code>	Index (1...n) des ersten Elements im Feld <code>dest_</code>	LONG
<code>array_idx</code>	<code>array[]</code> , in das geschrieben wird.	
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Lesezeigers zum <i>TiCo</i> -Prozessor. 1...80: Nummer (1...80) der globalen Variablen <b>Par_n</b> des <i>TiCo</i> -Prozessors, in die der aktuelle Lesezeigerwert geschrieben wird. 0: Der Lesezeiger wird nicht übertragen.	LONG
<code>struct</code>	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten lesen. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von <code>struct</code> sein.	LONG

`ret_val`

Erfolgsstatus des Befehls:

LONG

-1: Fehler, der Schreib-Ringspeicher des *TiCo* ist nicht korrekt deklariert.

≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.

### Bemerkungen

Der Befehl liest nicht mehr als `maxcount` Daten. Wenn weniger Daten im Ringspeicher enthalten sind, werden alle vorhandenen Daten übertragen.

Beim Lesen von einem Ringspeicher speichert `Get_TiCo_Ringbuffer` die letzte Leseposition, den Lesezeiger, in dem Feld `tdrv_datatable[]`. Wenn der Befehl `Ringbuffer_Empty` des *TiCo*-Prozessors korrekt arbeiten soll, muss in `tico_par` die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des *TiCo*-Prozessors übertragen.

### Siehe auch

[Set\\_TiCo\\_RingBuffer](#), [TDrv\\_Init](#)

### Gültig für

Gold II

### Beispiel

- / -

### Get\_TiCo\_Status

**Get\_TiCo\_Status** gibt zurück, ob der *TiCo*-Prozessor aktiv ist.

#### Syntax

```
#Include ADwinGoldII.inc  
ret_val = Get_TiCo_Status ()
```

#### Parameter

<b>ret_val</b>	Status des <i>TiCo</i> -Prozessors:	LONG
	0: Prozessor ist gestoppt.	
	1: Prozessor läuft.	
	-3: Fehler, kein <i>TiCo</i> -Prozessor vorhanden.	

#### Bemerkungen

- / -

#### Siehe auch

[Process\\_Status](#), [Workload](#)

#### Gültig für

Gold II

#### Beispiel

- / -

## GetData\_Long

**GetData\_Long** liest Werte aus einem globalen Feld von einem *TiCo*-Prozessor und schreibt die Werte in ein Feld.

## Syntax

```
#Include ADwinGoldII.inc

GetData_Long(tdrv_datatable[], src_array_no,
             src_array_idx, count, dest_array[],
             dest_array_idx, flowrate)
```

## Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>src_</code>		
<code>array_no</code>	Nummer (1...16) des globalen Felds <b>Data_x</b> auf dem <i>TiCo</i> -Prozessor.	LONG
<code>src_</code>		
<code>array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>src_array_no</code> gelesen wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>dest_</code>		
<code>array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>dest_</code>		
<code>array_idx</code>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte im Zielfeld abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriorie Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

## Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `src_array_no` auf dem *TiCo*-Prozessor deklariert ist und
- das Feld `dest_array` mit mindestens `count` Elementen deklariert ist.

**Siehe auch**

[Get\\_Par](#), [Set\\_Par](#), [SetData\\_Long](#), [TDrv\\_Init](#)

**Gültig für**

Gold II

**Beispiel**

```
#Include ADwinGoldII.INC
#Define tico_no 1 'TiCo no.
Dim tset_41[150] As Long 'settings array for data
                          'transfer

Dim Data_4[200] As Long

Init:
  REM initialize data transfer ADwin CPU <-> TiCo
  TDrv_Init(tico_no,tset_41)

Event:
  REM read 120 values from TiCo Data_2 (starting from Data_
  REM 2[20])
  REM into Data_4 of ADwin CPU (starting from index 5).
  REM flowrate is high
  GetData_Long(tset_41,2,20,120,Data_4,5,3)
```

## Process\_Status

**Process\_Status** gibt den Status eines Prozesses auf einem *TiCo*-Prozessor zurück.

### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val = Process_Status(tico_no, process_no)
```

### Parameter

tico_no	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
process_no	Nummer (1) des <i>TiCo</i> -Prozesses.	LONG
ret_val	Status des Prozesses: ≠1: Prozess läuft. 0: Prozess läuft nicht, d.h. er ist nicht geladen, nicht gestartet oder gestoppt.	LONG

### Bemerkungen

- / -

### Siehe auch

[Get\\_TiCo\\_Status](#), [TiCo\\_Get\\_Processdelay](#), [TiCo\\_Set\\_Processdelay](#), [TiCo\\_Start\\_Process](#), [TiCo\\_Stop\\_Process](#), [Workload](#)

### Gültig für

Gold II

**Beispiel**

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim Data_4[200] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no,tset)

Event:
    REM read status of TiCo process 1
    Par_1 = Process_Status(tico_no,1)
    REM if process is running, read TiCo Par_5
    If (Par_1 = 1) Then
        Par_2 = Get_Par(tico_no,5)
    EndIf
```



## RingBuffer\_Empty

**RingBuffer\_Empty** gibt die Anzahl der freien Elemente in einem Schreib-Ringspeicher eines *TiCo*-Prozessors zurück.

### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val = RingBuffer_Empty(tdrv_datatable[],  
                             Data_no)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>[]</code>		
<code>Data_no</code>	Nummer (1...16) des Schreib-Ringspeichers <code>Data_n</code> auf dem <i>TiCo</i> -Prozessor.	LONG
<code>ret_val</code>	Anzahl der freien Elemente im Schreib-Ringspeicher.	LONG

### Bemerkungen

Bei **Get\_TiCo\_RingBuffer** ist keine vorherige Abfrage mit **RingBuffer\_Empty** erforderlich.

### Siehe auch

[Get\\_TiCo\\_RingBuffer](#), [RingBuffer\\_Full](#), [TDrv\\_Init](#)

### Gültig für

Gold II

### Beispiel

- / -

## RingBuffer\_Full

**RingBuffer\_Full** gibt die Anzahl der genutzten Elemente in einem Lese-Ringspeicher eines *TiCo*-Prozessors zurück.

### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val = RingBuffer_Full(tdrv_datatable[],  
                          Data_no)
```

### Parameter

<code>tdrv_datatable</code> []	Feld, das Einstellungen für die Datenübertragung enthält, u.a. <i>TiCo</i> -Nummer.	ARRAY LONG
<code>Data_no</code>	Nummer (1...16) des Lese-Ringspeichers <code>Data_n</code> auf dem <i>TiCo</i> -Prozessor.	LONG
<code>ret_val</code>	Anzahl (0...n) der genutzten Elemente im Lese-Ringspeicher.	LONG

### Bemerkungen

Bei **Set\_TiCo\_RingBuffer** ist keine vorherige Abfrage mit **RingBuffer\_FULL** erforderlich.

### Siehe auch

[Set\\_TiCo\\_RingBuffer](#), [RingBuffer\\_Empty](#), [TDrv\\_Init](#)

### Gültig für

Gold II

### Beispiel

- / -

## Set\_Par

**Set\_Par** setzt den Wert einer globalen Variablen **Par\_x** auf einem *TiCo*-Prozessor.

### Syntax

```
#Include ADwinGoldII.inc  
Set_Par(tico_no, Par_no, value)
```

### Parameter

tico_no	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
Par_no	Nummer (1...80) der globalen Variablen.	LONG
value	Wert ( $-2^{31} \dots +2^{31}-1$ ) der globalen Variablen.	LONG

### Bemerkungen

**Set\_Par** setzt den Wert der globalen Variablen unabhängig davon, ob gerade ein *TiCo*-Prozess läuft. Da *ADwin* CPU- und *TiCo*-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den *TiCo*-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und *TiCo* mit Hilfe eines Software-Handshakes.

### Siehe auch

[Get\\_Par](#), [Get\\_Par\\_Block](#), [GetData\\_Long](#), [Set\\_Par\\_Block](#), [SetData\\_Long](#), [TDrv\\_Init](#)

### Gültig für

Gold II

**Beispiel**

```
#Include ADwinGoldII.Inc
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  TDrv_Init(tico_no,tset)
  Par_5=200

Event:
  REM write value of Par_5 (ADwin CPU) to TiCo Par_2
  Set_Par(tico_no,2,Par_5)
```

## Set\_Par\_Block

**Set\_Par\_Block** schreibt Werte aus einem Feld in eine Anzahl von globalen Variablen **Par\_x** des *TiCo*-Prozessors.

### Syntax

```
#Include ADwinGoldII.inc

Set_Par_Block(tico_no, src_array[],
              src_array_idx,
              Par_no, Par_count)
```

### Parameter

<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
<code>src_array[]</code>	Quellfeld, aus dem Daten übertragen werden.	ARRAY
<code>src_array_idx</code>	Indes des Feldelements, ab dem Werte aus dem Quellfeld gelesen werden.	LONG
<code>Par_no</code>	Index (1...80) der ersten globalen <i>TiCo</i> -Variablen, die beschrieben wird.	LONG
<code>Par_count</code>	Anzahl der zu übertragenden Werte.	LONG

### Bemerkungen

**Set\_Par\_Block** setzt die Wert der globalen Variablen unabhängig davon, ob gerade ein *TiCo*-Prozess läuft. Da *ADwin* CPU- und *TiCo*-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den *TiCo*-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und *TiCo* mit Hilfe eines Software-Handshakes.

### Siehe auch

[Get\\_Par](#), [Get\\_Par\\_Block](#), [GetData\\_Long](#), [Set\\_ParSetData\\_Long](#), [TDrv\\_Init](#)

**Gültig für**

Gold II

**Beispiel**

siehe Datei C:\ADwin\ADbasic\Examples\Set\_Par\_Block.bas.

```
#Include ADwinGoldII.Inc
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Dim Data_1[40] As Long
Dim i As Long

Init:
  TDrv_Init(tico_no,tset)
  For i = 1 To 40
    Data_1[i] = i*10
  Next

Event:
  REM write 40 values from Data_1 (starting from Data_1[1])
  REM into Par_1...Par_40 (TiCo)
  Set_Par_Block(tico_no,Data_1,1,1,40)
```

### Set\_TiCo\_RingBuffer

**Set\_TiCo\_RingBuffer** schreibt Werte aus einem Feld der *ADwin* CPU in einen Ringspeicher auf einem *TiCo*-Prozessor.

#### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val =  
    Set_TiCo_RingBuffer(tdrv_datatable[],
```

```
dest_array_no, src_array[], src_array_idx,
maxcount, flowrate, tico_par, struct)
```

## Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>dest_</code>		
<code>array_no</code>	Nummer (1...16) des Lese-Ringspeichers <b>Data_</b> <b>n</b> auf dem <i>TiCo</i> -Prozessor.	FLOAT
<code>src_</code>		
<code>array[]</code>	Quellfeld, aus dem die Werte übertragen werden.	ARRAY
<code>src_</code>		
<code>array_idx</code>	Index (1...n) des ersten Elements im Feld <code>src_</code> <code>array[]</code> , das gelesen wird.	LONG
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Ken- nwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Schreibzeigers zum <i>TiCo</i> -Prozessor. 1...80: Nummer (1...80) der globalen Variablen <b>par_n</b> des <i>TiCo</i> -Prozessors, in die der aktu- elle Schreibzeiger geschrieben wird. 0: Der Schreibzeiger wird nicht übertragen.	LONG
<code>struct</code>	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten schreiben. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von <code>struct</code> sein.	LONG



`ret_val`

Erfolgsstatus des Befehls:

LONG

-1: Fehler, der Lese-Ringspeicher des *TiCo* ist nicht korrekt deklariert.

≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.

## Bemerkungen

Der Befehl schreibt nicht mehr als `maxcount` Daten. Wenn weniger freie Elemente im Ringspeicher vorhanden sind, werden nur die freien Elemente im Ringspeicher beschrieben.

Beim Schreiben in einen Ringspeicher speichert **Set\_TICO\_Ringbuffer** die letzte Schreibposition, den Schreibzeiger, in dem Feld `tdrv_datatable[]`. Wenn der Befehl **Ringbuffer\_Full** des *TiCo*-Prozessors korrekt arbeiten soll, muss in `tico_par` die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des *TiCo*-Prozessors übertragen.

## Siehe auch

[Get\\_TiCo\\_RingBuffer](#), [TDrv\\_Init](#)

## Gültig für

Gold II

## Beispiel

- / -

## SetData\_Long

**SetData\_Long** liest Werte aus einem Feld und schreibt sie in ein globales Feld eines *TiCo*-Prozessors.

## Syntax

```
#Include ADwinGoldII.inc

SetData_Long(tdrv_datatable[], dest_array_no,
             dest_array_idx, count, src_array[],
             src_array_idx, flowrate)
```

## Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>dest_</code> <code>array_no</code>	Nummer (1...16) des globalen Felds <b>Data_x</b> auf dem <i>TiCo</i> -Prozessor.	LONG
<code>dest_</code> <code>array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>dest_array_no</code> beschrieben wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>src_</code> <code>array[]</code>	Quellfeld, aus dem die Werte gelesen werden.	ARRAY
<code>src_</code> <code>array_idx</code>	Quell-Startindex (1...n): Feldelement, ab dem die Werte gelesen werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

## Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `dest_array_no` auf dem *TiCo*-Prozessor deklariert ist und
- das Feld mit mindestens `count` Elementen deklariert ist.

**Siehe auch**

[Get\\_Par](#), [GetData\\_Long](#), [Set\\_Par](#), [TDrv\\_Init](#)

**Gültig für**

Gold II

**Beispiel**

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Dim Data_1[150] As Long
Dim tset_41[150] As Long
Dim i As Long
```

**Init:**

```
REM initialize data transfer ADwin CPU <-> TiCo
TDrv_Init(tico_no,tset_41)
For i = 1 To 80
    Data_1[i] = i
Next
```

**Event:**

```
REM read 120 values from TiCo Data_2 (starting from Data_
REM 2[1])
REM into Data_1 of ADwin CPU (starting from Data_1[5]).
REM flowrate is high
SetData_Long(tset_41,2,1,120,Data_1,5,3)
```

## TDrv\_Init

**TDrv\_Init** initialisiert die Datenübertragung zwischen *ADwin* CPU und einem bestimmten *TiCo*-Prozessor.

### Syntax

```
#Include ADwinGoldII.inc  
  
REM define settings array for TiCo x  
DIM tdrv_datatable[150] AS LONG  
  
TDrv_Init(tico_no, tdrv_datatable[])
```

### Parameter

<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung aufnimmt.	ARRAY
<code>[]</code>		LONG

### Bemerkungen

Der Befehl muss vor der Datenübertragung zwischen *ADwin* CPU und *TiCo* ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen.

Fast alle Befehle, die auf den *TiCo* zugreifen, benötigen eine Initialisierung der Datenübertragung.

Die Initialisierung muss für jeden *TiCo*-Prozessor separat durchgeführt werden. Für jede Datenübertragung mit einem *TiCo*-Prozessor muss ein Feld `tdrv_datatable[]` mit 150 Elementen angelegt werden.

Das Feld `tdrv_datatable[]` wird von den Befehlen **GetData\_Long**, **SetData\_Long** und **Workload** verwendet.

**Siehe auch**

Get\_Par, Get\_Par\_Block, TiCo\_Get\_Processdelay, GetData\_Long, Set\_Par, Set\_Par\_Block, TiCo\_Set\_Processdelay, SetData\_Long

**Gültig für**

Gold II

**Beispiel**

```
#Include ADwinGoldII.inc
#Define tico_a 1           'TiCo no.
#Define tico_b 1           'TiCo no.
Dim Data_4[200] As Long
Dim Data_5[1000] As Long
Dim tset_41[150] As Long
Dim tset_71[150] As Long
```

**Init:**

```
REM initialize 2 data transfers ADwin CPU <-> TiCo
TDrv_Init(tico_a,tset_41)
TDrv_Init(tico_b,tset_71)
```

**Event:**

