

# ***ADbasic***

**Echtzeit-Entwicklungstool für  
*ADwin*-Systeme**

***ADbasic* Version 5.00**

**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 Neues in <i>ADbasic 5</i> . . . . .	5
3 Entwicklungsumgebung . . . . .	9
3.1 Grundlegende Schritte . . . . .	9
3.1.1 Entwicklungsumgebung erstmals starten . . . . .	9
3.1.2 Lizenzen für <i>ADbasic</i> prüfen und ändern . . . . .	10
3.1.3 <i>ADwin</i> -Betriebssystem laden . . . . .	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- / ausfalten . . . . .	22
3.4 Suchen, markieren und ersetzen . . . . .	24
3.4.1 Text schnell suchen . . . . .	24
3.4.2 Text suchen und ersetzen . . . . .	24
Beispiele für das Suchen von Text . . . . .	28
Beispiele für das Ersetzen von Text . . . . .	29
3.4.3 Reguläre Ausdrücke . . . . .	29
3.4.4 Kontrollstruktur markieren . . . . .	32
3.4.5 Textmarken nutzen . . . . .	32
3.4.6 Zu einer Programmzeile springen . . . . .	33

3.4.7 Zur Deklaration eines Befehls oder Variablen springen	33
3.5 Programme leichter schreiben	34
3.5.1 Befehle und Variablen automatisch vervollständigen	34
3.5.2 Textbausteine einfügen	36
3.5.3 Befehlsparameter anzeigen	36
3.5.4 Deklaration eines Befehls oder Variablen anzeigen	37
3.5.5 Deklarationen einer Datei anzeigen	37
3.5.6 Verwendete globale Variablen und Felder anzeigen	38
3.6 Projektverwaltung	38
3.7 Menüs	39
3.7.1 Das Menü „File“	40
3.7.2 Das Menü „Edit“	41
3.7.3 Das Menü „View“	41
3.7.4 Das Menü „Build“	42
3.7.5 Das Menü „Options“	43
Dialogfenster „Compiler Options“	43
Dialogfenster „Process Options“	46
Dialogfenster „Settings“	49
3.7.6 Das Menü Debug	53
Option Timing Analyzer	53
Option Debug mode	53
3.7.7 Das Menü „Tools“	56
3.7.8 Das Menü „Window“	57
3.7.9 Das Menü „Help“	57
3.8 Fenster	58
3.8.1 Toolbox	58
3.8.2 Projektfenster	58
3.8.3 Parameterfenster	60
3.8.4 Prozessfenster	61
3.8.5 Statusleiste	63
3.9 Info-Bereich	64
3.9.1 Infofenster	65
3.9.2 ToDo-Liste	66
3.9.3 Fenster „Timing Analyzer“	67
3.9.4 Fenster „Global Variables“	70
3.9.5 Fenster „Declarations“	72
3.10 ADtools	73

4 Prozesse programmieren	75
4.1 Programmaufbau	75
4.1.1 Die Programmabschnitte	77
4.1.2 Benutzerdefinierte Befehle und Variablen	77
4.2 Variablen und Felder	79
4.2.1 Übersicht	79
4.2.2 Datenstrukturen	79
4.2.3 Datentypen	80
4.2.4 Zahlenwerte eingeben	82
4.2.5 Globale Variablen (Parameter)	82
4.2.6 Globale Felder (Arrays)	83
4.2.7 System-Variablen	85
4.2.8 Lokale Variablen und Felder	86
4.3 Variablen und Felder – Details	87
4.3.1 Variablen und Felder im Datenspeicher	87
4.3.2 Speicherbereiche	88
4.3.3 2-dimensionale Felder	89
4.3.4 Die Datenstruktur FIFO	90
4.3.5 Strings	93
Normale Zuweisung	94
Zuweisung per Escape-Sequenz	95
Nicht empfohlene Arten der Zuweisung	96
4.4 Berechnungsausdrücke	97
4.4.1 Auswertung von Operatoren	97
4.4.2 Typkonvertierung	98
4.5 Bedingungen, Schleifen und Module	100
4.5.1 Unterprogramm- und Funktions-Makros	100
4.5.2 Include-Dateien	101
4.5.3 Bibliotheken (Libraries)	101
5 Prozesse optimieren	105
5.1 Bearbeitungszeit messen	105
5.2 Verschiedene Tipps	106
5.2.1 Zugriff auf Hardware-Adressen	106
5.2.2 Konstanten anstelle von Variablen	107
5.2.3 Schnellere Messfunktion	107
5.2.4 Wartezeit genau einstellen	108

5.2.5 Wartezeiten nutzen	110
5.2.6 Optimierung mit dem Prozessor T11	111
5.3 Debugging und Analyse	112
5.3.1 Laufzeitfehler erkennen (Debug-Modus)	113
5.3.2 Zeitverhalten prüfen (Timing-Modus)	113
Anzahl und Priorität von Prozessen prüfen	114
Optimales Zeitverhalten eines Prozesses	115
6 Prozesse im Betriebssystem	117
6.1 Prozessverwaltung	118
6.1.1 Prozessarten	118
6.1.2 Prozesse mit der Priorität „Hoch“	119
6.1.3 Prozesse mit der Priorität „Niedrig“	119
6.1.4 Kommunikationsprozess	120
6.1.5 Speicherfragmentierung	120
6.2 Zeitverhalten von Prozessen	122
6.2.1 Processdelay	122
6.2.2 Zeitlich exakter Aufruf von Prozesszyklen	123
6.2.3 Niederpriorie Prozesse beim T11	124
6.2.4 Auslastung des ADwin-Systems	125
6.2.5 Verschiedene Betriebszustände im Betriebssystem	126
6.3 Kommunikation	128
6.3.1 Datenaustausch zwischen Prozessen	128
6.3.2 Kommunikation zwischen PC und ADwin-System	128
6.3.3 Die Device No	129
6.3.4 Kommunikation mit Entwicklungsumgebungen	130
7 Befehlsreferenz	131
7.1 Befehlssyntax	131
7.2 Befehle L16, Gold, Pro	132
7.3 FFT-Library	279
7.4 Mathematik-Befehle	297
8 Was tun bei Problemen?	299
Anhang	A-1
A.1 Tastaturkürzel in ADbasic	A-1

A.2 ASCII-Zeichensatz . . . . .	A-3
A.3 Lizenzvertrag . . . . .	A-5
A.4 Kommandozeilen-Aufruf . . . . .	A-9
A.5 Obsolete Programmteile . . . . .	A-15
A.6 Liste der Debug-Fehlermeldungen . . . . .	A-18
A.7 Index . . . . .	A-20
A.8 Befehle in diesem Handbuch. . . . .	A-37





## Liebe Leserin, lieber Leser!

*ADbasic* 5 ist das Werkzeug, mit dem Sie Ihr *ADwin*-System für Ihre spezielle Mess-, Regel- oder Steuer-Aufgabe programmieren. Dieses Handbuch führt Sie einerseits in die Grundlagen der Programmierung von Echtzeit-Prozessen ein, andererseits soll es Ihnen als Nachschlagewerk dienen.

Die Entwicklungsumgebung *ADbasic* 5 ist vollständig überarbeitet und bietet deutlich mehr Komfort bei der Handhabung und beim Editieren (siehe auch „[Neues in ADbasic 5](#)“ auf ).

Auch der Handbuch-Inhalt hat sich verändert: Die Befehlsreferenz beinhaltet nur noch die Befehle für Rechenoperationen des Prozessors. Alle Befehle für Zugriffe auf Ein- und Ausgänge oder Schnittstellen sind im jeweiligen Handbuch der *ADwin*-Hardware beschrieben.

Wenn Sie *ADbasic* das erste Mal verwenden, empfehlen wir Ihnen den schnellen Einstieg mit [Kapitel 1](#) und [4](#). Wir setzen voraus, dass Sie bereits über grundlegende Kenntnisse des Programmierens z.B. in Basic verfügen. Eine Einführung ins Programmieren von *ADwin*-Systemen und praktische Tipps finden Sie in unserem Tutorial.

[Kapitel 3](#) beschreibt die überarbeitete 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. Für diese Aufgabe programmieren Sie das *ADwin*-System mit dem Entwicklungssystem *ADbasic*.

Damit Sie schnell und effizient mit der Programmierung beginnen können, möchten wir Ihnen zunächst das Konzept eines *ADwin*-Systems kurz erklären.

Alle *ADwin*-Systeme besitzen als zentrale Einheit einen Prozessor, der alle zeitkritischen Aufgaben wie Messdaten-Erfassung, Regelung, Steuerung, Signalgenerierung oder Online-Verarbeitung von einzelnen Messwerten in Echtzeit ausführt. Analoge und digitale Ein- und Ausgänge sowie Erweiterungen wie Zähler und Bussysteme bilden die Verbindung zu Ihrem Prüfstand; die Kommunikation mit dem PC geschieht über Ethernet oder USB.

Der Prozessor des *ADwin*-Systems wird mit dem Echtzeit-Entwicklungs-Tool *ADbasic* programmiert, das die einfache und schnelle Erstellung von zeitkritischen Echtzeit-Prozessen ermöglicht. *ADbasic* ist eine integrierte Entwicklungsumgebung unter Windows mit Möglichkeiten zum Online-Debugging. Die gewohnte BASIC-Befehlssyntax wurde um Funktionen erweitert, mit denen Sie auf Ein- und Ausgänge zugreifen, Echtzeitprozesse steuern und den Datenaustausch mit dem PC vorbereiten können. In [Kapitel 4](#) ist erklärt, wie Sie *ADbasic*-Programme aufbauen.

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

Wenn Sie für einen komplexen Algorithmus mehrere Prozesse verwenden, regelt eine (einstellbare) Hierarchie das zeitliche Zusammenspiel der Prozesse untereinander. Einzelheiten zum Ablauf Ihrer Prozesse im Betriebssystem finden Sie in [Kapitel 6](#).



Die Entwicklungsumgebung *ADbasic* 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 (siehe [Kapitel 4](#)).

Mit dem integrierten Compiler erzeugen Sie aus dem Quelltext lauffähigen Binärcode, der als Prozess auf das *ADwin*-System übertragen und getestet



wird. *ADbasic* bietet Ihnen auch die Hilfsmittel, mit denen Sie Ihre Prozesse beobachten, Fehler suchen und die Programme optimieren können (siehe [Kapitel 3](#)).

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

Mit einer Bedienoberfläche können Sie die Prozesse und Prozessdaten des *ADwin*-Systems vom PC aus steuern und beobachten, d.h. den fertigen Binärcode zum System übertragen sowie die Prozesse starten, steuern und stoppen.

Obwohl das *ADwin*-System autark arbeitet, können Sie aus Ihrer Bedienoberfläche jederzeit auf globale Variablen und Felder zugreifen, ohne zeitkritische Prozesse zu verzögern. Über die globalen Variablen und Felder können alle Prozesse untereinander oder mit dem PC schnell Daten austauschen.

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

Unter Windows ermöglicht Ihnen eine DLL- oder ActiveX-Schnittstelle, auf das *ADwin*-System auch aus mehreren Programmen gleichzeitig zuzugreifen.

Darauf basierend gibt es Treiber für .NET sowie für alle gängigen Entwicklungsumgebungen, mit denen Sie Ihre eigene Bedienoberfläche gestalten können, z. B. Delphi, Visual-Basic, C#.NET, Visual-C++. Alternativ können Sie auch Messtechnik-Pakete wie TestPoint, LabVIEW, Diadem, HP-VEE, Intouch und Matlab nutzen.

Schließlich stehen auch Treiber für die Plattformen Linux, MacIntosh und Java zur Verfügung.

## 2 Neues in ADbasic 5

Mit ADbasic 5 arbeiten Sie wie gewohnt, allerdings komfortabler und mit neuer Optik.

Unter der neuen Oberfläche verbergen sich viele Neuigkeiten, die sich oft erst auf den zweiten Blick zeigen. Sie werden bald feststellen, dass die neuen Funktionen das Programmieren erheblich leichter machen.

Probieren Sie es aus!

### Einfacher programmieren

- [Befehle und Variablen automatisch vervollständigen](#) mit CTRL-SPACE (Seite 34).
- [Textbausteine einfügen](#) mit Tastaturkürzeln (Seite 36) oder CTRL-SPACE (Seite 34).
- [Deklaration eines Befehls oder Variablen anzeigen](#) (Seite 37).
- Alle [Deklarationen einer Datei anzeigen](#) (Seite 37).
- [ToDo-Liste](#) zur Verwaltung offener Aufgaben (Seite 66).
- Neue Tastaturkürzel (siehe [Kapitel A.1](#)).

### Verbesserte Quelltextdarstellung

- Automatisch [Textzeilen einrücken](#) (Seite 22).
- Zeilennummern am linken Rand.
- [Syntax hervorheben](#) mit verschiedenen Farben (Seite 21) und Schriftschnitten.

Beachten Sie: Wenn ein Befehl oder eine Variable nicht hervorgehoben wird, ist das ein Hinweis, dass das Wort falsch geschrieben ist oder die entsprechende Include-Datei nicht eingebunden wurde.

- Farbbalken am linken Rand für geänderte Zeilen.
- [Textbereiche ein- / ausfalten](#) (Seite 22).

### Umstellungen und Neues in der Oberfläche

- Neue [Editor-Leiste](#), über die Sie viele neue Funktionen aufrufen können (Seite 20).



- Neue [ADtools](#)-Leiste zum Direktstart der praktischen Hilfsprogramme (Seite 73).
- Werkzeugleiste:
  - Neue Schaltflächen für die [Projektverwaltung](#) (Seite 38).
  - Die Device No. wurde in die [Statusleiste](#) verschoben.
- Projekt-, Parameter- und Prozessfenster sind gemeinsam in der [Tool-box](#) organisiert (Seite 58).
- Im [Projektfenster](#) (Seite 58) werden Dateitypen nach Quelltext- und Include-Dateien gruppiert angezeigt.
- Im Quelltextfenster gibt es einen Reiter für jede geöffnete Datei.
- Die Statusleiste zeigt u. a. aktuelle Einstellungen des Compilers an (Seite 63). Ein Doppelklick auf eine Einstellung öffnet das entsprechende Dialogfenster.

### Schneller suchen und finden

- [Text suchen und ersetzen](#) über mehrere Dateien hinweg (Seite 24); auch [Reguläre Ausdrücke](#) sind dabei möglich (Seite 29).
- [Textmarken nutzen](#) (Seite 32).
- [Zu einer Programmzeile springen](#) (Seite 33).
- [Zur Deklaration eines Befehls oder Variablen springen](#) (Seite 33).

### Sonstige Neuheiten

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

Auch Library-Quelltexte haben ein eigenes Speicherformat (Dateityp `LibFile` mit der Dateierweiterung `*.bas`). Nur mit diesem Dateiformat ist das Erzeugen der Library-Binärdatei möglich.

- Sie können alle Dateien eines Projekts gleichzeitig kompilieren und dabei sowohl Binärdateien als auch Library-Dateien erzeugen: Menü Build, Eintrag `Make All Bin files of Project`.
- Vor dem Kompilieren werden geänderte Dateien automatisch gespeichert (im Format *ADbasic 5*, siehe oben).
- Im Quellcode können Sie für Include- und Library-Dateien relative Pfade verwenden.

Das Basisverzeichnis ist – wenn der Quelltext Teil einer Projektdatei ist – das Verzeichnis der Projektdatei, anderenfalls das Verzeichnis der Quelltextdatei.

- Der Aufruf über Kommandozeile ist vollständig überarbeitet und verbessert (siehe [Kapitel A.4](#), Anhang Seite 9). Durch die Umstellung ist der Aufruf nicht mehr kompatibel zu *ADbasic 4*.





## 3 Entwicklungsumgebung

Mit der Entwicklungsumgebung *ADbasic* können Sie einfach und schnell Prozesse für *ADwin*-Systeme programmieren und testen. Der *ADbasic*-Compiler verarbeitet eine erweiterte Basic-Syntax und erzeugt Binärdateien, die Sie auch ohne die Entwicklungsumgebung auf das *ADwin*-System übertragen und ausführen können.

### 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** ► **ADbasic** 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 *ADbasic*-Handbuchs.

Ohne Eingabe des gültigen **License key** befindet sich *ADbasic* 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 *ADbasic*-Lizenz finden Sie im Kapitel 3.1.2 auf Seite 10.

3. Stellen Sie nun das von Ihnen verwendete *ADwin*-System und den Prozessor im Menü **Options** ► **Compiler** ein.

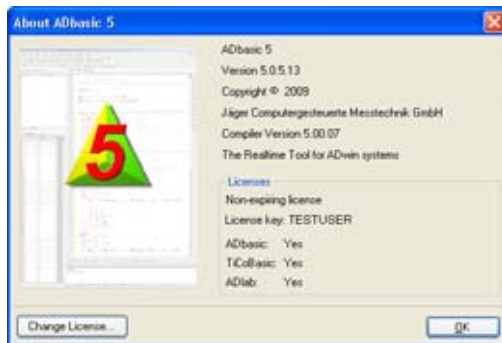
Die Entwicklungsumgebung speichert Ihre Angaben, so dass Sie sie bei einem erneuten Start nicht mehr eingeben müssen (es sei denn, Sie verwenden eine andere *ADwin*-Hardware).