```
REM read 120 values from TiCo Data_2 (starting from
REM Data_2[20]) into ADwin CPU Data_4 (starting from
REM index 5).
REM flowrate is high
GetData_Long(tset_41,2,20,120,Data_4,5,3)
REM read 800 values from TiCo Data_7 (starting from
REM Data_7[9]) into Data_5 of ADwin CPU (starting from
REM index 1).
REM Flowrate is high
GetData_Long(tset_71,7,9,800,Data_5,1,3)
```

## TiCo\_Flash

**TiCo\_Flash** überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Flash-Speicher eines *TiCo*-Prozessors.

### Syntax

```
#INCLUDE ADwinGoldII.inc  
  
ret_val = TiCo_Flash(tico_no,array[])
```

### Parameter

tico_no	Nummer (1...2) des TiCo-Prozessors.	LONG
array[]	Feld, in dem die <i>TiCoBasic</i> -Binärdatei enthalten ist.	ARRAY LONG
ret_val	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Externer Speicher ist zu klein. 7: Fehler: Flash-Speicher (EEPROM) ist zu klein. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

### Bemerkungen

Verwenden Sie **TiCo\_Flash** nur in niederprioren Prozessen.

Der Befehl **TiCo\_Flash** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den Flash-Speicher

eines *TiCo*-Prozessors zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit `Build ▶ Make Bin File`.
- Übertragen Sie die Binärdatei in ein globales Feld der *ADwin* CPU. Geeignet ist die Funktion `Data2File`, die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld `array[]` mit dem Befehl `TiCo_Flash` in den Flash-Speicher .

Wenn das Feld `array[]` keine *TiCoBasic*-Binärdatei enthält, ist der Rückgabewert des Befehls ungültig.

### Siehe auch

[TiCo\\_Load](#)

### Gültig für

Gold II

### Beispiel

- / -



## TiCo\_Get\_Processdelay

**TiCo\_Get\_Processdelay** gibt das Processdelay (die Zykluszeit) eines Prozesses auf einem *TiCo*-Prozessor zurück.

### Syntax

```
#Include ADwinGoldII.inc  
  
ret_val =  
    TiCo_Get_Processdelay(tdrv_datatable[],  
    process_no)
```

### Parameter

<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung enthält, u.a. <i>TiCo</i> -Nummer.	ARRAY
<code>process_no</code>	Nummer (1) des <i>TiCo</i> -Prozesses.	LONG
<code>ret_val</code>	Aktuell eingestellte Zykluszeit ( $-2^{31} \dots +2^{31}-1$ ) des <i>TiCo</i> -Prozessors in Taktzyklen. Ein Taktzyklus dauert 20ns.	LONG

### Bemerkungen

Bei einem zeitgesteuerten Prozess wird der Abschnitt **Event:** vom internen Zähler zyklisch und in festen Zeitabständen aufgerufen. Der Zeitabstand zwischen zwei Aufrufen, Zykluszeit oder Processdelay genannt, wird in Zähler-Taktzyklen gezählt.

### Siehe auch

[Process\\_Status](#), [TiCo\\_Set\\_Processdelay](#), [TDrv\\_Init](#)

### Gültig für

Gold II

**Beispiel**

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    TDrv_Init(tico_no, tset)

Event:
    REM read processdelay of process 1 into Par_1
    Par_1 = TiCo_Get_Processdelay(tset, 1)
```

## TiCo\_Load

**TiCo\_Load** überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Speicher eines *TiCo*-Prozessors.

### Syntax

```
#INCLUDE ADwinGoldII.inc

ret_val = TiCo_Load(tico_no, array[])
```

### Parameter

<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors.	LONG
<code>array[]</code>	Feld, in dem die <i>TiCoBasic</i> -Binärdatei enthalten ist.	ARRAY LONG
<code>ret_val</code>	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 1: Fehler: Ungültige Prozessornummer. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Zugriff auf externen Speicher ist nicht möglich. 10: Fehler: Der benötigte externe SRAM-Speicher ist zu klein. 11: Fehler: Das Feld enthält keine gültige <i>TiCo-Basic</i> -Binärdatei. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

### Bemerkungen

Verwenden Sie **TiCo\_Flash** nur in niederprioren Prozessen.

Der Befehl **TiCo\_Load** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den *TiCo*-Prozessor zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit **Build ▶ Make Bin File**.
- Übertragen Sie die Binärdatei in ein globales Feld des *ADwin*-Prozessors. Geeignet ist die Funktion **Data2File**, die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld **array[]** mit dem Befehl **TiCo\_Load** auf den *TiCo*-Prozessor.

Wenn das Feld **array[]** keine *TiCoBasic*-Binärdatei enthält, ist der Rückgabewert des Befehls ungültig.

### Siehe auch

[TiCo\\_Flash](#)

### Gültig für

Gold II

### Beispiel

- / -

### TiCo\_Reset

**TiCo\_Reset** stoppt die *TiCo*-Prozessoren und startet sie anschließend wieder.

#### Syntax

```
#Include ADwinGoldII.inc  
TiCo_Reset ( )
```

#### Parameter

- / -

#### Bemerkungen

Das Stoppen beendet auf den angesprochenen *TiCo*-Prozessoren sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

Die *TiCo*-Prozessoren werden gleichzeitig gestartet. Der Befehl eignet sich daher, um alle angesprochenen *TiCo*-Prozessoren zu synchronisieren.

#### Siehe auch

[TiCo\\_Reset\\_Mode](#), [TiCo\\_Stop](#), [TiCo\\_Start](#)

#### Gültig für

Gold II

#### Beispiel

- / -

## **TiCo\_Reset\_Mode**

### TiCo\_Reset\_Mode

**TiCo\_Reset\_Mode** stellt ein, ob beim *TiCo*-Prozessor ein Reset durchgeführt wird, wenn die *ADwin* CPU (T11) gebootet wird.

#### Syntax

```
#Include ADwinGoldII.inc  
TiCo_Reset_Mode (mode)
```

#### Parameter

<b>mode</b>	Gewünschter Reset-Modus beim Booten des T11. 0: Betriebszustand des <i>TiCo</i> -Prozessors bleibt unverändert. Default. 1: Der <i>TiCo</i> -Prozessor wird gestoppt und neu gestartet.
-------------	---

LONG
------

#### Bemerkungen

Der Betriebsmodus **mode**=1 hat nur eine Bedeutung, wenn der *TiCo*-Prozessor bereits läuft.

#### Siehe auch

[TiCo\\_Reset](#), [TiCo\\_Stop](#), [TiCo\\_Start](#)

#### Gültig für

Gold II

#### Beispiel

```
#Include ADwinGoldII.inc  
INIT:  
  TiCo_Reset_Mode (1)      'reset TiCo processor with T11  
                             'boot
```

## TiCo\_Set\_Processdelay

**TiCo\_Set\_Processdelay** setzt das Processdelay (die Zykluszeit) eines Prozesses auf einem *TiCo*-Prozessor.

### Syntax

```
#Include ADwinGoldII.inc  
  
TiCo_Set_Processdelay (tdrv_  
    datatable[], process_no,  
    value)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>[]</code>		
<code>process_</code>	Nummer (1) des <i>TiCo</i> -Prozesses.	LONG
<code>no</code>		
<code>value</code>	Einzustellender Wert für das Processdelay.	LONG

### Bemerkungen

- / -

### Siehe auch

[TiCo\\_Get\\_Processdelay](#), [Process\\_Status](#), [TDrv\\_Init](#)

### Gültig für

Gold II



### Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  TDrv_Init(tico_no, tset)
  Processdelay = 6000       'set cycle time

Event:
  REM set TiCo processdelay to run with same cycle time as
  REM the
  REM T12 process
  TiCo_Set_Processdelay(tset, 1, Processdelay/6)
```

## TiCo\_Start

**TiCo\_Start** startet alle *TiCo*-Prozessoren.

### Syntax

```
#Include ADwinGoldII.inc  
TiCo_Start()
```

### Parameter

- / -

### Bemerkungen

Die *TiCo*-Prozessoren werden gleichzeitig gestartet. Der Befehl eignet sich daher, um alle angesprochenen *TiCo*-Prozessoren zu synchronisieren.

### Siehe auch

[TiCo\\_Stop](#), [TiCo\\_Reset](#)

### Gültig für

Gold II

### Beispiel

- / -

## TiCo\_Start\_Process

**TiCo\_Start\_Process** startet einen Prozess auf einem *TiCo*-Prozessor.

### Syntax

```
#Include ADwinGoldII.inc  
  
TiCo_Start_Process (tdrv_datatable[],  
                    process_no)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>[]</code>		
<code>process_</code>	Nummer (1) des <i>TiCo</i> -Prozesses.	LONG
<code>no</code>		

### Bemerkungen

Der Prozess muss bereits auf dem *TiCo*-Prozessor vorhanden sein.

### Siehe auch

[TiCo\\_Get\\_Processdelay](#), [Process\\_Status](#), [TiCo\\_Set\\_Processdelay](#), [TiCo\\_Start\\_Process](#), [TiCo\\_Stop\\_Process](#), [Workload](#)

### Gültig für

Gold II

**Beispiel**

```
#Include ADwinGoldII.INC
Dim tset[150] As Long      'settings array for data
                             'transfer
#Define tico_no 1          'TiCo no.

Init:
  Par_1 = 0
  Processdelay = 100000
  REM start TiCo process in parallel to ADwin CPU
  REM process
  TDrv_Init(tico_no,tset)
  TiCo_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process in parallel to ADwin CPU
  REM process
  TiCo_Stop_Process(tset,1)
```

### TiCo\_Stop

**TiCo\_Stop** stoppt alle *TiCo*-Prozessoren.

#### Syntax

```
#Include ADwinGoldII.inc  
TiCo_Stop ()
```

#### Parameter

- / -

#### Bemerkungen

Das Stoppen beendet auf den angesprochenen *TiCo*-Prozessoren sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

#### Siehe auch

[TiCo\\_Start](#), [TiCo\\_Reset](#)

#### Gültig für

Gold II

#### Beispiel

- / -

## TiCo\_Stop\_Process

**TiCo\_Stop\_Process** stoppt einen Prozess auf einem *TiCo*-Prozessor.

### Syntax

```
#Include ADwinGoldII.inc  
  
TiCo_Stop_Process (tdrv_datatable[],  
                  process_no)
```

### Parameter

<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung enthält, u.a. <i>TiCo</i> -Nummer.	ARRAY
<code>process_no</code>	Nummer (1) des <i>TiCo</i> -Prozesses.	LONG

### Bemerkungen

Der Prozess muss auf dem *TiCo*-Prozessor vorhanden sein.

### Siehe auch

[TiCo\\_Get\\_Processdelay](#), [Process\\_Status](#), [TiCo\\_Set\\_Processdelay](#), [TiCo\\_Start\\_Process](#), [Workload](#)

### Gültig für

Gold II

### Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  Par_1 = 0
  Processdelay = 100000
  REM start TiCo process in parallel to ADwin CPU process
  TDrv_Init(tico_no,tset)
  TiCo_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process
  TiCo_Stop_Process(tset,1)
```

## Workload

**Workload** gibt die Auslastung eines *TiCo*-Prozessors zurück.

### Syntax

```
#Include ADwinGoldII.inc
ret_val = Workload(tdrv_datatable[])
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. <i>TiCo</i> -Nummer.	LONG
<code>ret_val</code>	Prozessor-Auslastung in Prozent (0.0 ... 100.0) oder Fehlerwert: <0: <i>TiCo</i> -Prozessor ist gestoppt oder kein <i>TiCo</i> -Prozessor vorhanden.	FLOAT

### Bemerkungen

Der Rückgabewert ist die mittlere Prozessorauslastung in der Zeit zwischen dem vorherigen und dem aktuellen Aufruf von **Workload**. Beim ersten Aufruf in einem Programm ist der Rückgabewert daher ungültig.

Der kürzeste Zeitabstand zwischen zwei Befehlsaufrufen sollte mindestens das 100fache des **Processdelay** betragen. Anderenfalls kann der Rückgabewert einen Fehler von mehr als 1% haben.

Wenn der vorherige Aufruf von **Workload** länger als 85 Sekunden zurück liegt, ist der Rückgabewert ungültig. Rufen Sie den Befehl dann ein zweites Mal auf, um einen gültigen Rückgabewert zu erhalten.

### Siehe auch

[TiCo\\_Get\\_Processdelay](#), [Process\\_Status](#), [TiCo\\_Set\\_Processdelay](#), [TiCo\\_Stop\\_Process](#), [TDrv\\_Init](#)



### Gültig für

Gold II

### Beispiel

```
#Include ADwinGoldII.INC
#Define tico_no 1           'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer
```

#### Init:

```
TDrv_Init(tico_no,tset)
```

#### Event:

```
REM read TiCo workload
FPar_1 = Workload(tset)
```



## 7.4 Pro II: TiCo-Prozessor

Mit den folgenden *ADbasic*-Befehlen greift die *ADwin* CPU auf den *TiCo*-Prozessor zu:

Datenzugriff Initialisieren	P2_TDrv_Init, P2_Get_TiCo_Status
Globale Variablen lesen und schreiben	P2_Get_Par, P2_Get_Par_Block P2_Set_Par, P2_Set_Par_Block
Globale Felder lesen und schreiben	P2_GetData_Long, P2_SetData_Long
Ringspeicher lesen, schreiben und Status abfragen	P2_Get_TiCo_RingBuffer, P2_RingBuffer_Empty P2_Set_TiCo_RingBuffer, P2_RingBuffer_Full
Processdelay setzen und abfragen	P2_TiCo_Get_Processdelay, P2_TiCo_Set_Processdelay
Prozesse starten und stoppen, Prozessor steuern	P2_TiCo_Start_Process, P2_TiCo_Stop_Process P2_TiCo_Stop, P2_TiCo_Start, P2_TiCo_Reset
Systeminformationen abfragen	P2_Process_Status, P2_Workload
Binärdatei übertragen	P2_TiCo_Flash, P2_TiCo_Load

Beachten Sie bitte: Bei fast allen Befehlen muss zuerst die Datenübertragung mit **P2\_TDrv\_Init** initialisiert werden.

## P2\_Get\_Par

**P2\_Get\_Par** gibt den Wert einer globalen Variablen **Par\_x** von einem TiCo-Prozessor des angegebenen Moduls zurück.

### Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Get_Par(module, tico_no, Par_no)
```

### Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>tico_no</code>	Nummer (1...2) des TiCo-Prozessors auf dem Modul.	LONG
<code>Par_no</code>	Nummer (1...80) der globalen Variablen.	LONG
<code>ret_val</code>	Wert ( $-2^{31} \dots +2^{31}-1$ ) der globalen Variablen.	LONG

### Bemerkungen

Mehrere Werte werden mit **P2\_Get\_Par\_Block** schneller gelesen.

### Siehe auch

P2\_Get\_Par\_Block, P2\_GetData\_Long, P2\_Set\_Par, P2\_Set\_Par\_Block, P2\_SetData\_Long, P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

siehe Datei C:\ADwin\ADbasic\Examples\P2\_Get\_Par.bas.

```
#Include ADwinPro_All.INC
```

```
#Define module 4 'module address
```

```
#Define tico_no 1 'TiCo no.
```

```
Dim tset[150] As Long 'settings array for data
```

```
'transfer
```

#### Init:

```
P2_TDrv_Init(module,tico_no,tset)
```

#### Event:

```
REM read Par_1 from TiCo and write value to Par_2 of ADwin
```

```
REM CPU
```

```
Par_2 = P2_Get_Par(module,tico_no,1)
```

## P2\_Get\_Par\_Block

**P2\_Get\_Par\_Block** liest eine Anzahl von globalen Variablen **Par\_x** des TiCo-Prozessors vom angegebenen Modul und schreibt die Werte in ein Feld.