### 3.1.2 Lizenzen für ADbasic prüfen und ändern

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

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

Das Fenster **About ADbasic** 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 *ADbasic*-Handbuchs.

Für *ADbasic* sind folgende Lizenzen möglich:

- Keine Lizenz (Demo mode)

Ohne Eingabe des gültigen **License key** befindet sich *ADbasic* 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.

- Zeitlich begrenzte Lizenz (Evaluation license)

Die Lizenz schaltet alle Funktionen der Entwicklungsumgebung für einen definierten Zeitraum frei. Anschließend arbeitet *ADbasic* 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 *ADbasic* sind im Lizenzvertrag (Anhang Seite A-5) beschrieben.

### 3.1.3 ADwin-Betriebssystem laden

Sie übertragen das *ADwin*-Betriebssystem in Ihr *ADwin*-System, indem Sie auf die Schaltfläche  klicken (= booten).

Diesen Boot-Vorgang benötigen Sie immer dann, wenn Sie Ihr *ADwin*-System (nach einer Unterbrechung der Stromversorgung) wieder einschalten oder wenn der PC einen Kommunikationsfehler erkannt und die Verbindung zum System unterbrochen hat.

Wenn Sie das Betriebssystem übertragen, gehen gleichzeitig die Inhalte des Programm- und Datenspeichers auf dem *ADwin*-System verloren und alle globalen Parameter werden auf den Wert 0 gesetzt.




Für jeden Prozessortyp wird das passende Betriebssystem benötigt, das jeweils in einer eigenen Datei *ADwin\*.btl* vorhanden ist (\* steht für den Prozessortyp). Aus Ihrer Einstellung im Menü *Options \ Compiler* entnimmt die Entwicklungsumgebung die Information, welche dieser Dateien beim Boot-Vorgang zu übertragen ist.

Die Dateien *ADwin\*.btl* werden bei der Installation im Verzeichnis *<C:\ADwin>* abgelegt (Standardinstallation).

### 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.

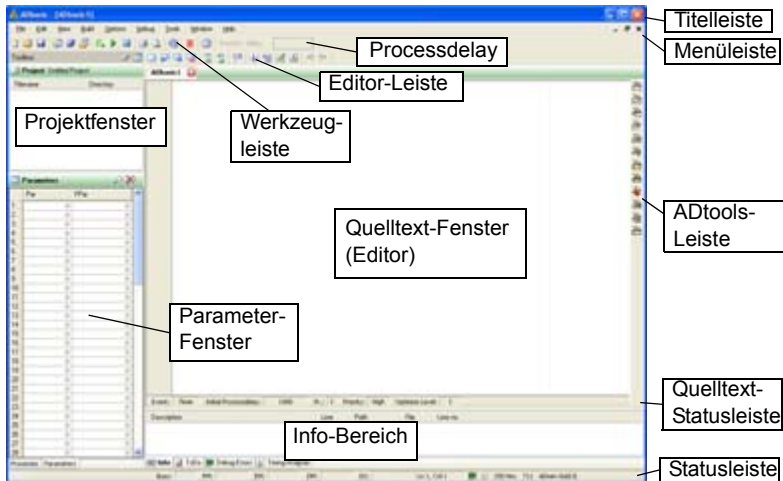


Abb. 1 – Elemente der ADbasic-Entwicklungsumgebung

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.

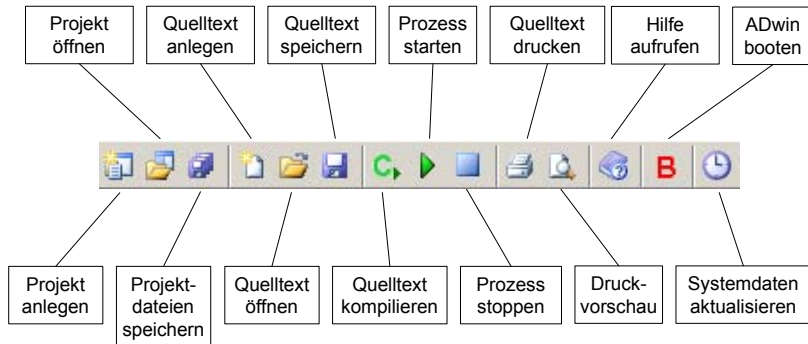



Abb. 2 – Die Werkzeugleiste

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 Quelltext-fenster 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 3.8.3 auf Seite 60) 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 38.

- Im Infobereich unten werden in verschiedenen Fenstern eine Reihe von Informationen angezeigt:
  - Infofenster: Fehlermeldungen (rot hinterlegt) und Warnungen des Compilers (siehe Kapitel 3.9.1 auf Seite 65).
  - ToDo-Liste: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.9.2 auf Seite 66).
  - Suchergebnisse bei einer Suche über alle Dateien eines Projekts (siehe Kapitel 3.4.2 auf Seite 24).
  - Debug-Informationen, wenn der Debug-Modus aktiv ist (siehe Option Debug mode, Seite 53).
  - ToDo-Liste: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.9.2 auf Seite 66).
  - Fenster „Global Variables“: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.9.2 auf Seite 66).
  - Fenster „Declarations“: Eine einfache ToDo-Liste aus Kommentarzeilen (siehe Kapitel 3.9.2 auf Seite 66).

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 Ihres *ADwin*-Systems werden kontinuierlich ausgelesen und angezeigt (nur, wenn der PC mit dem *ADwin*-System 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 61).
- Informationen zur Speicherauslastung in der Statusleiste (siehe Kapitel 3.8.5).

Je nach Compiler-Einstellung können zusätzliche Informationen über laufende Prozesse angezeigt werden:

- Das Zeitverhalten im Timing-Fenster (Seite 98).
- Laufzeitfehler im Debug-Fenster (siehe Option Debug mode, Seite 53).



### 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 38: Projektverwaltung).

Der Editor und der *ADbasic*-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 34



## 3.2.1 Online-Hilfe aufrufen

Das Menü „Help“ (Seite 57) 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 37.
- `Jump to Declaration`: Zur Deklaration eines Befehls oder Variablen springen, Seite 33.

Ohne vorherige Markierung sind folgende Befehle verfügbar:

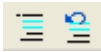
- `Add to Project`: Datei dem aktiven Projekt zufügen.
- `Code snippets`: Textbausteine einfügen, Seite 36.
- `Show all Declarations`: Deklarationen einer Datei anzeigen, Seite 37.

### 3.2.3 Editor-Leiste

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



Textmarken nutzen, Seite 32.



Zeilen in Kommentar ändern, Seite 22.



Textbereiche ein- / ausfalten, Seite 22.



Deklaration eines Befehls oder Variablen anzeigen, Seite 37.



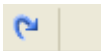
Zur Deklaration eines Befehls oder Variablen springen, Seite 33.



Textbausteine einfügen, Seite 36.



Deklarationen einer Datei anzeigen, Seite 37.



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 22

Hier finden Sie weitere Editor-Funktionen:

- Quelltexte erstellen, Seite 16
- Suchen, markieren und ersetzen, Seite 24
- Programme leichter schreiben, Seite 34

### 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 50) 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 49) 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 49) .

### 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

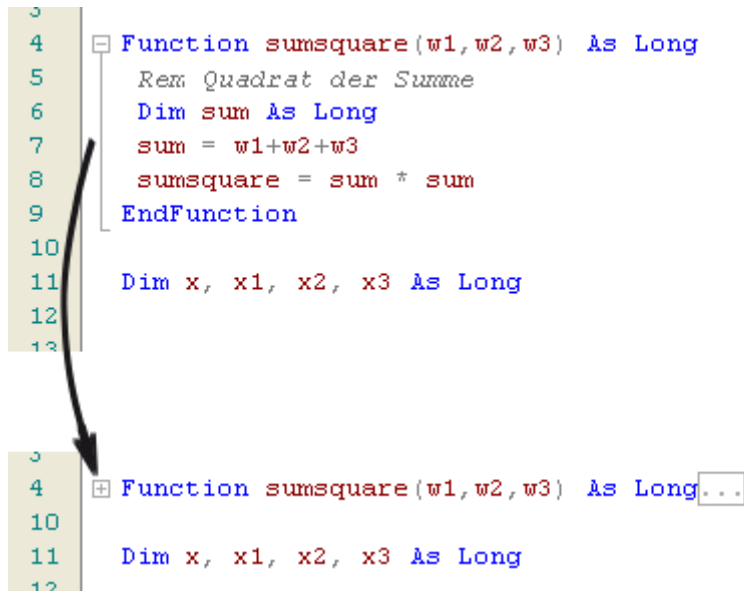
Mehrere markierte Zeilen können gemeinsam in Kommentarzeilen umgewandelt werden, indem im Kontextmenü des Quelltextfensters (Klick mit rechter Maustaste) der Eintrag `Comment Block` gewählt wird. Als Folge fügt der Editor an jedem Zeilenanfang ein Kommentarzeichen ein, so dass der Compiler diese Zeilen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



```


3
4  Function sumsquare(w1,w2,w3) As Long
5      Rem Quadrat der Summe
6      Dim sum As Long
7      sum = w1+w2+w3
8      sumsquare = sum * sum
9  EndFunction
10
11  Dim x, x1, x2, x3 As Long
12
13