### Syntax

```
#Include ADwinPro_All.inc

P2_Get_Par_Block(module, tico_no, dest_
array[],
    dest_array_idx, Par_no, Par_count)
```

### Parameter

<b>module</b>	Eingestellte Moduladresse (1...15).	LONG
<b>tico_no</b>	Nummer (1...2) des TiCo-Prozessors auf dem Modul.	LONG
<b>dest_array[]</b>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<b>dest_array_idx</b>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte abgelegt werden.	LONG
<b>Par_no</b>	Index (1...80) der ersten globalen Variablen, die gelesen wird.	LONG
<b>Par_count</b>	Anzahl (1...80) der zu lesenden Variablen.	LONG

### Bemerkungen

- / -

### Siehe auch

P2\_Get\_Par, P2\_GetData\_Long, P2\_Set\_Par, P2\_Set\_Par\_Block, P2\_SetData\_Long, P2\_TDrv\_Init

**Gültig für**

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim Data_1[80] As Long
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)

Event:
    REM read Par_1...Par_80 from TiCo and write values to
    REM Data_1[1]...Data_1[80] of ADwin CPU
    P2_Get_Par_Block(module,tico_no,Data_1,1,1,80)
    REM read Par_20...Par_29 from TiCo and write values to
    REM Par_5...Par_14 of ADwin CPU
    P2_Get_Par_Block(module,tico_no,PAR,5,20,10)
```

## P2\_Get\_TiCo\_Bootloader\_Status

**P2\_Get\_TiCo\_Bootloader\_Status** gibt den Status des *TiCo*-Bootloaders auf dem angegebenen Modul zurück.

### Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Get_TiCo_Bootloader_  
Status (module)
```

### Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ret_val</code>	Status des Bootloaders: 0: Bootloader ist ausgeschaltet. 1: Bootloader ist eingeschaltet. -1: Fehler; Befehl bei hoher Priorität genutzt. -2: Fehler; Modul nicht ansprechbar (Timeout). -3: Fehler; Kein <i>TiCo</i> -Prozessor auf dem Modul.	LONG

### Bemerkungen

Der Befehl kann nur bei niedriger Prozesspriorität ausgeführt werden.

### Siehe auch

P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -



### P2\_Get\_TiCo\_RingBuffer

**P2\_Get\_TiCo\_RingBuffer** liest Werte aus einem Ringspeicher von einem TiCo-Prozessor und schreibt die Werte in ein Feld der ADwin CPU.

#### Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Get_TiCo_RingBuffer(tdrv_  
datatable[],  
    src_array_no, dest_array[], dest_array_
```

```
idx,
    maxcount, flowrate, tico_par, struct)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>src_</code>	Nummer (1...16) des Schreib-Ringspeichers	FLOAT
<code>array_no</code>	<b>Data_n</b> auf dem TiCo-Prozessor.	
<code>dest_</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>array[]</code>		LONG
<code>dest_</code>	Index (1...n) des ersten Elements im Feld <code>dest_</code>	LONG
<code>array_idx</code>	<code>array[]</code> , in das geschrieben wird.	
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Lesezeigers zum TiCo-Prozessor. 1...80: Nummer (1...80) der globalen Variablen <b>Par_n</b> des TiCo-Prozessors, in die der aktuelle Lesezeigerwert geschrieben wird. 0: Der Lesezeiger wird nicht übertragen.	LONG
<code>struct</code>	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten lesen. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von <code>struct</code> sein.	LONG

`ret_val`

Erfolgsstatus des Befehls:

LONG

-1: Fehler, der Schreib-Ringspeicher des TiCo ist nicht korrekt deklariert.

≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.

## Bemerkungen

Der Befehl liest nicht mehr als `maxcount` Daten. Wenn weniger Daten im Ringspeicher enthalten sind, werden alle vorhandenen Daten übertragen.

Beim Lesen von einem Ringspeicher speichert `P2_Get_TiCo_RingBuffer` die letzte Leseposition, den Lesezeiger, in dem Feld `tdrv_datatable[]`. Wenn der Befehl `RingBuffer_Empty` des TiCo-Prozessors korrekt arbeiten soll, muss in `tico_par` die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des TiCo-Prozessors übertragen.

## Siehe auch

P2\_Set\_TiCo\_RingBuffer, P2\_TDrv\_Init

## Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

## Beispiel

- / -

## P2\_Get\_TiCo\_Status

**P2\_Get\_TiCo\_Status** gibt zurück, ob *TiCo*-Prozessoren auf dem angegebenen Modul aktiv sind.

### Syntax

```
#Include ADwinPro_All.inc
ret_val = P2_Get_TiCo_Status(module)
```

### Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>ret_val</code>	Status der <i>TiCo</i> -Prozessoren. -3: Fehler, kein <i>TiCo</i> -Prozessor vorhanden. -2: Fehler, kein <i>Pro II</i> -Modul. ≥0: Bitmuster für den Betriebszustand der Prozessoren: Bit=0: Prozessor ist gestoppt. Bit=1: Prozessor läuft.	LONG

Bits in <code>ret_val</code>	31:01	00
<i>TiCo</i> -Prozessornr. auf dem Modul	–	1

### Bemerkungen

- / -

### Siehe auch

P2\_Process\_Status, P2\_Workload

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -

## P2\_GetData\_Long

**P2\_GetData\_Long** liest Werte aus einem globalen Feld von einem TiCo-Prozessor und schreibt die Werte in ein Feld.

### Syntax

```
#Include ADwinPro_All.inc  
  
P2_GetData_Long(tdrv_datatable[], src_array_  
no,
```

```
src_array_idx, count, dest_array[],  
dest_array_idx, flowrate)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>src_array_no</code>	Nummer (1...16) des globalen Felds <code>Data_x</code> auf dem TiCo-Prozessor.	LONG
<code>src_array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>src_array_no</code> gelesen wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>dest_array[]</code>	Zielfeld, in das die Werte übertragen werden.	ARRAY
<code>dest_array_idx</code>	Ziel-Startindex (1...n): Feldelement, ab dem die Werte im Zielfeld abgelegt werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

### Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `src_array_no` auf dem *TiCo*-Prozessor deklariert ist und
- das Feld `dest_array` mit mindestens `count` Elementen deklariert ist.

### Siehe auch

P2\_Get\_Par, P2\_Set\_Par, P2\_SetData\_Long, P2\_TDrv\_Init

**Gültig für**

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset_41[150] As Long   'settings array for data
                             'transfer

Dim Data_4[200] As Long

Init:
    REM initialize data transfer ADwin CPU <-> TiCo
    P2_TDrv_Init(module,tico_no,tset_41)

Event:
    REM read 120 values from TiCo Data_2 (starting from Data_
    REM 2[20])
    REM into Data_4 of ADwin CPU (starting from index 5).
    REM flowrate is high
    P2_GetData_Long(tset_41,2,20,120,Data_4,5,3)
```



## P2\_Process\_Status

**P2\_Process\_Status** gibt den den Status eines Prozesses auf einem TiCo-Prozessor des angegebenen Moduls zurück.

### Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Process_Status(module, tico_no,  
                             process_no)
```

### Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>tico_no</code>	Nummer (1...2) des TiCo-Prozessors auf dem Modul.	LONG
<code>process_no</code>	Nummer (1) des TiCo-Prozesses.	LONG
<code>ret_val</code>	Status des Prozesses: ≠1: Prozess läuft. 0: Prozess läuft nicht, d.h. er ist nicht geladen, nicht gestartet oder gestoppt.	LONG

### Bemerkungen

- / -

### Siehe auch

P2\_TiCo\_Get\_Processdelay, P2\_TiCo\_Set\_Processdelay,  
P2\_TiCo\_Start\_Process, P2\_TiCo\_Stop\_Process, P2\_Work-  
load

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-  
32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim Data_4[200] As Long
Dim tset[150] As Long      'settings array for data
                             'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)

Event:
    REM read status of TiCo process 1
    Par_1 = P2_Process_Status(module,tico_no,1)
    REM if process is running, read TiCo Par_5
    If (Par_1 = 1) Then
        Par_2 = P2_Get_Par(module,tico_no,5)
    EndIf
```

## P2\_RingBuffer\_Empty

**P2\_RingBuffer\_Empty** gibt die Anzahl der freien Elemente in einem Schreib-Ringspeicher eines TiCo-Prozessors zurück.

### Syntax

```
#Include ADwinPro_All.inc  
  
ret_val = P2_Ringbuffer_Empty(tdrv_  
datatable[],  
Data_no)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code> []	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>Data_no</code>	Nummer (1...16) des Schreib-Ringspeichers <code>Data_n</code> auf dem TiCo-Prozessor.	LONG
<code>ret_val</code>	Anzahl der freien Elemente im Schreib-Ring- speicher.	LONG

### Bemerkungen

Bei **P2\_GET\_TiCo\_RingBuffer** ist keine vorherige Abfrage mit **P2\_RingBuffer\_Empty** erforderlich.

### Siehe auch

P2\_Get\_TiCo\_RingBuffer, P2\_RingBuffer\_Full, P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -

## P2\_RingBuffer\_Full

**P2\_RingBuffer\_Full** gibt die Anzahl der genutzten Elemente in einem Lese-Ringspeicher eines TiCo-Prozessors zurück.

### Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_Ringbuffer_Full(tdrv_datatable[],
                             Data_no)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>[]</code>		
<code>Data_no</code>	Nummer (1...16) des Lese-Ringspeichers <code>Data_</code>	LONG
	<code>n</code> auf dem TiCo-Prozessor.	
<code>ret_val</code>	Anzahl (0...n) der genutzten Elemente im Lese-Ringspeicher.	LONG

### Bemerkungen

Bei **P2\_Set\_TiCo\_RingBuffer** ist keine vorherige Abfrage mit **P2\_RingBuffer\_Full** erforderlich.

### Siehe auch

P2\_Set\_TiCo\_RingBuffer, P2\_RingBuffer\_Empty, P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -

## P2\_Set\_Par

**P2\_Set\_Par** setzt den Wert einer globalen Variablen **Par\_x** auf einem TiCo-Prozessor des angegebenen Moduls.

### Syntax

```
#Include ADwinPro_All.inc  
P2_Set_Par(module, tico_no, Par_no, value)
```

### Parameter

<b>module</b>	Eingestellte Moduladresse (1...15).	LONG
<b>tico_no</b>	Nummer (1...2) des TiCo-Prozessors auf dem Modul.	LONG
<b>Par_no</b>	Nummer (1...80) der globalen Variablen.	LONG
<b>value</b>	Wert ( $-2^{31} \dots +2^{31}-1$ ) der globalen Variablen.	LONG

### Bemerkungen

**P2\_Set\_Par** setzt den Wert der globalen Variablen unabhängig davon, ob gerade ein TiCo-Prozess läuft. Da *ADwin* CPU- und TiCo-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den TiCo-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und TiCo mit Hilfe eines Software-Handshakes.

### Siehe auch

P2\_Get\_Par, P2\_Get\_Par\_Block, P2\_GetData\_Long, P2\_Set\_Par\_Block, P2\_SetData\_Long, P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)
    Par_5=200

Event:
    REM write value of Par_5 (ADwin CPU) to TiCo Par_2
    P2_Set_Par(module,tico_no,2,Par_5)
```

## P2\_Set\_Par\_Block

**P2\_Set\_Par\_Block** schreibt Werte aus einem Feld in eine Anzahl von globalen Variablen **Par\_x** des TiCo-Prozessors vom angegebenen Modul.

### Syntax

```
#Include ADwinPro_All.inc

P2_Set_Par_Block(module, tico_no, src_array[],
                 src_array_idx, Par_no, Par_count)
```

### Parameter

<b>module</b>	Eingestellte Moduladresse (1...15).	LONG
<b>tico_no</b>	Nummer (1...2) des TiCo-Prozessors auf dem Modul.	LONG
<b>src_array[]</b>	Quellfeld, aus dem Daten übertragen werden.	ARRAY
<b>src_array_idx</b>	Indes des Feldelements, ab dem Werte aus dem Quellfeld gelesen werden.	LONG
<b>Par_no</b>	Index (1...80) der ersten globalen TiCo-Variablen, die beschrieben wird.	LONG
<b>Par_count</b>	Anzahl der zu übertragenden Werte.	LONG

### Bemerkungen

**P2\_Set\_Par\_Block** setzt die Wert der globalen Variablen unabhängig davon, ob gerade ein TiCo-Prozess läuft. Da *ADwin* CPU- und TiCo-Prozesse nicht synchron arbeiten, können sich die Werte globaler Variablen für den TiCo-Prozess völlig unerwartet während der Laufzeit ändern.

Falls erforderlich, empfehlen wir eine Synchronisierung von *ADwin* CPU und *TiCo* mit Hilfe eines Software-Handshakes.

**Siehe auch**

P2\_Get\_Par, P2\_Get\_Par\_Block, P2\_GetData\_Long, P2\_Set\_Par  
P2\_SetData\_Long, P2\_TDrv\_Init

**Gültig für**

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

siehe Datei C:\ADwin\ADbasic\Examples\P2\_Set\_Par\_Block.bas.

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Dim Data_1[40] As Long
Dim i As Long
```

**Init:**

```
P2_TDrv_Init(module,tico_no,tset)
For i = 1 To 40
    Data_1[i] = i*10
Next
Par_15=1111
Par_16=2222
Par_17=3333
Par_18=4444
Par_19=5555
```

**Event:**

```
REM write 40 values from Data_1 (starting from Data_1[1])
REM into Par_1...Par_40 (TiCo)
P2_Set_Par_Block(module,tico_no,Data_1,1,1,40)
REM write Par_15...Par_19 (ADwin CPU) into Par_50...Par_54
REM (TiCo)
P2_Set_Par_Block(module,tico_no,PAR,15,50,5)
```



### P2\_Set\_TiCo\_RingBuffer

**P2\_Set\_TiCo\_RingBuffer** schreibt Werte aus einem Feld der *ADwin* CPU in einen Ringspeicher auf einem TiCo-Prozessor.

#### Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_Set_TiCo_RingBuffer(tdrv_  
datatable[],
```

```
dest_array_no, src_array[], src_array_idx,
maxcount, flowrate, tico_par, struct)
```

## Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>dest_array_no</code>	Nummer (1...16) des Lese-Ringspeichers <code>Data_n</code> auf dem TiCo-Prozessor.	FLOAT
<code>src_array[]</code>	Quellfeld, aus dem die Werte übertragen werden.	ARRAY
<code>src_array_idx</code>	Index (1...n) des ersten Elements im Feld <code>src_array[]</code> , das gelesen wird.	LONG
<code>maxcount</code>	Max. Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG
<code>tico_par</code>	Kennzahl zum Übertragen des Schreibzeigers zum TiCo-Prozessor. 1...80: Nummer (1...80) der globalen Variablen <code>Par_n</code> des TiCo-Prozessors, in die der aktuelle Schreibzeiger geschrieben wird. 0: Der Schreibzeiger wird nicht übertragen.	LONG
<code>struct</code>	Kennwert für Messwert-Sätze: 0: Beliebige Anzahl von Daten schreiben. >0: Die Anzahl der übertragenen Daten muss ein Vielfaches von <code>struct</code> sein.	LONG

`ret_val`

Erfolgsstatus des Befehls:

LONG

-1: Fehler, der Lese-Ringspeicher des TiCo ist nicht korrekt deklariert.

≥0: Befehl war erfolgreich. Der Rückgabewert ist die Anzahl der übertragenen Werte.

## Bemerkungen

Der Befehl schreibt nicht mehr als `maxcount` Daten. Wenn weniger freie Elemente im Ringspeicher vorhanden sind, werden nur die freien Elemente im Ringspeicher beschrieben.

Beim Schreiben in einen Ringspeicher speichert **P2\_Set\_TiCo\_RingBuffer** die letzte Schreibposition, den Schreibzeiger, in dem Feld `tdrv_datatable[]`. Wenn der Befehl **RingBuffer\_Full** des *TiCo*-Prozessors korrekt arbeiten soll, muss in `tico_par` die Nummer einer globalen Variablen angegeben werden. Dadurch wird der Wert des Lesezeigers in die globale Variable des *TiCo*-Prozessors übertragen.

## Siehe auch

P2\_Get\_TiCo\_RingBuffer, P2\_TDrv\_Init

## Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

## Beispiel

- / -

## P2\_SetData\_Long

**P2\_SetData\_Long** liest Werte aus einem Feld und schreibt sie in ein globales Feld eines TiCo-Prozessors auf dem angegebenen Modul.