```

```

3
4  Function sumsquare(w1,w2,w3) As Long...
10
11  Dim x, x1, x2, x3 As Long
12

```

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 49) 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 24
- Reguläre Ausdrücke, Seite 29
- Kontrollstruktur markieren, Seite 32
- Textmarken nutzen, Seite 32
- 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 34

#### 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.

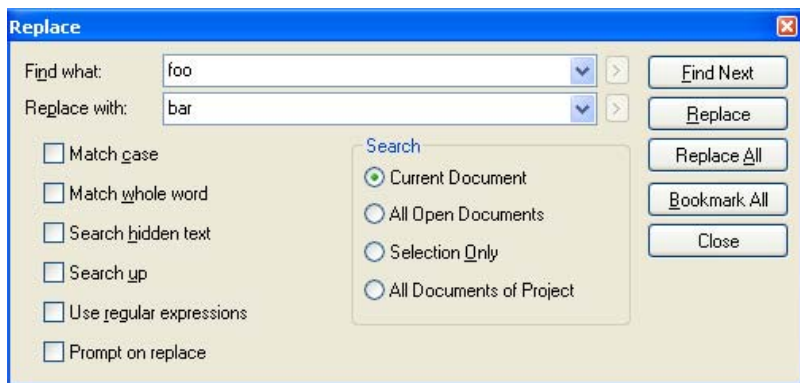
#### 3.4.2 Text suchen und ersetzen

Sie können nach jedem Auftreten einer beliebigen Kombination von Zeichen suchen, zum Beispiel auch nach Groß- und Kleinbuchstaben, ganzen Wör-



tern, Teilen von Wörtern und regulären Ausdrücken (siehe Reguläre Ausdrücke auf Seite 29).

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>
Match whole word	<p>Option aktiviert: Es wird nur nach Text gesucht, bei dem der Suchausdruck ein ganzes Wort ist.</p> <p>Option deaktiviert: Der Suchausdruck kann auch ein Teil eines Wortes sein.</p>

Option	Beschreibung
Search hidden text	<p>Die Option bezieht sich auf das Textbereiche ein- / ausfallen (siehe Seite 22).</p> <p>Option aktiviert: Es werden auch eingeklappte Textbereiche durchsucht.</p> <p>Option deaktiviert: Eingeklappte Textbereiche werden übersprungen.</p>
Search up	<p>Option aktiviert: Es wird in Richtung zum Dateianfang hin gesucht.</p> <p>Option deaktiviert: Es wird in Richtung zum Dateiende hin gesucht.</p>
Use regular expressions	<p>Die Option bezieht sich auf Reguläre Ausdrücke (siehe Seite 29).</p> <p>Option aktiviert: Der eingegebene Suchausdruck ist ein regulärer Ausdruck; das ist eine Methode, die variable Textmuster beschreibt.</p> <p>Option deaktiviert: Der eingegebene Suchausdruck ist konstanter Text.</p>
Prompt on replace	<p>Die Option ist nur in Verbindung mit Replace All einsetzbar.</p> <p>Option aktiviert: Bei jedem Vorkommen erscheint ein Dialogfenster, mit dem Sie das Ersetzen steuern.</p> <p>Option deaktiviert: Alle Vorkommen werden ohne Rückfrage ersetzt.</p>

5. Stellen Sie einen Bereich für die Suche ein:

Option	Beschreibung
Current Document	<p>Die Suche beginnt im aktuellen Quelltext an der Cursor-Position.</p> <p>Wenn Text markiert ist, steht der Cursor hinter der Markierung.</p>
All open Documents	Alle geöffneten Dateien werden durchsucht, beginnend mit dem aktuellen Quelltext.

Option	Beschreibung
Selection only	Nur der markierte Bereich wird durchsucht. Falls keine Markierung vorhanden ist, startet die Suche an der Cursor-Position.
All Documents of Project	<p>Alle Dateien des Projekts werden durchsucht, auch wenn der aktuelle Quelltext nicht zum Projekt gehört. Für Ersetzen nicht verfügbar.</p> <p>Die Suchergebnisse werden im Fenster <code>Find</code> im Infobereich angezeigt.</p> <p>Mit einem Doppelklick auf ein Ergebnis springt der Cursor in die entsprechende Zeile. Alternativ springen Sie mit den Pfeil-Schaltflächen im Fenster <code>Find</code> von einem Ergebnis zum nächsten.</p>

6. Starten Sie die gewünschte Aktion durch eine der Schaltflächen.
  - `Find Next`: 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 *ADbasic* 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 1. explizit in den eckigen Klammern angegeben sind, oder 2. eines aus dem Bereich von Zeichen, der durch den Bindestrich (-) definiert ist. 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.
\$	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.

Meta- zeichen:	Bedeutung:
\n	<p>Zeilenschaltzeichen (Newline). Zu verwenden bei Ausdrücken, die sich über mehrere Zeilen erstrecken.</p> <p>Hinter \n dürfen nicht die Operatoren *, + oder {} stehen. Um ein Suchmuster am Ende einer Zeile zu finden, sollten Sie unbedingt den Operator \$ und nicht das Zeilenschaltzeichen verwenden.</p>
( )	<p>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.</p> <p>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 \$x im Ersetzungsausdruck und \x im Suchausdruck, mit x = 1 ... 9.</p> <p>Beispiel: Der Ausdruck ([a-z]+) ([a-z]+) findet isses so. \$2 \$1 würde ihn ersetzen durch so isses.</p>
*	<p>Beliebig häufiges Auftreten des vorangehenden Zeichens oder Ausdrucks – kein Mal, einmal oder mehrmals.</p> <p>Beispiel: hu*f findet unter anderem hf, huf und huuf.</p>
?	<p>Kein oder genau ein Auftreten des vorangehenden Zeichens oder Ausdrucks.</p> <p>Beispiel: hu?f findet hf und huf, nicht aber huuf.</p>
+	<p>Mindestens ein (d.h. ein- oder mehrmaliges) Auftreten des vorangehenden Zeichens oder Ausdrucks.</p> <p>Beispiel: hu+f findet huf und huuf, nicht aber hf.</p>
	<p>Entweder der Ausdruck auf der linken Seite des Striches oder der auf der rechten Seite.</p> <p>Beispiel: huf huuf findet huf oder huuf.</p>
\	<p>Hebt die besondere Wirkung aller Operatoren auf, so dass sie wie normaler, konstanter Text behandelt werden. Um nach dem Backslash \ zu suchen, muss also \\ verwendet werden.</p> <p>Beispiel: ^a bedeutet, dass nach einem a am Anfang einer Zeile gesucht wird, dagegen sucht \^a nach der Zeichenfolge ^a.</p>

### 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:`, `LowInit:`, `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 22).

### 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 line numbers` unter `Editor - General` (siehe Seite 49) 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 49) aktiv ist.

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

### 3.5 Programme leichter schreiben

Die folgenden Funktionen erleichtern das Programmieren erheblich:

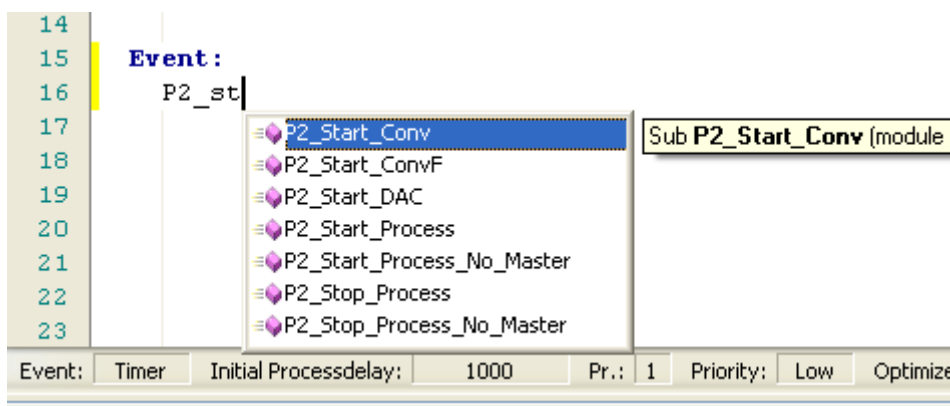
- Befehle und Variablen automatisch vervollständigen, Seite 34
- Befehlsparameter anzeigen, Seite 36
- Textbausteine einfügen, Seite 36
- Deklaration eines Befehls oder Variablen anzeigen, Seite 37
- Deklarationen einer Datei anzeigen, Seite 37
- Verwendete globale Variablen und Felder anzeigen, Seite 38

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.