### Syntax

```
#Include ADwinPro_All.inc  
  
P2_SetData_Long(tdrv_datatable[], dest_array_  
no,
```

```
dest_array_idx, count, src_array[],
src_array_idx, flowrate)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>dest_array_no</code>	Nummer (1...16) des globalen Felds <code>Data_x</code> auf dem TiCo-Prozessor.	LONG
<code>dest_array_idx</code>	Index (1...n) des ersten Elements, das im globalen Feld <code>dest_array_no</code> beschrieben wird.	LONG
<code>count</code>	Anzahl (1...n) der zu übertragenden Werte.	LONG
<code>src_array[]</code>	Quellfeld, aus dem die Werte gelesen werden.	ARRAY
<code>src_array_idx</code>	Quell-Startindex (1...n): Feldelement, ab dem die Werte gelesen werden.	LONG
<code>flowrate</code>	Auswertung nur für niederpriore Prozesse: Kennwert für den Datendurchsatz. 1: langsam. 2: mittel. 3: schnell.	LONG

### Bemerkungen

Es muss gewährleistet sein, dass

- das globale Feld `dest_array_no` auf dem TiCo-Prozessor deklariert ist und
- das Feld mit mindestens `count` Elementen deklariert ist.

### Siehe auch

P2\_Get\_Par, P2\_GetData\_Long, P2\_Set\_Par, P2\_TDrv\_Init

**Gültig für**

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                             'transfer

Dim Data_1[150] As Long
Dim tset_41[150] As Long
Dim i As Long
```

**Init:**

```
REM initialize data transfer ADwin CPU <-> TiCo
P2_TDrv_Init(module,tico_no,tset_41)
For i = 1 To 80
    Data_1[i] = i
Next
```

**Event:**

```
REM read 120 values from TiCo Data_2 (starting from Data_
REM 2[1])
REM into Data_1 of ADwin CPU (starting from Data_1[5]).
REM flowrate is high
P2_SetData_Long(tset_41,2,1,120,Data_1,5,3)
P2_TDrv_Init(module,tico_no,tset)
REM start TiCo process in parallel to ADwin CPU process
P2_Tico_Start_Process(tset,1)

P2_TDrv_Init(module,tico_no,tset)
REM start TiCo process in parallel to ADwin CPU process
P2_Tico_Stop_Process(tset,1)
```

## P2\_TDrv\_Init

**P2\_TDrv\_Init** initialisiert die Datenübertragung zwischen *ADwin* CPU und einem bestimmten *TiCo*-Prozessor.

### Syntax

```
#Include ADwinPro_All.inc

REM define settings array for TiCo y on module x
DIM tdrv_datatable[150] AS LONG

P2_TDrv_Init(module, tico_no, tdrv_
datatable[])
```

### Parameter

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors auf dem Modul.	LONG
<code>tdrv_</code> <code>datatable</code>	Feld, das Einstellungen für die Datenübertragung aufnimmt.	ARRAY LONG
<code>[]</code>		

### Bemerkungen

Der Befehl muss vor der Datenübertragung zwischen *ADwin* CPU und *TiCo* ausgeführt werden. Der Befehl sollte im Abschnitt **Init:** stehen.

Fast alle Befehle, die auf den *TiCo* zugreifen, benötigen eine Initialisierung der Datenübertragung.

Die Initialisierung muss für jeden *TiCo*-Prozessor separat durchgeführt werden. Dabei muss auch jeweils ein Feld `tdrv_datatable[]` mit 150 Elementen angelegt werden.

Das Feld `tdrv_datatable[]` wird von den Befehlen **P2\_GetData\_Long**, **P2\_SetData\_Long** und **P2\_Workload** verwendet.

**Siehe auch**

P2\_Get\_Par, P2\_Get\_Par\_Block, P2\_TiCo\_Get\_Processdelay, P2\_GetData\_Long, P2\_Set\_Par, P2\_Set\_Par\_Block, P2\_TiCo\_Set\_Processdelay, P2\_SetData\_Long

**Gültig für**

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.inc
#Define module_a 4          'address of module a
#Define TiCo_a 1            'TiCo no.
#Define module_b 7          'address of module b
#Define TiCo_b 1            'TiCo no.
Dim Data_4[200] As Long
Dim Data_5[1000] As Long
Dim tset_41[150] As Long
Dim tset_71[150] As Long
```

**Init:**

```
REM initialize data transfer ADwin CPU <-> TiCo
P2_TDrv_Init(module_a,TiCo_a,tset_41)
P2_TDrv_Init(module_b,TiCo_b,tset_71)
```

**Event:**

```
REM read 120 values from module a, TiCo Data_2 (starting
REM from
REM Data_2[20]) into Data_4 of ADwin CPU (starting from
REM index 5)
REM flowrate is high
P2_GetData_Long(tset_41,2,20,120,Data_4,5,3)
REM read 800 values from module b, TiCo Data_7 (starting
REM from
REM Data_7[9]) into Data_5 of ADwin CPU (starting from
REM index 1).
REM flowrate is high
P2_GetData_Long(tset_71,7,9,800,Data_5,1,3)
```



## P2\_TiCo\_Get\_Processdelay

**P2\_TiCo\_Get\_Processdelay** gibt das Processdelay (die Zykluszeit) eines Prozesses auf einem *TiCo*-Prozessor des angegebenen Moduls zurück.

### Syntax

```
#Include ADwinPro_All.inc

ret_val = P2_TiCo_Get_Processdelay (
    tdrv_datatable[], process_no)
```

### Parameter

<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung enthält, u.a. Moduladresse und TiCo-Nummer.	ARRAY
<code>process_no</code>	Nummer (1) des TiCo-Prozesses.	LONG
<code>ret_val</code>	Aktuell eingestellte Zykluszeit ( $-2^{31} \dots +2^{31}-1$ ) des TiCo-Prozessors in Taktzyklen. Ein Taktzyklus dauert 20ns.	LONG

### Bemerkungen

Bei einem zeitgesteuerten Prozess wird der Abschnitt **Event:** vom internen Zähler zyklisch und in festen Zeitabständen aufgerufen. Der Zeitabstand zwischen zwei Aufrufen, Zykluszeit oder Processdelay genannt, wird in Zähler-Taktzyklen gezählt.

### Siehe auch

P2\_Process\_Status, P2\_TiCo\_Set\_Processdelay, P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.INC
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)

Event:
    REM read processdelay of process 1 into Par_1
    Par_1 = P2_TiCo_Get_Processdelay(tset,1)
```

### P2\_TiCo\_Flash

**P2\_TiCo\_Flash** überträgt eine *TiCoBasic*-Binärdatei aus einem Feld in den Flash-Speicher eines *TiCo*-Prozessors.

**Syntax**

```
#Include ADwinPro_All.inc

ret_val = P2_TiCo_Flash(module,tico_
no,array[])
```

**Parameter**

<code>module</code>	Eingestellte Moduladresse (1...15).	LONG
<code>tico_no</code>	Nummer (1...2) des <i>TiCo</i> -Prozessors auf dem Modul.	LONG
<code>array[]</code>	Feld, in dem die zu übertragenden Daten enthalten sind.	ARRAY
<code>ret_val</code>	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -2: kein Modul an dieser Adresse oder das Modul hat keinen <i>TiCo</i> -Prozessor. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 1: Fehler: Ungültige Prozessornummer. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Externer Speicher ist zu klein. 7: Fehler: Flash-Speicher (EEPROM) ist zu klein. 12:Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

**Bemerkungen**

Verwenden Sie **P2\_TiCo\_Flash** nur in niederprioren Prozessen.

Der Befehl **P2\_TiCo\_Flash** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den Flash-Spei-

cher eines *TiCo*-Prozessors zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit `Build ► Make Bin File`.
- Übertragen Sie die Binärdatei in ein globales Feld der *ADwin* CPU. Geeignet ist die Funktion `Data2File`, die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld `array[]` mit dem Befehl **P2\_TiCo\_Flash** in den Flash-Speicher .

Wenn das Feld `array[]` keine *TiCoBasic*-Binärdatei enthält, ist der Rückgabewert des Befehls ungültig.

### Siehe auch

P2\_TiCo\_Load

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -

## P2\_TiCo\_Load

**P2\_TiCo\_Load** überträgt den Inhalt eines Felds in den Speicher eines *TiCo*-Prozessors.

## Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_TiCo_Load(module,tico_no,array[])
```

## Parameter

module	Eingestellte Moduladresse (1...15).	LONG
tico_no	Nummer (1...2) des TiCo-Prozessors auf dem Modul.	LONG
array[]	Feld, in dem die zu übertragenden Daten enthalten sind.	ARRAY LONG
ret_val	Status der Datenübertragung: 0: Datenübertragung erfolgreich. -2: kein Modul an dieser Adresse oder das Modul hat keinen <i>TiCo</i> -Prozessor. -1: Fehler: Befehl darf nur bei niedriger Priorität verwendet werden. 1: Fehler: Ungültige Prozessornummer. 2: Fehler: Programmspeicher ist zu klein. 3: Fehler: Datenspeicher ist zu klein. 4: Fehler: Programm- und Datenspeicher sind zu klein. 5: Fehler: Falsches Passwort. 6: Fehler: Externer Speicher ist zu klein. 10: Fehler: Der benötigte externe SRAM-Speicher ist zu klein. 11: Fehler: Das Feld enthält keine gültige <i>TiCo-Basic</i> -Binärdatei. 12: Fehler: Die Binärdatei ist nicht für den <i>TiCo</i> -Prozessor geeignet.	LONG

## Bemerkungen

Verwenden Sie **P2\_TiCo\_Load** nur in niederprioren Prozessen.

Der Befehl **P2\_TiCo\_Load** dient dazu, eine *TiCoBasic*-Binärdatei – ein kompiliertes *TiCo*-Programm – in den *TiCo*-Prozessor zu übertragen. Dazu sind folgende Schritte erforderlich:

- Erzeugen Sie in der Oberfläche *TiCoBasic* eine Binärdatei mit **Build ▶ Make Bin File**.
- Übertragen Sie die Binärdatei in ein globales Feld des *ADwin*-Prozessors. Geeignet ist die Funktion **File2Data**, die in mehreren Programmiersprachen zur Verfügung steht.
- Übertragen Sie die Daten im globalen Feld **array[]** mit dem Befehl **P2\_TiCo\_Load** auf den *TiCo*-Prozessor.

### Siehe auch

P2\_TiCo\_Flash

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -



## P2\_TiCo\_Reset

**P2\_TiCo\_Reset** stoppt die TiCo-Prozessoren auf den angegebenen Modulen und startet sie anschließend wieder.

### Syntax

```
#Include ADwinPro_All.inc  
P2_TiCo_Reset(module_pattern)
```

### Parameter

`module_` Bitmuster zum Ansprechen der Module, deren LONG  
`pattern` TiCos gestoppt werden sollen:  
Bit = 0: Modul ignorieren.  
Bit = 1: TiCos auf dem Modul stoppen.

Bits in <code>module_pattern</code>	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

### Bemerkungen

Das Stoppen beendet auf den angesprochenen TiCo-Prozessoren sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

Die TiCo-Prozessoren auf den gewählten Modulen werden gleichzeitig gestartet. Der Befehl eignet sich daher, um alle angesprochenen TiCo-Prozessoren zu synchronisieren.

### Siehe auch

P2\_TiCo\_Stop, P2\_TiCo\_Start

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

- / -

## P2\_TiCo\_Set\_Processdelay

**P2\_TiCo\_Set\_Processdelay** setzt das Processdelay (die Zykluszeit) eines Prozesses auf einem TiCo-Prozessor des angegebenen Moduls.

### Syntax

```
#Include ADwinPro_All.inc  
  
P2_TiCo_Set_Processdelay(tdrv_datatable[],  
    process_no, value)
```

### Parameter

<code>tdrv_datatable</code>	Feld, das Einstellungen für die Datenübertragung enthält, u.a. Moduladresse und TiCo-Nummer.	ARRAY
<code>process_no</code>	Nummer (1) des TiCo-Prozesses.	LONG
<code>value</code>	Einzustellender Wert für das Processdelay.	LONG

### Bemerkungen

- / -

### Siehe auch

P2\_TiCo\_Get\_Processdelay, P2\_Process\_Status, P2\_TDrv\_Init

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    P2_TDrv_Init(module,tico_no,tset)
    Processdelay = 6000 'set cycle time

Event:
    REM set TiCo processdelay to run with same cycle time as
    REM the
    REM T12 process
    P2_TiCo_Set_Processdelay(tset,1,Processdelay/6)
```

## P2\_TiCo\_Start

**P2\_TiCo\_Start** startet die TiCo-Prozessoren auf den angegebenen Modulen.

### Syntax

```
#Include ADwinPro_All.inc

P2_TiCo_Start(module_pattern)
```

### Parameter

**module\_** Bitmuster zum Ansprechen der Module, deren LONG  
**pattern** TiCos gestartet werden sollen:  
 Bit = 0: Modul ignorieren.  
 Bit = 1: TiCos auf dem Modul starten.

Bits in <b>module_pattern</b>	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

### Bemerkungen

Die TiCo-Prozessoren auf den gewählten Modulen werden gleichzeitig gestartet. Der Befehl eignet sich daher, um alle angesprochenen TiCo-Prozessoren zu synchronisieren.

### Siehe auch

P2\_TiCo\_Stop, P2\_TiCo\_Reset

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -

## P2\_TiCo\_Start\_Process

**P2\_TiCo\_Start\_Process** startet einen Prozess auf einem TiCo-Prozessor.

### Syntax

```
#Include ADwinPro_All.inc

P2_TiCo_Start_Process (tdrv_datatable[],
process_no)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>process_</code>	Nummer (1) des TiCo-Prozesses.	LONG
<code>no</code>		

### Bemerkungen

Der Prozess muss bereits auf dem TiCo-Prozessor vorhanden sein.

### Siehe auch

P2\_TiCo\_Get\_Processdelay, P2\_Process\_Status, P2\_TiCo\_Set\_Processdelay, P2\_TiCo\_Start\_Process, P2\_TiCo\_Stop\_Process, P2\_Workload

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
  Par_1 = 0
  Processdelay = 100000
  P2_TDrv_Init(module,tico_no,tset)
  REM start TiCo process in parallel to ADwin CPU process
  P2_Tico_Start_Process(tset,1)

Event:
  REM ... program

Finish:
  REM stop TiCo process in parallel to ADwin CPU process
  P2_Tico_Stop_Process(tset,1)
```

## P2\_TiCo\_Stop

**P2\_TiCo\_Stop** stoppt die TiCo-Prozessoren auf den angegebenen Modulen.

### Syntax

```
#Include ADwinPro_All.inc

P2_TiCo_Stop(module_pattern)
```

### Parameter

`module_` Bitmuster zum Ansprechen der Module, deren LONG  
`pattern` TiCos gestoppt werden sollen:  
 Bit = 0: Modul ignorieren.  
 Bit = 1: TiCos auf dem Modul stoppen.

Bits in <code>module_pattern</code>	31:15	14	13	...	01	00
Moduladresse	–	15	14	...	2	1

### Bemerkungen

Das Stoppen beendet auf den angesprochenen TiCo-Prozessoren sofort jeden Prozess sowie den Zähler. Die Daten werden durch das Stoppen nicht verändert.