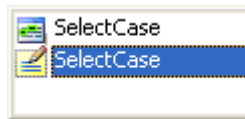
Automatisches Vervollständigen ist nur möglich, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 49) aktiv ist.

### 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\ADbasic\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 49) 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 49) 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 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 49) 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. Deklarationen aus anderen Quelltextdateien – auch innerhalb eines Projekts – werden nicht berücksichtigt.

Die Deklarationen können nur angezeigt werden, wenn die Option `Parse Declarations` unter Editor - General (siehe Seite 49) 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 60) drücken.

Sie erhalten zwei Anzeigen:

- im Fenster „Global Variables“ werden alle verwendeten globalen Variablen und Felder angezeigt. Näheres siehe Seite 70.
- 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. 
- Rot: Parameter wird im aktiven Quelltext verwendet, aber auch in einem anderen Quelltext des Projekts. 
- Blau: Parameter wird im Projekt verwendet, jedoch nicht im aktiven Quelltext. 

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 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.

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 auf einmal kompilieren, mit dem Menüeintrag `Build ▶ Make all Bin Files of Project`.
- 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 58) oder im Menü `File` (Kapitel 3.7.1).

## 3.7 Menüs

Sie finden in der Menüleiste folgende Funktionen:

- |                         |  |            |
|-------------------------|--|------------|
| – <code>File:</code>    | Dateien und Projekte verwalten.            | (Seite 40) |
| – <code>Edit:</code>    | Quelltexte editieren.                      | (Seite 41) |
| – <code>View:</code>    | Fenster und Leisten anzeigen.              | (Seite 41) |
| – <code>Build:</code>   | Ausführbare Programme erzeugen.            | (Seite 42) |
| – <code>Options:</code> | Optionen aller Art einstellen.             | (Seite 43) |
| – <code>Debug:</code>   | Hilfe zur Fehlersuche.                     | (Seite 53) |
| – <code>Tools:</code>   | Verschiedene Hilfsfunktionen.              | (Seite 56) |
| – <code>Window:</code>  | Quelltextfenster anordnen.                 | (Seite 57) |
| – <code>Help:</code>    | Hilfe, Versions- und Lizenz-Informationen. | (Seite 57) |

### 3.7.1 Das Menü „File“

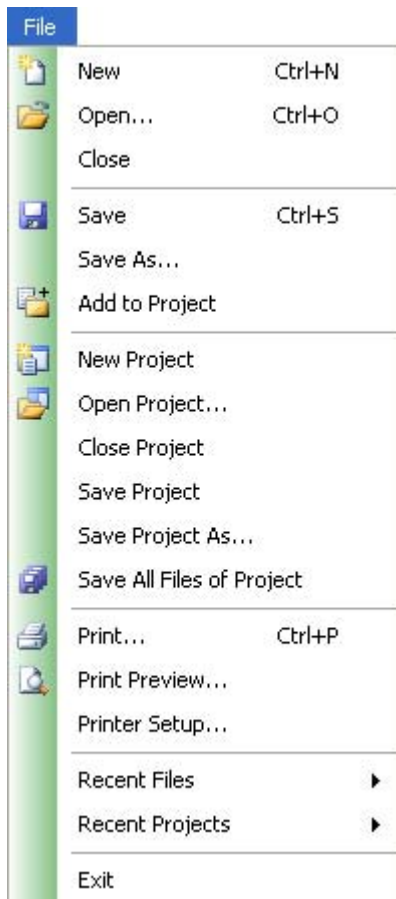
Das Menü **File** enthält Befehle zum Verwalten von Dateien und Projekten.

Mit den Befehlen können Sie Dateien öffnen, neu anlegen, speichern oder schließen. Sie können beliebig viele Quelltexte bearbeiten, aber höchstens 10 Prozesse gleichzeitig auf Ihr ADwin-System laden.

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 3.8.2).

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

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.







## 3.7.2 Das Menü „Edit“

Das Menü `Edit` enthält die nach den Windows-Konventionen üblichen Editor-Funktionen.

Darüber hinaus enthält das Menü eine Such-Funktion (`Find`, `Find Next`) sowie eine Ersetzen-Funktion (`Replace`); siehe Text suchen und ersetzen auf Seite 24.

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

## 3.7.3 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 3.8.4 auf Seite 61, zur Werkzeugleiste siehe Abb. 2.

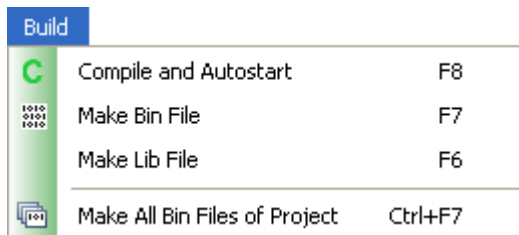
View	
<input checked="" type="checkbox"/>	Standard Toolbar
<input checked="" type="checkbox"/>	Editor Toolbar
<input checked="" type="checkbox"/>	ADtools Toolbar
<input checked="" type="checkbox"/>	Statusbar
<hr/>	
	Restore Default Layout

Mit `Restore Default Layout` stellen Sie mit einem Tastendruck die Standard-Ansicht wieder her, die beim ersten Öffnen des Programms *ADbasic* aktiv war, darunter auch die Einstellungen der Toolbox (Seite 58).

### 3.7.4 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.
- mit `Make all Bin Files of Project` alle Dateien des Projekts in Binärdateien.



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

Eine Änderung kann auch durch automatisches Einrücknen 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 den aktiven Quelltext und überträgt den erzeugten Binärcode als Prozess auf das ADwin-System.

Der Prozess wird außerdem auch automatisch gestartet, wenn Sie im Menü `Options\Compiler` den Eintrag `Autostart` auf `Yes` gesetzt haben. Anderenfalls können Sie den Prozess mit der Schaltfläche  aus der Werkzeugleiste oder im Prozessfenster starten (siehe Seite 62).

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 *ADbasic*-Benutzer verfügbar. Der Befehl übersetzt den aktiven Quelltext in Binärcode und speichert diesen automatisch in einer Datei ab. Die Binärdatei wird mit dem Namen und im Verzeichnis der Quelltext-Datei abgelegt, jedoch mit der Dateieindung `.Txn`.

Hierbei steht *x* für den Prozessortyp und *n* für die Prozessnummer (siehe auch Das Menü „Options“, Dialogfenster „Process Options“).

Binärdatei mit *TiCoBasic* Eine Binärdatei mit der Endung *<\*.TA3>* beispielsweise können Sie auf ein *ADwin*-System mit dem Prozessor T10 übertragen, der den Prozess unter der Nummer 3 startet. Sie können Binärdateien aus allen Entwicklungsumgebungen (wie C oder Visual Basic) an das *ADwin*-System übertragen (siehe Kapitel 6.3.4 auf Seite 130).



*Make Lib File* ist nur für lizenzierte *ADbasic*-Benutzer verfügbar. Der Befehl übersetzt den aktiven Quelltext – die Datei muss mit dem Dateityp *Lib-File* gespeichert sein – 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 Dateiendung *.LIX*. 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 101).

*Make All Bin Files of Project* ist nur für lizenzierte *ADbasic*-Benutzer verfügbar. Der Befehl ist eine Kombination der Befehle *Make Bin File* und *Make Lib File*: Er übersetzt alle Quelltexte des Projekts und erzeugt dabei sowohl Library-Dateien als auch Binärdateien.

### 3.7.5 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, auf dem die übersetzten Quelltexte als Prozess laufen sollen.

Wenn Sie Quelltexte für verschiedene *ADwin*-Systeme kompilieren möchten, müssen Sie die Einstellungen in diesem Dialogfenster für jedes System neu anpassen.



Abb. 3 – Das Dialogfenster Compiler Options

- **System:** Wählen Sie den Eintrag aus, der Ihrem *ADwin*-System entspricht.
- **Processor:** Wählen Sie den Prozessor des *ADwin*-System.