### Siehe auch

P2\_TiCo\_Start, P2\_TiCo\_Reset

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

### Beispiel

- / -



## P2\_TiCo\_Stop\_Process

**P2\_TiCo\_Stop\_Process** stoppt einen Prozess auf einem TiCo-Prozessor.

### Syntax

```
#Include ADwinPro_All.inc  
  
P2_TiCo_Stop_Process (tdrv_datatable[],  
process_no)
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>[]</code>		
<code>process_</code>	Nummer (1) des TiCo-Prozesses.	LONG
<code>no</code>		

### Bemerkungen

Der Prozess muss auf dem TiCo-Prozessor vorhanden sein.

### Siehe auch

P2\_TiCo\_Get\_Processdelay, P2\_Process\_Status, P2\_TiCo\_Set\_Processdelay, P2\_TiCo\_Start\_Process, P2\_Workload

### Gültig für

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer

Init:
    Par_1 = 0
    Processdelay = 100000
    P2_TDrv_Init(module,tico_no,tset)
    REM start TiCo process in parallel to ADwin CPU process
    P2_Tico_Start_Process(tset,1)

Event:
    REM ... program

Finish:
    REM stop TiCo process in parallel to ADwin CPU process
    P2_Tico_Stop_Process(tset,1)
```

## P2\_Workload

**P2\_Workload** gibt die Auslastung eines TiCo-Prozessors auf dem angegebenen Modul zurück.

### Syntax

```
#Include ADwinPro_All.inc  
ret_val = P2_Workload(tdrv_datatable[])
```

### Parameter

<code>tdrv_</code>	Feld, das Einstellungen für die Datenübertragung	ARRAY
<code>datatable</code>	enthält, u.a. Moduladresse und TiCo-Nummer.	LONG
<code>[]</code>		
<code>ret_val</code>	Prozessor-Auslastung in Prozent (0.0 ... 100.0) oder Fehlerwert: <0: TiCo-Prozessor ist gestoppt oder das Modul besitzt keinen TiCo-Prozessor oder kein Modul an der übergebenen Adresse.	FLOAT

### Bemerkungen

Der Rückgabewert ist die mittlere Prozessorauslastung in der Zeit zwischen dem vorherigen und dem aktuellen Aufruf von **P2\_Workload**. Beim ersten Aufruf in einem Programm ist der Rückgabewert daher ungültig.

Der kürzeste Zeitabstand zwischen zwei Befehlsaufrufen sollte mindestens das 100fache des **Processdelay** betragen. Anderenfalls kann der Rückgabewert mit einem Fehler über 1% behaftet sein.

Wenn der vorherige Aufruf von **P2\_Workload** länger als 85 Sekunden zurück liegt, ist der Rückgabewert ungültig. Rufen Sie den Befehl einfach ein zweites Mal auf, um einen gültigen Rückgabewert zu erhalten.

**Siehe auch**

P2\_TiCo\_Get\_Processdelay, P2\_Process\_Status, P2\_TiCo\_Set\_Processdelay, P2\_TiCo\_Stop\_Process, P2\_TDrv\_Init

**Gültig für**

CAN-2 Rev. E, CNT-D Rev. E, Cnt-I Rev. E, Cnt-T Rev. E, DIO-32-TiCo Rev. E, RSxxx-2 Rev. E, RSxxx-4 Rev. E

**Beispiel**

```
#Include ADwinPro_All.Inc
#Define module 4           'module address
#Define tico_no 1          'TiCo no.
Dim tset[150] As Long      'settings array for data
                           'transfer
```

**Init:**

```
P2_TDrv_Init(module,tico_no,tset)
```

**Event:**

```
REM read TiCo workload
FPar_1 = P2_Workload(tset)
```

## 8 Was tun bei Problemen?

Wenn Sie bei der Installation Probleme haben, ziehen Sie bitte die Dokumentation zu Ihrem *ADwin*-System zu Rate. Überprüfen Sie, ob alle Einstellungen richtig und vollständig durchgeführt wurden. Prüfen Sie auch, ob die Einstellungen im Menü „Options\Compiler“ richtig sind.

Sollten Ihre Probleme dann immer noch bestehen, rufen Sie uns bitte an. Auch wenn Sie weitergehende Hilfe wünschen, stehen wir Ihnen gern zur Verfügung; Sie finden Adresse und Telefonnummer Ihres Ansprechpartners in der vorderen Umschlagseite des Handbuchs.




## Anhang

### A.1 Tastaturkürzel in TiCoBasic

Um die Tastaturkürzel für Textbausteine zu sehen, öffnen Sie die Datei `<TiCoBasicCS.xml>` in `C:\ADwin\TiCoBasic\Common\` mit einem Browser.

Tastenkürzel	Funktion	Menüaufruf
F1	Hilfethema für den markierten Befehl aufrufen.	
CTRL-F1	Inhaltsverzeichnis der Hilfe aufrufen.	Help ► Content
F2	Deklaration des markierten Befehls anzeigen.	
CTRL-F2	Zur Deklaration des markierten Befehls springen.	
F3	Text nochmals vorwärts suchen.	Edit ► Find Next
SHIFT-F3	Text nochmals rückwärts suchen.	
CTRL-F3	Text an Cursor-Position vorwärts suchen.	
CTRL-SHIFT-F3	Text an Cursor-Position rückwärts suchen.	
CTRL-F5	ADwin-CPU für Kommunikation mit dem TiCo-Prozessor initialisieren.	
CTRL-SHIFT-F5	TiCo-Prozessor sofort stoppen und zurücksetzen	
F6	Bibliothek erzeugen.	Build ► Make Lib File
F7	Binärdatei erzeugen.	Build ► Make Bin File
F8	Quelltext kompilieren.	Build ► Compile
CTRL-F8	Prozess starten.	
F9	Prozess stoppen.	
CTRL-SPACE	Deklaration einfügen oder vervollständigen.	
CTRL-SHIFT-SPACE	Parameter einer Sub / Function anzeigen.	

Tastenkürzel	Funktion	Menüaufruf
CTRL-A	Alles markieren.	Edit ► Select All
CTRL-B	Markierte Zeilen in Kommentar umwandeln.	Source context menu: Comment Block
CTRL-SHIFT-B	Kommentarzeichen aus markierten Zeilen löschen.	Source context menu: Uncomment Block
CTRL-C	Kopieren.	Edit ► Copy
CTRL-F	Text suchen.	Edit ► Find
CTRL-G	Zu einer Zeile springen.	
CTRL-H	Text ersetzen.	Edit ► Replace
CTRL-I	Zeilen nach rechts einrücken	Source context menu: Indent
CTRL-SHIFT-I	Zeilen nach links rücken	Source context menu: Outdent
CTRL-N	Neue Quelltextdatei.	File ► New
CTRL-O	Quelltextdatei öffnen.	File ► Open
CTRL-P	Quelltextdatei drucken.	File ► Print
CTRL-R	Verwendete Parameter markieren.	Parameter window: Icon 
CTRL-S	Quelltextdatei speichern.	File ► Save
CTRL-V	Einfügen.	Edit ► Paste
CTRL-X	Ausschneiden.	Edit ► Cut
CTRL-Z	Änderung zurücknehmen.	Edit ► Undo
CTRL-SHIFT-Z	Änderung wieder herstellen.	Edit ► Redo
CTRL-K + K	Textmarke ein- oder ausschalten.	
CTRL-K + N	Zur nächsten Textmarke springen.	
CTRL-K + P	Zur vorigen Textmarke springen.	
CTRL-K + X	Einen Textbaustein aus einer Liste einfügen.	

Legende:



A-B: Tasten A und B gleichzeitig drücken.

A+B: Tasten A und B nacheinander drücken, erst A, dann B.

## A.2 ASCII-Zeichensatz

<b>NUL</b> 00h 0	<b>SOH</b> 01h 1	<b>STX</b> 02h 2	<b>ETX</b> 03h 3	<b>EOT</b> 04h 4	<b>ENQ</b> 05h 5	<b>ACK</b> 06h 6	<b>BEL</b> 07h 7
<b>BS</b> <sup>1</sup> 08h 8	<b>TAB</b> <sup>2</sup> 09h 9	<b>LF</b> <sup>3</sup> 0Ah 10	<b>VT</b> 0Bh 11	<b>FF</b> 0Ch 12	<b>CR</b> <sup>4</sup> 0Dh 13	<b>SO</b> 0Eh 14	<b>SI</b> 0Fh 15
<b>DLE</b> 10h 16	<b>DC1</b> 11h 17	<b>DC2</b> 12h 18	<b>DC3</b> 13h 19	<b>DC4</b> 14h 20	<b>NAK</b> 15h 21	<b>SYN</b> 16h 22	<b>ETB</b> 17h 23
<b>CAN</b> 18h 24	<b>EM</b> 19h 25	<b>SUB</b> 1Ah 26	<b>ESC</b> 1Bh 27	<b>FS</b> 1Ch 28	<b>GS</b> 1Dh 29	<b>RS</b> 1Eh 30	<b>US</b> 1Fh 31
<b>SPC</b> <sup>5</sup> 20h 32	<b>!</b> 21h 33	<b>"</b> 22h 34	<b>#</b> 23h 35	<b>\$</b> 24h 36	<b>%</b> 25h 37	<b>&amp;</b> 26h 38	<b>'</b> 27h 39
<b>(</b> 28h 40	<b>)</b> 29h 41	<b>*</b> 2Ah 42	<b>+</b> 2Bh 43	<b>,</b> 2Ch 44	<b>-</b> 2Dh 45	<b>.</b> 2Eh 46	<b>/</b> 2Fh 47
<b>0</b> 30h 48	<b>1</b> 31h 49	<b>2</b> 32h 50	<b>3</b> 33h 51	<b>4</b> 34h 52	<b>5</b> 35h 53	<b>6</b> 36h 54	<b>7</b> 37h 55
<b>8</b> 38h 56	<b>9</b> 39h 57	<b>:</b> 3Ah 58	<b>;</b> 3Bh 59	<b>&lt;</b> 3Ch 60	<b>=</b> 3Dh 61	<b>&gt;</b> 3Eh 62	<b>?</b> 3Fh 63
<b>@</b> 40h 64	<b>A</b> 41h 65	<b>B</b> 42h 66	<b>C</b> 43h 67	<b>D</b> 44h 68	<b>E</b> 45h 69	<b>F</b> 46h 70	<b>G</b> 47h 71
<b>H</b> 48h 72	<b>I</b> 49h 73	<b>J</b> 4Ah 74	<b>K</b> 4Bh 75	<b>L</b> 4Ch 76	<b>M</b> 4Dh 77	<b>N</b> 4Eh 78	<b>O</b> 4Fh 79
<b>P</b> 50h 80	<b>Q</b> 51h 81	<b>R</b> 52h 82	<b>S</b> 53h 83	<b>T</b> 54h 84	<b>U</b> 55h 85	<b>V</b> 56h 86	<b>W</b> 57h 87
<b>X</b> 58h 88	<b>Y</b> 59h 89	<b>Z</b> 5Ah 90	<b>[</b> 5Bh 91	<b>\</b> 5Ch 92	<b>]</b> 5Dh 93	<b>^</b> 5Eh 94	<b>_</b> 5Fh 95
<b>`</b> 60h 96	<b>a</b> 61h 97	<b>b</b> 62h 98	<b>c</b> 63h 99	<b>d</b> 64h 100	<b>e</b> 65h 101	<b>f</b> 66h 102	<b>g</b> 67h 103
<b>h</b> 68h 104	<b>i</b> 69h 105	<b>j</b> 6Ah 106	<b>k</b> 6Bh 107	<b>l</b> 6Ch 108	<b>m</b> 6Dh 109	<b>n</b> 6Eh 110	<b>o</b> 6Fh 111
<b>p</b> 70h 112	<b>q</b> 71h 113	<b>r</b> 72h 114	<b>s</b> 73h 115	<b>t</b> 74h 116	<b>u</b> 75h 117	<b>v</b> 76h 118	<b>w</b> 77h 119
<b>x</b> 78h 120	<b>y</b> 79h 121	<b>z</b> 7Ah 122	<b>{</b> 7Bh 123	<b> </b> 7Ch 124	<b>}</b> 7Dh 125	<b>~</b> 7Eh 126	<b>□</b> 7Fh 127

<sup>1</sup> Backspace, <sup>2</sup> Tabulator, <sup>3</sup> Linefeed,  
<sup>4</sup> Carriage Return, <sup>5</sup> Space

### A.3 Lizenzvertrag

Zwischen dem Käufer von *TiCoBasic* – nachfolgend „Lizenznehmer“ genannt –

und Jäger Computergesteuerte Messtechnik GmbH, Rheinstraße 2 - 4, 64653 Lorsch – nachfolgend „Jäger Messtechnik GmbH“ genannt – besteht der folgende Lizenzvertrag:

#### 1. GEGENSTAND DES LIZENZVERTRAGES

1.1 Gegenstand des Lizenzvertrages ist die Software des Compilers und Entwicklungssystems *TiCoBasic* (im folgenden als „*TiCoBasic*-Software“ bezeichnet) und die gedruckte Benutzerdokumentation mit der Bezeichnung „*TiCoBasic*: Das Echtzeit-Entwicklungstool für *ADwin*-Systeme“ (im folgenden als „zugehöriges Schriftmaterial“ bezeichnet).

1.2 Die Jäger Messtechnik GmbH macht darauf aufmerksam, dass es nach dem Stand der Technik nicht möglich ist, Computersoftware so zu erstellen, dass sie in allen Anwendungen und Kombinationen fehlerfrei arbeitet. Gegenstand des Lizenzvertrages ist nur eine Computersoftware, die im Sinne der Benutzerdokumentation grundsätzlich brauchbar ist.

#### 2. UMFANG DER BENUTZUNG

2.1 Die Jäger Messtechnik GmbH gewährt dem Lizenznehmer das einfache, nicht ausschließliche und persönliche Nutzungsrecht. Dies bedeutet, dass die beiliegende Kopie der *TiCoBasic*-Software nur auf einem einzelnen Computer und nur an einem Ort benutzt werden darf. Der Lizenznehmer darf die *TiCoBasic*-Software in körperlicher Form (d. h. auf einem Datenträger abgespeichert) von einem Computer auf einen anderen Computer übertragen, vorausgesetzt, dass sie zu jedem Zeitpunkt immer nur auf einem einzelnen Computer genutzt wird. Eine weitergehende Nutzung ist nicht zulässig.

2.2 Programme, die durch den Lizenznehmer mit der *TiCoBasic*-Software erstellt werden, dürfen uneingeschränkt verteilt und weiterverwendet werden.

#### 3. BESONDERE BESCHRÄNKUNGEN

Dem Lizenznehmer ist es untersagt,

- a) ohne vorherige schriftliche Einwilligung der Jäger Messtechnik GmbH die *TiCoBasic*-Software oder das zugehörige schriftliche Material an

einen Dritten zu übergeben oder einem Dritten sonstwie zugänglich zu machen,

- b) die *TiCoBasic*-Software von einem Computer über ein Netz oder einen Datenübertragungskanal auf einen anderen Computer zu übertragen,
- c) ohne vorherige schriftliche Einwilligung der Jäger Messtechnik GmbH die *TiCoBasic*-Software abzuändern, zu übersetzen, zurückzuentwickeln, zu entkompilieren oder zu disassemblieren,

#### 4. INHABERSCHAFT AN RECHTEN

4.1 Der Lizenznehmer erhält mit dem Erwerb des Produktes nur Eigentum an dem körperlichen Datenträger, auf dem die *TiCoBasic*-Software aufgezeichnet ist. Ein Erwerb von Rechten an der *TiCoBasic*-Software selbst ist damit nicht verbunden.

4.2 Die Jäger Messtechnik GmbH behält sich insbesondere alle Veröffentlichungs-, Vervielfältigungs-, Bearbeitungs- und Verwertungsrechte an der *TiCoBasic*-Software vor.

#### 5. VERVIELFÄLTIGUNG

5.1 Die *TiCoBasic*-Software und das zugehörige Schriftmaterial sind urheberrechtlich geschützt.

Zu Sicherungszwecken ist dem Lizenznehmer das Anfertigen einer einzigen Reservekopie der *TiCoBasic*-Software erlaubt. Er ist verpflichtet, auf der Reservekopie den Urheberrechtsvermerk der Jäger Messtechnik GmbH anzubringen. Der in der *TiCoBasic*-Software vorhandene Urheberrechtsvermerk darf nicht entfernt werden.

5.2 Es wird ausdrücklich untersagt, die *TiCoBasic*-Software, wie auch das zugehörige Schriftmaterial, ganz oder teilweise in ursprünglicher oder abgeänderter Form oder in mit anderer Software zusammengemischer oder in anderer Software eingeschlossener Form zu kopieren oder anders zu vervielfältigen.

#### 6. ÜBERTRAGUNG DES BENUTZUNGSRECHTES

6.1 Das Recht zur Benutzung der *TiCoBasic*-Software kann nur mit vorheriger schriftlicher Einwilligung der Jäger Messtechnik GmbH an einen Dritten übertragen werden. Der Lizenznehmer hat dann die bei ihm installierte Software vollständig zu löschen und dem Dritten die Software vollständig (Original-Datenträger mit Dokumentation, einschließlich der Sicherungskopie) zu übergeben. Das Nutzungsrecht darf ferner nur auf einen Dritten übertragen werden, wenn sich der Dritte zu

Gunsten der Jäger Messtechnik GmbH mit den Bestimmungen dieses Lizenzvertrages und den allgemeinen Geschäftsbedingungen der Jäger Messtechnik GmbH einverstanden erklärt.

6.2 Vermietung und Verleihung der *TiCoBasic*-Software sind ausdrücklich untersagt.

## 7. DAUER DES VERTRAGES

7.1 Der Lizenzvertrag läuft auf unbestimmte Zeit.

7.2 Das Recht des Lizenznehmers zur Benutzung der *TiCoBasic*-Software erlischt automatisch ohne Kündigung, wenn er eine Bedingung dieses Lizenzvertrages verletzt. Bei Beendigung des Nutzungsrechtes ist er verpflichtet, den Original-Datenträger und alle Kopien der *TiCoBasic*-Software einschließlich etwaiger abgeänderter Exemplare sowie das zugehörige Schriftmaterial zu vernichten.

## 8. SCHADENSERSATZ UND VERTRAGSSTRAFE BEI VERTRAGS- VERLETZUNG

8.1 Verletzt der Lizenznehmer Bestimmungen dieses Lizenzvertrages, so ist er zum Schadensersatz verpflichtet.

8.2 Unbeschadet dessen wird bei Verletzung des Urheberrechts der Jäger Messtechnik GmbH, unbefugter Benutzung der Software und unbefugter Weitergabe der Software an Dritte, eine Vertragsstrafe von 20.000,- EURO für jeden Fall der Zuwiderverhandlung vereinbart.

8.3 Unterlassungsansprüche werden von den Schadensersatzansprüchen und Vertragsstrafen nicht berührt.

## 9. ÄNDERUNGEN UND AKTUALISIERUNGEN

Die Jäger Messtechnik GmbH ist berechtigt, Aktualisierungen der *TiCoBasic*-Software nach eigenem Ermessen zu erstellen. Die Jäger Messtechnik GmbH ist nicht verpflichtet, Aktualisierungen der *TiCoBasic*-Software dem Lizenznehmer zur Verfügung zu stellen.

Für umfangreiche Aktualisierungen behält sich die Jäger Messtechnik GmbH vor, einen Kostenbeitrag zu erheben.

## 10. GEWÄHRLEISTUNG UND HAFTUNG DER JÄGER MESSTECHNIK GMBH

a) Die Jäger Messtechnik GmbH gewährleistet gegenüber dem Lizenznehmer, dass zum Zeitpunkt der Übergabe der Datenträger, auf dem die *TiCoBasic*-Software aufgezeichnet ist, unter normalen Betriebsbe-

dingungen und bei normaler Instandhaltung in seiner Materialausführung fehlerfrei ist.

- b) Sollte der Datenträger fehlerhaft sein, so kann der Lizenznehmer Ersatzlieferung während der Gewährleistungszeit von 6 Monaten ab Lieferung verlangen. Er muss dazu den Datenträger einschließlich einer Kopie der Rechnung/Quittung an die Jäger Messtechnik GmbH oder an den Vertriebspartner, von dem das Produkt bezogen wurde, zurückgeben.
- c) Wird ein Fehler im Sinne von Ziffer 10 b) nicht innerhalb angemessener Frist durch eine Ersatzlieferung behoben, so kann der Lizenznehmer nach seiner Wahl die Herabsetzung des Erwerbspreises oder das Rückgängigmachen des Lizenzvertrages verlangen. Weitere Ansprüche gegen die Jäger Messtechnik GmbH entstehen nicht.
- d) Aus den vorstehend unter Punkt 1.2 genannten Gründen übernimmt die Jäger Messtechnik GmbH keine Haftung für die Fehlerfreiheit der *TiCoBasic*-Software. Insbesondere übernimmt die Jäger Messtechnik GmbH keine Gewähr dafür, dass die *TiCoBasic*-Software den Anforderungen und Zwecken des Lizenznehmers genügt oder mit anderen von ihm ausgewählten Programmen zusammenarbeitet. Die Verantwortung für die richtige Auswahl und die Folgen der Benutzung der *TiCoBasic*-Software, sowie der damit beabsichtigten oder erzielten Ergebnisse trägt der Lizenznehmer. Das gleiche gilt für das die *TiCoBasic*-Software begleitende zugehörige Schriftmaterial.
- e) Jäger Messtechnik GmbH haftet nicht für Schäden, es sei denn, dass ein Schaden durch Vorsatz oder grobe Fahrlässigkeit seitens der Jäger Messtechnik GmbH verursacht worden ist. Eine Haftung wegen eventuell von der Jäger Messtechnik GmbH zugesicherten Eigenschaften bleibt unberührt. Eine Haftung für Mangelfolgeschäden, die nicht von der Zusicherung umfasst sind, ist ausgeschlossen.
- f) Die Jäger Messtechnik GmbH übernimmt keine Haftung für Schäden durch Viren, die mit dem Datenträger übertragen werden. Der Lizenznehmer ist angehalten, die Datenträger auf Viren zu überprüfen, bevor er die *TiCoBasic*-Software auf seinem Computer installiert.

## 11. SCHLUSSBESTIMMUNGEN

Die Unwirksamkeit einzelner Bestimmungen berührt die Wirksamkeit des Lizenzvertrages im übrigen nicht.

Ergänzend zu den Bestimmungen dieses Lizenzvertrages gelten die allgemeinen Geschäftsbedingungen der Jäger Messtechnik GmbH.

### A.4 Kommandozeilen-Aufruf

Der *TiCoBasic*-Compiler kann nicht nur in der Bedienoberfläche aktiviert werden, sondern auch direkt aus Windows oder DOS aufgerufen werden (sogenannter „Kommandozeilen“-Aufruf). Der Compiler arbeitet in beiden Fällen auf gleiche Weise, kann also eine Quelltext-Datei kompilieren und daraus eine Binär- oder Library-Datei erzeugen.

Der Compiler-Aufruf wird nur ausgeführt, wenn Sie in *TiCoBasic* Ihren License key bereits eingegeben haben.



Beachten Sie die allgemeinen Hinweise zur Kommandozeile unter Windows auf Seite A-9.

#### A.4.1 Syntax

Es gibt Kommandozeilen-Aufrufe zum Erzeugen einer Binärdatei (Hauptoption `/M`) und zum Erzeugen einer Bibliothek (Hauptoption `/L`).

Über Schalter werden die Optionen für den Compiler eingestellt. Wenn ein Schalter nicht angegeben ist, wird die jeweilige Voreinstellung (Default) verwendet; wir empfehlen aber dennoch, alle Schalter anzugeben, um Unklarheiten zu vermeiden<sup>1</sup>.

Beim Erzeugen einer Binärdatei können gleichzeitig mehrere Quelldateien kompiliert werden. Daher wird unterschieden zwischen Optionen, die für alle Dateien gelten, und Optionen, die für jede Datei separat anzugeben sind (siehe Schalter `/PROCESS`).

Alternativ können Sie die Optionen für einen Aufruf in eine Datei, das `make-file`, schreiben und den Compiler mit der Hauptoption `/MAKE` aufrufen.

Schließlich gibt es die Hauptoptionen `/H` zum Aufruf eines kurzen Hilfetexts und `/VER` zur Anzeige der Compiler-Version.

Der Aufruf wird in einer einzelnen Zeile geschrieben. Achten Sie auf Großschreibung der Schalter.

---

1. Beispielsweise bleibt ein Aufruf mit allen Schaltern korrekt, auch wenn sich eine Default-Einstellung ändern sollte.

**Syntax**

```

TiCoBasicCompiler /M [/A"dest"] [/IP"path"]
[/LP"path"] [/Lx] [/Sx] [/P1]/PROCESS src.bas [/ET |
/EA | /EN | /EE /EEAx /EEMx /EEVx /EEOx] [/PNx] [/PH
| /PL] [/PDx] [/Ox] [/Vx]

TiCoBasicCompiler /L src.bas [/A"dest"] [/IP"path"]
[/LP"path"] [/Lx] [/Sx] [/P1] [/Ox]

TiCoBasicCompiler /MAKE"makefile"

TiCoBasicCompiler /H

TiCoBasicCompiler /VER

```

Optionale Angaben stehen in eckigen Klammern []. Das Zeichen | trennt Schalter, die sich gegenseitig ausschließen.

Dateinamen können ohne, mit relativem oder mit absolutem Pfadnamen angegeben werden. Das Basisverzeichnis für die beiden ersten Angaben ist das Arbeitsverzeichnis, aus dem heraus die Kommandozeile aufgerufen wird.

**Hauptschalter**

/M	Binärdatei mit der Endung <code>.TIn</code> erzeugen. n Prozessnummer; siehe Schalter /PNx.
/L	Library-Datei (Bibliothek) mit der Endung <code>.TLx</code> erzeugen. x Prozessortyp; siehe Schalter /Px.
/MAKE	Hauptschalter, Dateiname und weitere Schalter für einen Aufruf aus dem <code>makefile</code> entnehmen. Der Text im <code>makefile</code> kann über mehrere Zeilen verteilt geschrieben werden. Schalter außerhalb des <code>makefile</code> sind nicht erlaubt.
/H	Kurzen Hilfetext anzeigen.
/VER	Versionsnummer des Compilers anzeigen.



### Schalter

<code>src.bas</code>	Dateiname des zu kompilierenden Quelltextes; mit Dateieindung <code>.bas</code> angeben. Bei Hauptschalter <code>/M</code> nach <code>/PROCESS</code> angeben.  Compiler-Warnungen werden in die Datei <code>src.wrn</code> geschrieben, Fehlermeldungen in die Datei <code>src.err</code> .
<code>/A"dest"</code>	[Pfad und] Name der zu erzeugenden Datei <code>&lt;dest&gt;</code> <i>ohne</i> Dateieindung. Voreinstellung ist der Dateiname <code>src</code> .  Die Dateieindung <code>.TIn</code> (Binärdatei) oder <code>.TLx</code> (Library-Datei) wird automatisch angehängt.
<code>/IP"path"</code>	Verzeichnis, in dem nach Include-Dateien gesucht wird.  Die Einstellung überschreibt den Standard-Pfad von <i>TiCoBasic</i> und sollte mit Bedacht eingesetzt werden.
<code>/LP"path"</code>	Verzeichnis, in dem nach Library-Dateien gesucht wird.  Die Einstellung überschreibt den Standard-Pfad von <i>TiCoBasic</i> und sollte mit Bedacht eingesetzt werden.
<code>/Lx</code>	Sprache, in der Fehlermeldungen und Warnungen ausgegeben werden.  <code>/LE</code> Englisch. Default. <code>/LG</code> Deutsch
<code>/Sx</code>	Hardware einstellen, für die die Datei kompiliert wird:  <code>/SGII</code> Gold II <code>//SPII</code> Pro II; Default
<code>/Px</code>	Prozessortyp, für den die Datei kompiliert wird:  <code>/P1</code> <i>TiCo</i> -Prozessor 1
<code>/PROCESS</code>	Schlüsselwort für die Optionen des nächsten Quelltexts. Muss für jeden Quelltext wiederholt werden.  Nur gemeinsam mit dem Hauptschalter <code>/M</code> .
<code>/ET</code>	Zeitgesteuerten Prozess erzeugen, siehe Kapitel 6.1.1 auf Seite 107. Default.  Schließt <code>/EE</code> und <code>/EN</code> aus.
<code>/EE</code>	Extern gesteuerten Prozess erzeugen, siehe Kapitel 6.1.2 auf Seite 108; erfordert die Optionen <code>/EEA</code> , <code>/EEM</code> , <code>/EEV</code> , <code>/EEO</code> .  Schließt <code>/ET</code> und <code>/EN</code> aus.

/EEAn	Hardware-Adresse <i>n</i> (dezimal), die für das Event-Signal ausgewertet wird.
/EEMn	Maskenwert <i>n</i> (dezimal), mit dem die Adresse UND-verknüpft wird.
/EEVn	Vergleichswert <i>n</i> (dezimal).
/EEOx	Vergleichsoperator <i>x</i> : 1: < kleiner 2: = gleich 3: <= kleiner gleich 4: > größer 6: >= größer gleich 8: <> ungleich
/EN	Ungesteuerten Prozess erzeugen, siehe Kapitel 6.1.3 auf Seite 109. Schließt /EE und /ET aus.
/PNx	Nummer <i>x</i> (1...4) des Prozesses. Default: 1.
/PH	Prozess mit hoher Priorität erzeugen. Default-Einstellung. Siehe auch Kapitel 6.1 auf Seite 107.
/PL	Prozess mit niedriger Priorität erzeugen (nur für zeitgesteuerten Prozess). Siehe auch Kapitel 6.1 auf Seite 107.
/PDx	Zykluszeit (Processdelay) des Prozesses auf <i>x</i> einstellen. Default: 3000. Siehe auch Kapitel 6.2.1 auf Seite 109.
/Ox	Optimierungsstufe <i>x</i> (0, 1, 2) für die Kompilierung einstellen, siehe auch Dialogfenster „Process Options“ (Seite 49). /O0      Optimierungsstufe 0 (=keine Optimierung) /O1      Optimierungsstufe 1 (Default) /O2      Optimierungsstufe 2
/Vx	Version <i>x</i> des Prozesses einstellen, siehe Dialogfenster „Process Options“ (Seite 49). Default: 1.

#### A.4.2 Bemerkungen

Die Reihenfolge der Schalter ist beliebig. Bei den Schaltern wird Groß-/Kleinschreibung unterschieden, bei Dateinamen nicht.

Wenn der Schalter `/A` nicht verwendet wird, wird die erzeugte Binärdatei oder Library-Datei in dem Verzeichnis gespeichert, in dem sich der Quelltext befindet.

Treten während der Kompilierung Warnungen oder Fehler auf, werden sie in die Dateien `src.wrn` und `src.err` geschrieben. Die Fehlermeldungen sind identisch mit denjenigen, die *TiCoBasic* im Infofenster (siehe Kapitel 3.10.1) ausgeben würde.

Die Dateien werden in dem Verzeichnis gespeichert, wo sich der Quelltext `src.bas` befindet. Wenn Sie den Schalter `/A` verwenden, werden die Dateien in dem Verzeichnis gespeichert, wo die Binär- oder Library-Datei erzeugt wird.

Wir empfehlen, vor dem Kompilieren Dateien mit Warn- und Fehlermeldungen zu löschen, damit Sie nach dem Kompiliervorgang einfach prüfen können, ob dieser fehlerlos abgelaufen ist.

### A.4.3 Beispiele

```
C:\ADwin\TiCoBasic\TiCoBasiccompiler.exe /L  
Z:\Myfiles\test.bas
```



Diese Kommandozeile kompiliert den Quelltext `<test.bas>` und erzeugt die Library-Datei `<test.TL9>` im Verzeichnis `<Z:\Myfiles\>`.

Da nichts anderes angegeben ist, werden alle Standardeinstellungen verwendet:

- erzeugte Datei im Verzeichnis der Quelldatei speichern
- englische Warnungen und Fehlermeldungen.
- Hardware: *ADwin-Pro II*.
- *TiCo*-Prozessor 1.
- Optimierungsstufe 1.

Wenn Sie sich beim Aufruf im Verzeichnis `<C:\ADwin\TiCoBasic>` befinden, können Sie obige Zeile kürzer schreiben:

```
TiCoBasiccompiler.exe /L Z:\Myfiles\test.bas
```

Die kürzeste Aufruf-Variante ergibt sich, wenn auch der Quelltext im Verzeichnis `<C:\ADwin\TiCoBasic>` liegt:

```
TiCoBasiccompiler /L test.bas
```

Wir empfehlen in jedem Fall – speziell für eine Automatisierung des Aufrufs – die vollständige Variante:

```
TiCoBasiccompiler /L test.bas /A"test" /LE /SPII /P1 /O1
```



```
TiCoBasicCompiler /M /LE /SGII /P1 /PROCESS bas_dmo6f.bas  
/ET /PN3 /PH /O1
```

Kompiliert die Demo-Datei <bas\_dmo6f.bas> in eine Binär-Datei für ein Gold II-System mit TiCo-Prozessor; der Prozess ist zeitgesteuert, hat die Nummer 3 und hohe Priorität.



```
TiCoBasicCompiler /M /A"Y:\somewhere\your_file" /LE /SGII  
/P1  
  
/PROCESS C:\user\my_file.bas /ET /PN3 /PH /O1
```

Die Binärdatei heißt nun <your\_file.TL1> und befindet sich im Verzeichnis <Y:\somewhere>; der Prozess ist zeitgesteuert, hat die Nummer 3 und hohe Priorität.