Die Kurzbezeichnungen der Prozessor-Typen entsprechen den folgenden Vollbezeichnungen:

Kurzbezeichnung	T11	T10	T9	T8	T5	T4	T2
Vollbezeichnung	ADSP TS101S	ADSP 21160	ADSP 21062	T805	T450	T400	T225

Abb. 4 – Prozessorbezeichnungen

- **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.

- **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.
- **Load standard processes:** Bei aktiver Option werden während des Boot-Vorgangs die Standard-Prozesse 11, 12 und 15 (siehe Kapitel 6.1.1 auf Seite 118) in das *ADwin*-System übertragen. Bei deaktivierter Option wird die Übertragung der Prozesse 11 und 12 unterdrückt.  
Die Einstellung ist nur für *ADwin-Gold* und *ADwin-light-16* verfügbar.
- **Autostart:** Bei aktiver Option wird die beim Kompilieren erzeugte und ins *ADwin*-System übertragene Binärdatei sofort als Prozess gestartet. Bei deaktivierter Option müssen Sie den Prozess manuell mit der Schaltfläche  aus der Werkzeugeiste oder im Prozessfenster starten.
- **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 *ADbasic* wird die zuletzt – bei aktiver Option – gespeicherte Device No. verwendet.

### Dialogfenster „Process Options“

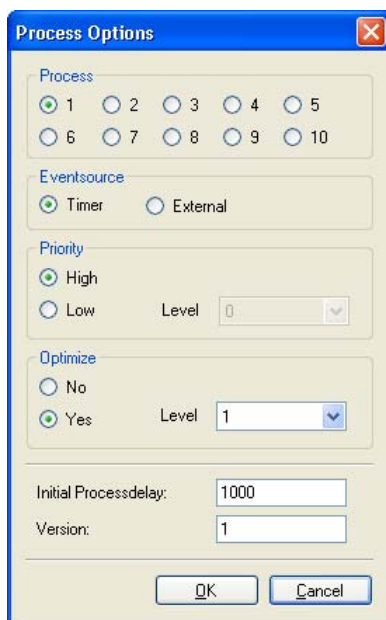
Sie legen in diesem Dialogfenster Compiler-Optionen für das aktive Quelltextfenster fest, d.h. Sie bestimmen Eigenschaften des Prozesses, der aus dem aktiven Quelltext übersetzt und zur ADwin-Hardware übertragen wird.

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.

Das Dialogfenster für die Prozessortypen T4, T5 oder T8 weicht vom Standardfenster in wenigen Punkten ab und ist im Anhang beschrieben.

Einstellungen für Quelltext



Einstellungen für Library

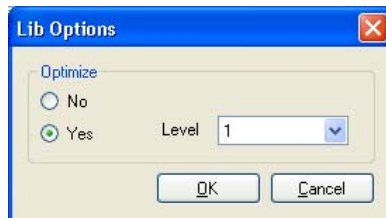


Abb. 5 – Das Dialogfenster *Process Options*

- Process: Prozessnummer

Stellen Sie die Nummer ein, unter der der übertragene Prozess auf dem System angesprochen wird.

Wenn Sie mehrere Prozesse gleichzeitig auf einem *ADwin*-System 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**  
Ein Signal am Event-Eingang der *ADwin*-Hardware startet den Prozess. Dies könnte beispielsweise ein Impuls eines Messaufnehmers sein. Ein solcher Prozess läuft immer mit hoher Priorität ab.  
Wie Sie bei einem *ADwin-Pro*-System einen externen Event-Eingang nutzen können, lesen Sie bitte in der Software-Dokumentation unter dem Befehl **EventEnable** nach.
- **Priority:** Priorität des Prozesses.  
  
Stellen Sie die Priorität des Prozesses ein, mit der er im System laufen soll. Weitere Informationen zu diesem Thema finden Sie in Kapitel 6.1.1 „Prozessarten“.  
  
Der Eintrag **Level** (-10...+10) definiert die Priorität innerhalb der *niederpriorigen* Prozesse, d.h. ein Prozess mit höherem **Level** kann solche mit niedrigem **Level** unterbrechen, nicht aber umgekehrt. Eine höhere Zahl steht für höhere Priorität.
- **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.
- `Number of Loops`: Die Einstellung ist nur bei den Prozessoren T2...T8 verfügbar.




## 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 ADbasic 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:

- **ADbasic-Syntax** (System related):
  - **ADbasic sections:** Die Kennwörter **Init:**, **LowInit:**, **Event:** und **Finish:** für die Programmabschnitte.
  - **Compiler Directives:** Befehle für den Pre-Compiler wie **#Define**, die mit **#** beginnen.
  - **Reserved Keywords:** Die Basis-Befehle in *ADbasic* wie **Dim**.
  - **Global Variables:** Die globalen Variablen **Par\_1 ... Par\_80**, **FPar\_1 ... FPar\_80** und **Data\_1 ... Data\_200**.
  - **External Keywords:** *ADbasic*-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 den Betriebssystem-Dateien `<*.btl>`, die beim Boot-Vorgang in das ADwin-System übertragen werden (siehe Kapitel 3.1.3 auf Seite 11).
- `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..

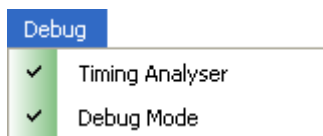
Wir empfehlen, die voreingestellten Verzeichnisse für BTL, Include und Library nicht zu verändern. Wenn Sie Include- und Library-Dateien aus anderen Verzeichnissen einbinden, können Sie im Einbinde-Befehl den vollständigen oder relativen Pfadnamen angeben.

## ADtools

Die Hilfsprogramme *ADtools* (Beschreibung siehe Kapitel 3.10 auf Seite 73) können über eine eigene Symbolleiste aufgerufen werden. Bei aktiver Option erscheint das jeweilige Hilfsprogramm in der *ADtools*-Leiste.

### 3.7.6 Das Menü Debug

Im Menü *Debug* stellen Sie Optionen ein, die Ihnen beim Auffinden von Laufzeit- oder Semantik-Fehlern helfen. Beachten Sie bitte, dass alle Einstellungen erst nach dem nächsten Kompilieren wirksam werden.



#### Option Timing Analyzer

Aktivieren Sie die Compiler-Option *Timing Analyser*, um nach dem Kompilieren eines Quelltexts zusätzliche Informationen zum Zeitverhalten des Prozesses verfügbar zu machen. Die Timing-Informationen werden im Fenster „Timing Analyzer“ (Seite 67) angezeigt.

Die Compiler-Option wird auch in der Statusleiste angezeigt, die Einstellung für einen laufenden Prozess dagegen im Prozessfenster.

Die Option benötigt pro Event-Zyklus und Prozess etwa 60 Taktzyklen (bei den Prozessoren T9, T10 und T11) zusätzlich und beeinflusst damit das Zeitverhalten geringfügig. Wir empfehlen daher, die Option nur für das Kompilieren eines oder weniger Prozesse zu aktivieren und sie dann wieder zu deaktivieren. Die Einstellung dieser Option wird beim Verlassen der Entwicklungsoberfläche nicht gespeichert.

#### Option Debug mode

Wenn Sie die Compiler-Option *Debug mode* im Menü *Debug* aktivieren und anschließend einen Quelltext kompilieren, werden zusätzliche Sicherheitsabfragen in den Prozess eingebaut (siehe auch Kapitel 5.3.1 auf Seite 113).

Die Compiler-Einstellung wird auch in der Statusleiste angezeigt, die Einstellung für einen laufenden Prozess dagegen im Prozessfenster.

Das Einschalten der Option verlängert die Programm-Ausführungszeit und vergrößert den Speicherbedarf. In der Regel liegt dies in einer Größenordnung von ca. 20 %, aber auch größere Werte sind möglich. Sie sollten daher diese Option nur während der Programmentwicklung nutzen.

Wenn ein Laufzeitfehler im ADwin-System auftritt, wird die Fehlermeldung im Fenster „Debug Errors“ angezeigt (unten im Info-Bereich), und zwar mit dem Zeitpunkt des ersten Auftretens. Beim ersten Auftreten eines Fehlers wird das Fenster automatisch in den Vordergrund gesetzt.

Ein Neustarten des Prozesses setzt die Fehlermeldungen nicht zurück, das Rücksetzen geschieht jedoch bei einem Neuladen des Prozesses.

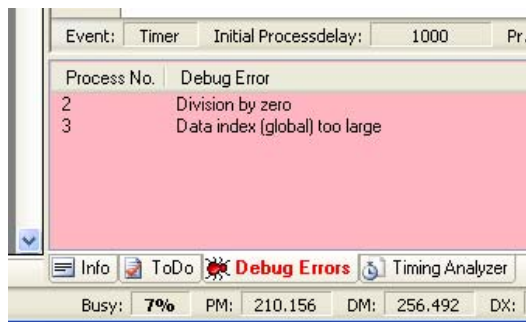


Abb. 6 – Das Fenster Debug Errors

Das Betriebssystem korrigiert einige Laufzeitfehler so, dass ein stabiler Betriebszustand erhalten bleibt; dies kann jedoch zu unerwarteten Programm-Ergebnissen führen. Bei manchen Laufzeitfehlern mit Pro II-Modulen wird der Prozess gestoppt.