#### A.4.4 Kommandozeile unter Windows

Der Begriff und die Funktionalität „Kommandozeilen-Aufruf“ stammen noch aus der DOS-Zeit, in der man Befehle an das Betriebssystem DOS in einer Kommandozeile eingeben musste. Solche Eingaben sind auch unter Windows nach wie vor möglich und eignen sich z.B. für die Automatisierung von Abläufen.

Sie haben mehrere Möglichkeiten, Kommandozeilen unter Windows einzugeben:

- Öffnen Sie unter Windows ein MS-DOS-Fenster (Windows Start-Menü, Verzeichnis *Programs / Accessories*). Jede Eingabe ist hier eine Kommandozeile.



Der Compiler-Aufruf erfordert in jedem Fall die Windows-Umgebung. Der Aufruf funktioniert also nur aus diesem Windows-Fenster, nicht aus der originären DOS-Ebene.

- Wählen Sie im Start-Menü den Menüeintrag „Run“ und geben Sie im Eingabefenster eine Kommandozeile ein.
- Legen Sie für öfter benötigte Kommandozeilen-Aufrufe jeweils ein Icon auf dem Desktop an. Bei der Erstellung eines Icon geben Sie eine Kommandozeile direkt ein.

Ein oder mehrere Kommandozeilen-Aufrufe können in einer Batch-Datei `<*.bat>` zusammengefasst werden, z.B. um mehrere Quelltexte eines Projekts mit nur einem Aufruf zu kompilieren.

Bei jedem Aufruf müssen Sie die jeweils passenden Schalter und Parameter übergeben.





**A.5 Befehle für ADwin-Gold II**

Im folgenden sind die in ADbasic verfügbaren Befehle für ADwin-Prozessoren aufgeführt. Befehle zur Steuerung der Ein- und Ausgänge finden Sie in der Hardware-Dokumentation.

**Symbole**

< = > (Vergleich)  
 + (Addition)  
 - (Subtraktion)  
 \* (Multiplikation)  
 / (Division)  
 ^ (Potenz)  
 = (Zuweisung)  
 : Doppelpunkt  
 #Define  
 #If ... Then ... {#Else ...} #EndIf  
 #Include  
 #..., Präprozessor-Anweisung

**A**

Absl  
 And

**D**

DATA\_n  
 Dec  
 Dim  
 Do ... Until

**E**

End  
 Event:

**F**

Finish:  
 For ... To ... {Step ...} Next  
 Function ... EndFunction

**G-L**

If ... Then ... {Else ...} EndIf  
 Import  
 In  
 Inc  
 Init:  
 Lib\_Function ... Lib\_EndFunction  
 Lib\_Sub ... Lib\_EndSub

**M-Q**

Max\_Long  
 Min\_Long  
 NOP  
 NOPS  
 Not  
 Or  
 Out  
 Processdelay  
 ProcessN\_Running

**R**

Read\_Timer  
 Refresh\_RingBuffer  
 Rem  
 Ringbuffer  
 RingBuffer\_Clear  
 RingBuffer\_Empty

**S**

SelectCase  
 Shift\_Left  
 Shift\_Right  
 Sleep  
 Sub ... EndSub



**T-Z**

XOr

## A.6 Befehle für ADwin-Pro

Im folgenden sind die in ADbasic verfügbaren Befehle für ADwin-Prozessoren aufgeführt. Befehle zur Steuerung der Ein- und Ausgänge finden Sie in der Hardware-Dokumentation.

Modul

### Symbole

< = > (Vergleich)  
 + (Addition)  
 - (Subtraktion)  
 \* (Multiplikation)  
 / (Division)  
 ^ (Potenz)  
 = (Zuweisung)  
 : Doppelpunkt  
 #Define  
 #If ... Then ... {#Else ...} #EndIf  
 #Include  
 #..., Präprozessor-Anweisung

### A

Absl  
 And ,

### D

DATA\_n  
 Dec  
 Dim  
 Do ... Until

### E

End  
 Event:

### F

Finish:  
 For ... To ... {Step ...} Next  
 Function ... EndFunction

### G-L

If ... Then ... {Else ...} EndIf  
 Import  
 In  
 Inc  
 Init:  
 Lib\_Function ... Lib\_EndFunction  
 Lib\_Sub ... Lib\_EndSub

### M-Q

Max\_Long  
 Min\_Long  
 NOP  
 NOPS  
 Not  
 Or ,  
 Out  
 Processdelay  
 ProcessN\_Running

## R

Read\_Timer  
Refresh\_RingBuffer  
Rem  
Ringbuffer  
RingBuffer\_Clear  
RingBuffer\_Empty  
RingBuffer\_Full

## S

SelectCase  
Shift\_Left  
Shift\_Right  
Sleep  
Sub ... EndSub

## T-Z

XOr



## A.7 Index

### Symbole

- · 120
- # · 124
- #Define · 133
- #Else · 148
- #EndIf · 148
- #If · 148
- #Include · 154
- \* · 121
- + · 119
- / · 122
- : · 125
- < = > · 127
- = · 126
- ^ · 123
- ' (Rem) · 177

### Zahlen

150h, siehe Device No.

### A

- Abbruch siehe Prozess beenden
- Absl · 128
- Absolutwert
  - Ganze Zahlen · 128
- ActiveX
  - Kommunikation zum ADwin-System · 113
- Add Open Files to Project · 59
- Add to Project · 18, 59
- Addition · 119
- ADtools
  - Leiste einstellen · 56
- ADtools · 71
- ADWIN\_GOLDII · 148
- ADWIN\_PROII · 148
- ADWIN\_SYSTEM · 148

- And · 129
- Anhalten
  - TiCo-Prozessor · 12
- Anhalten siehe Prozess beenden
- Anmerkungen siehe Kommentar
- Anzeige
  - aktuelle Informationen · 15
  - Auslastung: CPU, PM, DM, DX · 66
  - Befehlsparameter · 37
  - Prozessoptionen · 13
  - ToDo-Liste · 68

### Arithmetische Funktionen

- · 120
- \* · 121
- + · 119
- / · 122
- ^ · 123
- Dec · 132
- Inc · 153
- Arrays, siehe Felder
- (Dim) As · 135
- ASCII-Zeichensatz · 4
- (Dim ...) At · 135

### Auslastung

- Anzeige · 66
- Definition · 110
- auswerten
  - Operatoren · 93
- Autoindent · 53
- Automatisch vervollständigen, Befehle und Variablen · 35
- Automatisches Einrücken · 22
- Automatisches Formatieren · 21
- AutoSave · 45
- Autostart · 45

### B

- Bearbeitungszeit messen · 99
- Bedienoberfläche · 12

- Bedingter Sprung
    - If ... Then · 146
    - SelectCase · 188
  - Befehl
    - Automatisch
      - vollständigen · 35
    - Deklaration anzeigen · 38
    - Übergabeparameter anzeigen · 37
    - zur Deklaration springen · 33
  - Befehlsreferenz · 117
  - Befehls-Separator (:) · 125
  - Befehlszeile
    - Groß-/Kleinschreibung · 73
    - max. Zeilenlänge · 73
    - Zeilenlänge
      - mit #Include · 154
  - benutzerdefinierte Befehle und Variablen
    - Übersicht · 75
  - Berechnungsausdrücke · 93
    - auswerten · 93
    - symbolische Namen · 75
  - Betriebssystem
    - laden, siehe Initialisieren
    - Verzeichnis einstellen · 56
  - Bibliothek
    - Allgemeines · 96
    - Ein-/ausfalten · 23
    - Einbinden · 150
    - Erzeugen
      - aus Kommandozeile · 9
      - aus TiCoBasic · 46
    - erzeugen · 45
    - Funktion · 157
    - Position im Programm · 76
    - Rekursion verboten · 96
    - Unterprogramm · 161
    - Verzeichnis einstellen · 56
  - Binärdatei
    - Erzeugen
      - aus Kommandozeile · 9
      - erzeugen · 45
      - aus TiCoBasic · 45
      - siehe auch Bibliothek
    - übertragen zum TiCo-Prozessor · 39
  - Binäre Schreibweise · 79
  - Bits verschieben
    - nach links · 191
    - nach rechts · 192
  - Bookmark · 33
  - Bootloader · 105
    - Menüeintrag · 57
    - programmieren · 40
  - BTL-Datei: Verzeichnis einstellen · 56
  - Busy-Anzeige · 66
- ## C
- Clear Parameter Scan · 39
  - Code size · 67
  - code snippets · 37
  - Comment Block · 22
  - Compiler
    - Aufruf · 45
    - AutoSave · 45
    - Compilermeldung, Fehler / Status · 67
    - Fehlermeldung · 45
    - Kommandozeilen-Aufruf · 9
    - Optionen einstellen · 46
  - Compiler-Anweisungen · 124
    - #Define · 133
    - #If ... Then · 148
    - #Include · 154
  - Control block · 18
  - Controlblock · 32
  - Cursor-Position · 66

## D

- DATA\_n
  - dimensionieren · 135
  - Übersicht · 131
- Data\_n
  - Globale Felder · 80
- Dateiname
  - Bibliothek · 46
  - Binärdatei · 45
- Datenaustausch
  - zwischen ADwin CPU und TiCo-Prozessor · 114
  - zwischen PC und TiCo-Prozessor · 113
  - zwischen Prozessen · 112
- Datenspeicher
  - siehe auch Speicher
  - Übersicht, intern, extern · 84
- Datenspeicher, intern (DM) · 84
- Datenstrukturen
  - Globale Felder · 80
  - Globale Variablen · 79
  - Lokale Variablen und Felder · 82
  - Ringbuffer · 85
  - Übersicht · 77
- Datentypen
  - Übersicht · 78
- Datenverlust
  - beim Initialisieren · 12
  - beim Rücksetzen · 12
  - Ringbuffer · 86
- Datenwort: Nummerierung von
  - Bits · 2
- Dec · 132
- Declaration Info · 38
- Declarations · 70
- DEFINE siehe #Define

- Deklaration
  - alle anzeigen · 38, 70
  - anzeigen · 38
  - siehe Dimensionierung
  - zur ~ springen · 33
- Dekrementieren · 132
- Demo-Modus · 10
- Device No.
  - Definition · 115
  - einstellen · 47
- Dezimale Schreibweise · 79
- Dim · 135
- Dimensionierung
  - Befehl Dim · 135
  - Position im Programm · 75
  - Speicherbereich · 83
- Directory siehe Verzeichnis
- Disable Trace · 18
- Division
  - durch 2 · 192
  - einfache · 122
- DM, siehe Datenspeicher, intern
- DM\_Local
  - Dim · 135
- Do ... Until · 138
- DRAM\_Extern
  - Dim · 135
- Druckeinstellungen · 55
- DX, siehe Externer Speicher

## E

- Editor
  - General · 53
  - Print Settings · 55
  - Syntax Colors · 54
- Editor-Leiste · 20
- Ein-/ausfalten, Textbereich · 23
- Einbinden
  - Include-Datei · 154
  - Library · 150
- Einrücken · 22

- Else (If ...) · 146
- Enable Trace · 18
- End · 139
- EndFunction · 143
- EndIf (If ...) · 146
- EndSelect (SelectCase ...) · 188
- EndSub · 194
- Entwicklungsumgebung
  - Leisten und Fenster · 12
  - starten · 9
  - Tastaturkürzel · 1
- Ersetzen
  - Beispiele · 29
  - Reguläre Ausdrücke · 30
  - Text · 25
- Event
  - extern gesteuert · 108
  - externes Signal · 105
  - nicht gesteuert · 109
  - Signalquelle einstellen · 51
  - verlorenes Signal
    - ein Prozess zeitgesteuert · 111
    - extern gesteuerter Prozess · 111
  - Zeitdifferenz messen · 100
  - zeitgesteuert · 107
- Event: · 140
- Exklusiv-ODER-Verknüpfung · 196
- Exponential-Schreibweise · 79
- Externer Speicher (DX) · 84
- externer Speicher (SDRAM) · 84
- Externer Speicher (SX) · 84
- externes Event-Signal · 105
- Extremwert
  - Maximum Ganze Zahlen · 165
  - Minimum Ganze Zahlen · 166
- F**
  - F1: Hilfe aufrufen · 17
  - Falten, Textbereiche · 23
  - Farbe einstellen · 54
  - Farbige Darstellung · 21
  - Fehler
    - durch Cut&Paste erzeugt · 44
    - geringere Optimierung einstellen · 52
  - Fehlermeldung Compiler · 67
  - Felder
    - DATA\_n · 131
    - globale · 80
      - erstes Element · 81
      - verwendete anzeigen · 39
    - initialisieren · 75
    - lokale · 82
      - erstes Element · 83
    - Ringbuffer · 178
    - Speicherbereich festlegen · 83
    - Übersicht · 77
  - Fenster
    - Compiler Options · 46
    - Declarations · 70
    - Global Variables · 69
    - Info-Bereich · 66
    - Info-Fenster · 67
    - Parameter · 61
    - Process Options · 49
    - Projekt · 59
    - Quelltext-Informationen · 13
    - Quelltext-Statusleiste · 13
    - Statusleiste · 66
    - ToDo-Liste · 68
    - Toolbox · 59
    - Übersicht · 12
  - FIFO · 85
  - Finden · 25
    - Beispiele · 28
    - Deklaration von Befehl/Variable · 33
    - Reguläre Ausdrücke · 30
  - Finish: · 141
  - Font einstellen · 54
  - For ... Next · 142



formatieren · 21

Function · 143

Funktion

    Allgemeines zu Bibliotheken · 96

    Allgemeines zu Makros · 95

    Bibliothek (Lib\_Function) · 157

    Makro · 143

    Position im Programm · 76

## G

Ganze Zahlen

    Wertebereich · 78

Gerätenummer, Definition · 115

Get\_Par · 198

Get\_Par\_Block · 200

Get\_TiCo\_RingBuffer · 202

Get\_TiCo\_Status · 205

GetData\_Long · 206

gleich = · 127

Global Variables · 69

Globaldelay · 173

globale Felder, siehe Felder, globale

globale Variablen, siehe Variablen, globale

Goto Line · 33

Groß-/Kleinschreibung · 16

größer als >, >= · 127

## H

Halt siehe Prozess beenden

Hardware-Zugriff

    Lesen · 152

    Schreiben · 172

Header · 55

Hexadezimale Schreibweise · 79

Hilfe

    aufrufen · 17

Hilfsprogramme (ADtools<>)> · 71

## I

If · 146

    siehe auch #If · 148

Import · 150

In · 152

Inc · 153

Include-Datei

    Allgemeines · 95

    einbinden · 154

    Verzeichnis einstellen · 56

Indent · 22

Indent TiCoBasic sections · 53

Info-Bereich · 66

Info-Fenster · 67

Init: · 156

Initialisieren · 11

Initialisierung · 75

Inkrementieren · 153

Interner Speicher

    Datenspeicher (DM) · 84

    Gesamt (SRAM) · 84

    Programm (PM) · 84

## J

Jump to Declaration · 33

## K

kleiner als <, <= · 127

Kommandozeilen-Aufruf · 9

Kommentar

    Syntax · 177

    Zeilen in ~ ändern · 22

Kommunikation

    zwischen ADwin CPU und TiCo-  
    Prozessor · 114

    zwischen PC und TiCo-  
    Prozessor · 113

    zwischen Prozessen · 112

kompilieren siehe Compiler

Konstante · 75

- Kontextmenü · 18
- Projektfenster · 59
- Kontrollstrukturen · 94

**L**

- Language · 56
- Lib\_EndFunction · 157
- Lib\_EndSub · 161
- Lib\_Function · 157
- Lib\_Sub · 161
- Library
  - Funktion · 157
  - Import · 150
  - siehe auch Bibliothek
  - siehe Bibliothek
  - Unterprogramm · 161
- Lib-Verzeichnis einstellen · 56
- License key eingeben · 10
- Lizenzvertrag · 5
- Load Bin File · 57
- Logische Funktionen
  - And · 129
  - Not · 169
  - Or · 170
  - Shift\_Left · 191
  - Shift\_Right · 192
  - XOr · 196
- Long, siehe Ganze Zahlen

**M**

- Make Bin File, Make Lib File · 45
- Makro
  - Allgemeines · 95
  - Ein-/ausfalten · 23
  - Funktion · 143
  - Position im Programm · 76
- Mark Controlblock · 32
- Max\_Long · 165

- Maximale Zeilenlänge
  - mit #Include · 154
- Maximum
  - Ganze Zahlen · 165
- Menü
  - auswählen · 13
  - Build · 45
  - Edit · 44
  - File · 43
  - Help · 58
  - Leiste · 42
  - Options · 46
  - Tools · 57
  - View · 44
  - Window · 57
- Menüleiste · 42
- Messwertverlauf darstellen · 71
- Metazeichen · 30
- Min\_Long · 166
- Minimum
  - Ganze Zahlen · 166
- Multiplikation
  - einfache · 121
  - mit 2 · 191

**N**

- Namen, lokale Variablen · 82
- negatives Vorzeichen · 94
- Neues in TiCoBasic · 5
- Next (For ...) · 142
- NICHT · 169
- None: ohne Event · 109
- NOP · 167
- Not · 169

**O**

- ODER-Verknüpfung · 170
- ohne Event · 109

- Operatoren
  - And · 129
  - auswerten · 93
  - negatives Vorzeichen · 94
  - Or · 170
  - Priorität · 93
  - XOr · 196
- Optimierung
  - Allgemein · 99
  - Bearbeitungszeit messen · 99
  - Konstanten statt Variablen · 100
  - Polynom schneller
    - berechnen · 123
  - Registerzugriff · 100
  - schneller messen · 101
  - Speicherzugriff · 103
  - Wartezeit einstellen · 101
  - Wartezeit nutzen · 101
- Optionen einstellen
  - ADtools · 56
  - Allgemein (Settings) · 53
  - Compiler · 46
  - Drucken · 55
  - Editor · 53
  - Prozess · 49
  - Sprache · 56
  - strukturierte
    - Befehlsdarstellung · 54
  - Verzeichnisse · 56
- Or · 170
- Ordner siehe Verzeichnis
- Out · 172
- Outdent · 22
- P**
  - P2\_Get\_Par · 246
  - P2\_Get\_Par\_Block · 248
  - P2\_Get\_TiCo\_Bootloader\_Status · 250
  - P2\_Get\_TiCo\_RingBuffer · 251
  - P2\_Get\_TiCo\_Status · 254
  - P2\_GetData\_Long · 256
  - P2\_Process\_Status · 259
  - P2\_RingBuffer\_Empty · 261
  - P2\_RingBuffer\_Full · 262
  - P2\_Set\_Par · 263
  - P2\_Set\_Par\_Block · 265
  - P2\_Set\_TiCo\_RingBuffer · 267
  - P2\_SetData\_Long · 270
  - P2\_TDrv\_Init · 273
  - P2\_TiCo\_Flash · 277
  - P2\_TiCo\_Get\_Processdelay · 275
  - P2\_TiCo\_Load · 280
  - P2\_TiCo\_Reset · 283
  - P2\_TiCo\_Set\_Processdelay · 285
  - P2\_TiCo\_Start · 287
  - P2\_TiCo\_Start\_Process · 288
  - P2\_TiCo\_Stop · 290
  - P2\_TiCo\_Stop\_Process · 291
  - P2\_Workload · 293
  - Par\_n, globale Variablen · 79
  - Parameter Scan · 39
  - Parameter, siehe Variablen, globale
  - Parameterfenster · 61
  - Parse and Indent · 53
  - PM, siehe Programmspeicher
  - Polynom, schneller
    - berechnen · 123
  - Potenz · 123
    - in Polynom ersetzen · 123
  - Präprozessor-Anweisungen · 124
  - Präprozessor-Anweisungen
    - #Define · 133
    - #If ... Then · 148
    - #Include · 154
  - Print layout · 55
  - Priorität
    - Operatoren · 93
  - Probleme
    - langsamer Editor · 53
  - Process\_Running · 175
  - Process\_Status · 209

- Processdelay
    - Syntax · 173
    - Zeitverhalten · 109
  - ProcessN\_Running · 175
  - Processor · 148
  - Programm verbessern, siehe Optimierung
  - Programmabschnitte
    - Event: · 75
    - Finish: · 75
    - Init: · 75
    - Übersicht · 75
  - Programmspeicher (PM) · 84
  - Programmstruktur
    - Übersicht · 94
    - Bibliothek
      - Lib\_Function · 157
      - Lib\_Sub · 161
      - Übersicht · 96
    - Ein-/ausfalten · 23
    - Include-Datei · 95
    - Kommentar Rem · 177
    - Makros
      - Funktion Function · 143
      - Übersicht · 95
      - Unterprogramm Sub · 194
    - Schleife
      - Do ... Until · 138
      - For ... Next · 142
    - Sprung
      - If ... Then · 146
      - SelectCase · 188
  - Projektfenster · 59
  - Projektverwaltung
    - Allgemeines · 41
    - Fenster · 59
    - verwendete Variablen anzeigen · 39
  - Prozess
    - Bearbeitungszeit · 110
    - beenden
      - sich selbst (in Event:) · 139
    - Betriebszustände beim Zeitverhalten · 111
    - extern gesteuert · 108
    - Kommunikation zwischen ~en · 112
    - nicht gesteuert · 109
    - Optionen
      - einstellen · 49
    - Optionen, Anzeige · 13
    - Status ermitteln · 175
    - zeitgesteuert
      - Priorität hoch · 107
      - Priorität niedrig · 107
    - Zeitverhalten · 109
    - Zyklus, siehe Prozesszyklus
  - Prozess optimieren, siehe Optimierung
  - Processor, siehe Processor · 148
  - Prozess-Steuerung
    - End · 139
    - ProcessN\_Running · 175
  - Prozesszyklus
    - Aufruf
      - mit Event-Signal · 105
      - Zeiteinheit · 110
  - Punkt vor Strich, siehe Operatoren
- Q**
- Quelltext
    - erstellen · 16
    - farbig darstellen · 21
    - formatieren · 21
    - im Projekt verwenden · 59
    - ToDo-Liste · 68
  - Quelltext-Statusleiste · 13

## R

Read\_Timer · 176  
 Rechenzeit sparen  
     Konstanten statt Variablen · 100  
     Registerzugriff · 100  
     schneller messen · 101  
     Wartezeit einstellen · 101  
     Wartezeit nutzen · 101  
 Refresh\_RingBuffer · 180  
 Registerzugriff · 100  
 Reguläre Ausdrücke · 30  
 Rekursion bei Bibliothek · 96  
 Rem · 177  
 Reset  
     TiCo-Prozessor · 12  
 Ringbuffer  
     Aufbau der Datenstruktur · 85  
     Datenverlust · 86  
     dimensionieren · 135  
     Elementanzahl prüfen · 87  
     Übersicht · 178  
 RingBuffer\_Clear · 182  
 RingBuffer\_Empty · 184, 211  
 Ringbuffer\_For\_Read · 178  
 Ringbuffer\_For\_Write · 178  
 RingBuffer\_Full · 186, 212  
 Ringspeicher · 85

## S

Save All Files of Project · 59  
 Schleife, Do ... Until · 138  
 Schreibweise von Zahlen · 79  
 Schriftart einstellen · 54  
 SDRAM, siehe Externer Speicher  
 SelectCase · 188  
 Separator : · 125  
 Set\_Par · 213  
 Set\_Par\_Block · 215  
 Set\_TiCo\_RingBuffer · 217

SetData\_Long · 220  
 Shift\_Left · 191  
 Shift\_Right · 192  
 Short-Cuts · 1  
 Show Declarations · 38  
 Show line numbers · 53  
 Sleep · 193  
 Smart format · 21  
 snippets · 37  
 Speicher  
     Auslastung · 66  
     Bedarf bestimmen · 67  
     Bereich festlegen · 83  
     Bereiche (PM, DM, DX) · 84  
     siehe auch Datenspeicher  
     Zusatzbedarf durch  
         Bibliotheken · 96  
         Makros · 95  
 springen zu Zeile · 33  
 Sprung, bedingter  
     If ... Then · 146  
     SelectCase · 188  
 SRAM, siehe Interner Speicher, Gesamt  
 Stack size · 67  
 Starten, TiCoBasic · 9  
 Statusleiste · 66  
 Statusmeldung Compiler · 67  
 Step (For ...) · 142  
 Stopp siehe Prozess beenden  
 Stoppen  
     TiCo-Prozessor · 12  
 Strukturieren  
     Ein-/ausfalten · 23  
     Farbig Darstellung · 21  
     Programmabschnitte · 94  
     Zeilen einrücken · 22  
 Sub · 194  
 Subtraktion · 120

Suchen  
  Beispiele · 28  
  Deklaration von  
    Befehl/Variable · 33  
  Reguläre Ausdrücke · 30  
  schnell · 24  
  Text · 25  
SX, siehe Externer Speicher  
symbolische Namen · 75  
Syntax  
  Colors · 54  
  Farbige Darstellung · 21  
Systemvariablen  
  Processdelay · 173  
  ProcessN\_Running · 175  
  Übersicht · 81

## T

T10 · 148  
T11  
  Bedingung mit #If · 148  
T9 · 148  
Tabsize · 53  
Tabulator  
  Schrittweite einstellen · 53  
Tastatur  
  Anzeige von Einstellungen · 66  
  Kürzel · 1  
TDrv\_Init · 223  
Terminierung siehe Prozess beenden  
Text formatieren · 21  
Text schnell suchen · 24  
text snippets · 37  
Text suchen und ersetzen · 25  
Textbausteine einfügen · 37  
Textbereich ein-/ausfallen · 23  
Textmarke · 33  
Then (If ...) · 146  
TiCo\_Flash · 225  
TiCo\_Get\_Processdelay · 227

TiCo\_Load · 229  
TiCo\_Reset · 231  
TiCo\_Reset\_Mode · 233  
TiCo\_Set\_Processdelay · 234  
TiCo\_Start · 236  
TiCo\_Start\_Process · 237  
TiCo\_Stop · 239  
TiCo\_Stop\_Process · 240  
TiCo-11 · 148  
TiCoBasic  
  Demo-Modus · 10  
  Lizenzvertrag · 5  
  starten · 9  
  Verzeichnisse einstellen · 56  
TiCoBasic: Unterschiede zu  
  ADbasic · 5  
TiCoBasicCompiler · 9  
TiCo-Bootloader  
  Menüeintrag · 57  
  programmieren · 40  
TiCo-Prozessor  
  Reset / Stopp · 12  
Timer siehe Zähler  
Timer-Event · 105  
To (For ...) · 142  
ToDo-Liste · 68  
Toolbox · 59  
Tools  
  Binärdatei laden · 57  
  Bootloader · 57  
  TGraphTiCo aufrufen · 57

## U

Übergabeparameter anzeigen · 37  
Überlastung des Prozessors · 110  
umsteigen auf TiCoBasic · 5  
Uncomment Block · 22  
UND-Verknüpfung · 129  
ungleich <> · 127  
Unmark Controlblock · 32

Unterprogramm  
 Allgemeines zu Bibliotheken · 96  
 Allgemeines zu Makros · 95  
 Bibliothek (Lib\_Sub) · 161  
 Makro (Sub) · 194  
 Position im Programm · 76  
 Until (Do ...) · 138

## V

Variablen  
 Automatisch  
   vollständigen · 35  
 Deklaration anzeigen · 38  
 Deklaration finden · 33  
 globale · 79  
   Anzeige · 61  
   verwendete anzeigen · 39  
   Werte hexadezimal  
     anzeigen · 62  
 initialisieren · 75  
 Initialisierung · 12  
 lokale · 82  
   erlaubte Zeichen im  
     Namen · 82  
   Länge des Namens · 82  
   Speicherbereich festlegen · 83  
 symbolische Namen · 75  
 Übersicht · 77  
 vordefinierte Namen · 77  
 siehe auch Systemvariablen  
 Vergleich  
   < = > · 127  
 Verzeichnisse einstellen · 56

## W

Warten  
 Prozessor: NOP · 167  
 Sleep · 193  
 Wartezeit genau einstellen · 101

Werkzeuge (ADtools<>) · 71  
 Werkzeugleiste · 13  
 Wertebereich · 78  
 Workload · 242  
 Workspace size · 67

## X

XOr · 196

## Z

Zahlenwerte  
   Schreibweise · 79  
 Zähler  
   auslesen · 176  
   interner, Taktzyklus · 110  
 Zeile, zu ~ springen · 33  
 Zeilen einrücken · 22  
 Zeilen formatieren · 21  
 Zeilenlänge  
   max. Länge · 73  
 Zeilenlänge.  
   max. mit #Include · 154  
 Zeilennummern · 53  
 Zeit  
   Zykluszeit · 109  
 Zeitdifferenz messen · 99  
 Zeitoptimierung, siehe Optimierung  
 Zeitverhalten  
   Betriebszustände  
     allgemein · 111  
     ein Prozess zeitgesteuert · 111  
     extern gesteuerter  
       Prozess · 111  
 zuweisen, Zahlen · 79  
 Zuweisung (=) · 126





<b>Symbole</b>		FFT_Calc_DM	293	Or	226
< = > (Vergleich)	144	FFT_Calc_DX	295	<b>P</b>	
+ (Addition)	133	FFT_Init	290	P1_Sleep	228
+ (String-Addition)	134	FFT_Mag	285	P2_Sleep	230
- (Subtraktion)	136	FFT_Mag_Scale	289	Peek	232
* (Multiplikation)	137	FFT_Phase	287	Poke	233
/ (Division)	138	FFT_Scale	283	Processdelay	234
^ (Potenz)	139	FIFO	173	ProcessN_Running	238
= (Zuweisung)	143	FIFO_Clear	175	Process_Error	237
: Doppelpunkt	142	FIFO_Empty	177	<b>R</b>	
" " (String)	258	FIFO_Full	178	Read_Timer	239
#Define	162	Finish:	179	Rem	240
#If ... Then ... {#Else ...}		Flo40ToStr	183	Reset_Event	241
#EndIf	192	FloToStr	181	Restart_Process	242
#Include	197	For ... To ... {Step ...}		<b>S</b>	
#..., Präprozessor-Anweisung	141	Next	185	SelectCase	243
<b>A-C</b>		Function ... EndFunction	187	Shift_Left	246
AbsF	145	<b>G-J</b>		Shift_Right	247
AbsI	146	If ... Then ... {Else ...} En-		Sin	249
And	147	dlf	190	Sleep	250
ArcCos	149	Import	194	Sqrt	252
ArcSin	150	Inc	196	Start_Process	253
ArcTan	151	Init:	199	Start_Process_Delayed	254
Asc	152	IO_Sleep	201	Stop_Process	256
Cast_FloatToLong	153	<b>K-L</b>		" " (String)	258
Cast_LongToFloat	154	Lib_Function ... Lib_End-		StrComp	260
Chr	155	Function	203	StrLeft	261
Cos	156	Lib_Sub ... Lib_EndSub	208	StrLen	263
CPU_Sleep	157	LN	212	StrMid	264
<b>D</b>		LngToStr	213	StrRight	266
DATA_n	159	Log	215	Sub ... EndSub	268
Dec	161	LowInit:	216	<b>T-Z</b>	
Dim	164	<b>M-O</b>		Tan	271
Do ... Until	167	Max_Float	218	Trace_Mode_Pause	272
<b>E-F</b>		Max_Long	220	Trace_Mode_Resume	273
End	168	Min_Float	219	ValF	274
Event:	169	Min_Long	221	Vall	276
Exit	171	Mod	298	XOr	278
Exp	172	NOP	224		
FFT	281	Not	225		
FFT_Calc	291				