Die folgende Tabelle zeigt, welche Fehler angezeigt werden und welche Korrektur dazu jeweils erfolgt. Die vollständige Liste der Debug-Fehlermeldungen – auch solche, bei denen keine Korrekturen vorgenommen werden – finden Sie im Anhang auf Seite A-18.

Fehlermeldung, Ursache	Korrekturmaßnahme
Division durch Null	Das Ergebnis einer Float-Division wird durch +3.40282E+38 ersetzt, das einer Long-Division durch +2147483647.
Wurzel aus negativer Zahl	Das Ergebnis des Wurzelziehens wird durch den Wert 0 ersetzt.

Fehlermeldung, Ursache	Korrekturmaßnahme
Data index zu groß / <1 Array index zu groß / <1 Zugriff auf nicht deklarierte Elemente eines lokalen oder globalen Felds, d.h. auf eine zu große oder zu kleine Elementnummer.	Eine zu kleine Elementnummer (<1) wird durch 1 ersetzt, eine zu große Elementnummer durch die größte dimensionierte Elementnummer.
FIFO-Index ist kein FIFO Das Feld mit dieser Nummer ist nicht als FIFO oder gar nicht deklariert.	Befehl ( <b>FIFO_Clear</b> , <b>FIFO_Full</b> , <b>FIFO_Empty</b> ) wird nicht ausgeführt.
Adresse des Pro II Moduls ist >15 oder <1	Der Prozess wird beendet.

Es wird für jeden Prozess nur ein einzelner Fehler angezeigt (in der Regel der zuletzt aufgetretene), auch wenn der Prozess mehrere Laufzeitfehler erzeugt.

Bitte beachten Sie: Beim Befehl **MemCpy** wird nur der Zugriff auf das Zielfeld geprüft und korrigiert; ein Zugriff auf nicht deklarierte Elemente des Quellfelds wird nicht erkannt.




---

1. Gilt für **P2\_Burst\_Init**, **P2\_Burst\_Read**, **P2\_Burst\_Write**

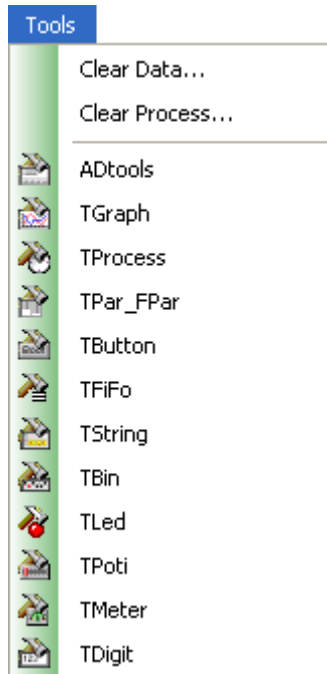
### 3.7.7 Das Menü „Tools“

Im Menü **Tools** rufen Sie Hilfsprogramme auf.

Der Menüeintrag **Clear Data** gibt den Speicher im angeschlossenen *ADwin*-System frei, der von einem bestimmten **DATA**-Feld belegt wird (Gegenstück zum Befehl **Dim**). Die im Feld enthaltenen Daten gehen damit verloren.

Geben Sie im folgenden Dialogfenster die Nummer des gewünschten **Data**-Felds ein, z.B. 3 für **Data\_3** und bestätigen Sie die Freigabe des Felds.

Der Menüeintrag **Clear Process** löscht einen bestimmten Prozess aus dem Speicher. Beachten Sie, dass ein Prozess erst gelöscht werden kann, wenn Sie ihn vorher gestoppt haben.



Die Menüeinträge **ADtools** und folgende starten jeweils ein Hilfsprogramm. Die Programme sind in Kapitel 3.10 auf Seite 73 kurz beschrieben.

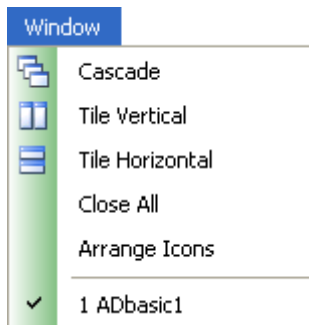


## 3.7.8 Das Menü „Window“

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.

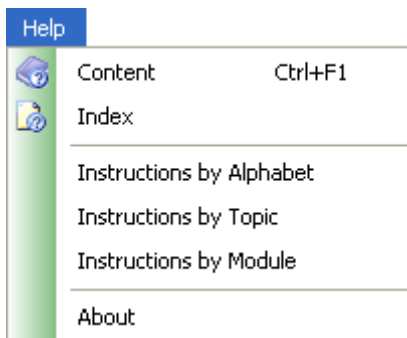
Unten im Menü können Sie über die Dateinamen einen der geöffneten Quelltexte zum aktiven Fenster machen. Der aktive Quelltext ist mit einem Haken gekennzeichnet; im Beispiel rechts ist dies `ADbasic1.bas`.




## 3.7.9 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 43 eingestellt ist.

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

`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 *ADbasic* 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.

## 3.8 Fenster

Der Info-Bereich wird separat behandelt, siehe Seite 64.

### 3.8.1 Toolbox

Die Toolbox ist der Bereich in der Oberfläche links, in dem das Projektfenster, das Parameterfenster 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.

### 3.8.2 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 58).

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.

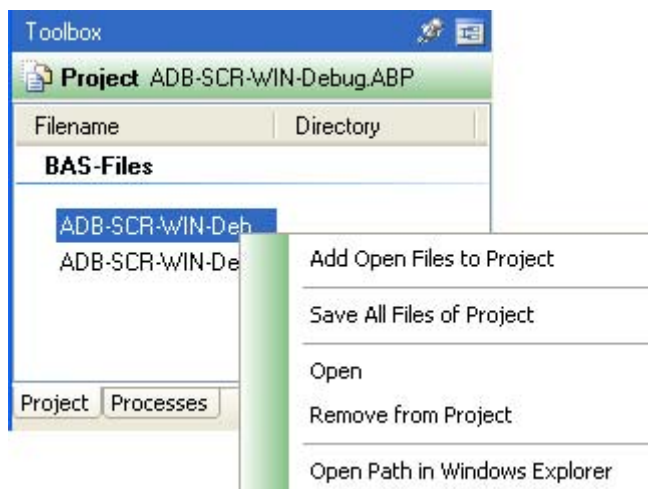



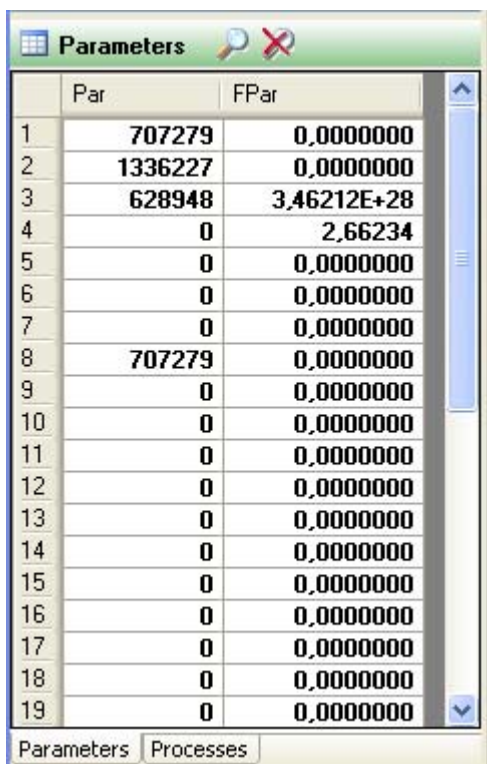
Abb. 7 – Das Projekt-Fenster mit Kontextmenü

### 3.8.3 Parameterfenster

Das Parameterfenster zeigt die globalen Parameter **Par\_1...Par\_80** und **FPar\_1...FPar\_80** an. Mit dem Schieberegler am rechten Rand können Sie den angezeigten Parameter-Bereich auswählen.

Das Parameterfenster ist Teil der Toolbox (siehe Seite 58).


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	0,0000000
2	1336227	0,0000000
3	628948	3,46212E+28
4	0	2,66234
5	0	0,0000000
6	0	0,0000000
7	0	0,0000000
8	707279	0,0000000
9	0	0,0000000
10	0	0,0000000
11	0	0,0000000
12	0	0,0000000
13	0	0,0000000
14	0	0,0000000
15	0	0,0000000
16	0	0,0000000
17	0	0,0000000
18	0	0,0000000
19	0	0,0000000


Abb. 8 – Das Parameter-Fenster

Sie können die Werteanzeige eines ganzzahligen Parameters zwischen dezimal und hexadezimal umstellen (siehe **Par\_5** in Abb. 8), 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 38).

### 3.8.4 Prozessfenster

Das Prozessfenster zeigt Informationen über die Prozesse 1...10 auf einem ADwin-System 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 58). Sie öffnen das Prozessfenster mit einem Klick auf den Reiter `Processes`.

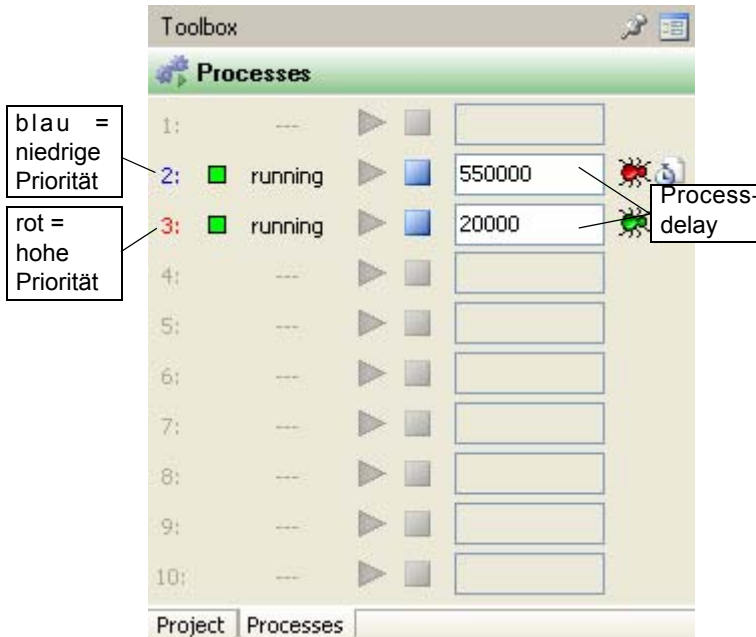




Abb. 9 – Das Prozess-Fenster

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 auf das ADwin-System übertragen. Achten Sie darauf, das System 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 122 erläutert.

- Prozess arbeitet im Debug-Modus

Das Icon wird angezeigt, wenn der Prozess im Debug-Modus arbeitet. Näheres zum Debug-Modus ist unter Option Debug mode beschrieben.

Die Compiler-Einstellung zum Debug-Modus wird in der Statusleiste angezeigt.

- Prozess arbeitet im Timing-Modus

Das Icon wird angezeigt, wenn der Prozess im Timing-Modus arbeitet. Näheres zum Timing-Modus ist unter der Option Timing Analyzer beschrieben (Seite 53). Die Timing-Informationen werden im Fenster „Timing Analyzer“ angezeigt.

Die Compiler-Einstellung zum Timing-Modus wird in der Statusleiste angezeigt.

### 3.8.5 Statusleiste

Die Statusleiste befindet sich am unteren Rand der Bedienoberfläche.



- Links: Informationen zur zuletzt ausgeführten Aktion.
- Mitte: Die aktuelle Prozessor- und Speicherauslastung des *ADwin*-Systems (bei Verbindung zwischen PC und *ADwin*-System).
- 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:** Freier Programmspeicher in Bytes.

**EM:** Freier Zusatzspeicher in Bytes (nur für T11).

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

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

### 3.9 Info-Bereich

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

- Infofenster
- ToDo-Liste
- Das Fenster `Debug Errors`
- Fenster „Timing Analyzer“
- Fenster „Global Variables“
- Fenster „Declarations“



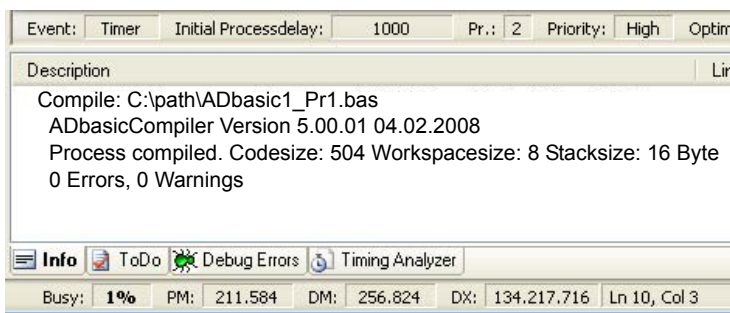
## 3.9.1 Infofenster

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).
- `Stacksize`: Größe des internen Stapels, der für Libraries verwendet wird.

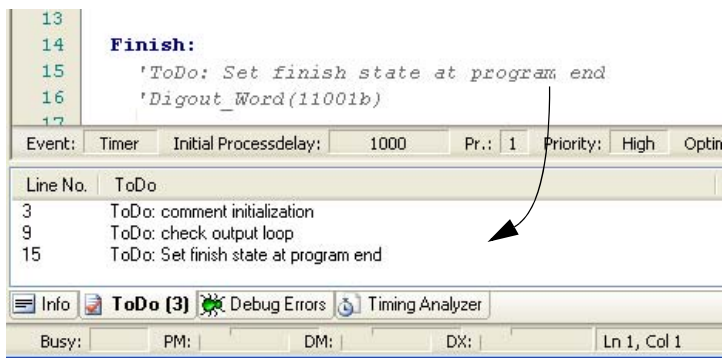
Es wird nicht angezeigt, wieviel Speicherplatz im externen Datenspeicher (DX) benötigt wird.

### 3.9.2 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 64).



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

### 3.9.3 Fenster „Timing Analyzer“

Bei aktivem Timing-Modus zeigt das Fenster **Timing Analyzer** 7 Kennwerte, die das Zeitverhalten der Prozesse 1...10 seit dem Zeitpunkt des letzten Starts beschreiben. Nähere Hinweise über die Auswertung dieser Informationen gibt Ihnen Kapitel 5.3.2 „Zeitverhalten prüfen (Timing-Modus)“.

Sie aktivieren den Timing-Modus vor dem Kompilieren im Menü **Debug** mit der Option **Timing Analyzer** (Seite 53).

Das Fenster **Timing Analyzer** ist Teil des Info-Bereichs (siehe Seite 64).

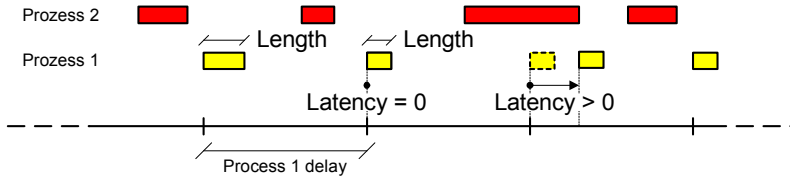
Alle Zeitangaben sind in Taktzyklen des Prozessors angegeben (Einheiten siehe Abb. 16 auf Seite 122).

Die Kennwerte sind nur für hoch priorisierte Prozesse verwendbar. Bei einem extern gesteuerten Prozess sind die Werte in den Zeilen 4...6 nicht sinnvoll und werden als 0 (Null) angezeigt.

Event:	Timer	Initial Processdelay:	1000	Pr.: 1	Priority:	High	Optimize Level:	1
	Process 1	Process 2	Process 3	Process 4	Process 5	Process 6	Process 7	Process 8
min. Length		11	27					
max. Length		11	33					
⌘ Length		11.0	27.1					
max. Latency		0	57					
max(Latency + Length)		11	84					
count (Length > Delay)		0	0					
Critical timings		0	0					

Abb. 10 – Das Fenster **Timing Analyzer**

**Length** bezeichnet die Dauer eines Prozesszyklus (Abschnitt **Event:**); diese Bearbeitungsdauer kann man auch ermitteln wie in Kapitel 5.1 „Bearbeitungszeit messen“ beschrieben. **Latency** ist die Zeitdauer zwischen dem Auftreten eines Event-Signals (extern oder vom internen Timer) und dem tatsächlichen Start des Prozesszyklus, im Bild am zeitgesteuerten Prozess 1 dargestellt.



Im Fenster werden folgende Kennwerte angezeigt:

- min. Length: Minimale gemessene Dauer eines Prozesszyklus
- max. Length: Maximale gemessene Dauer eines Prozesszyklus
- Ø Length: Durchschnittliche Dauer des Prozesszyklus

Der Durchschnitt wird berechnet als gleitender Mittelwert aus vorhergehenden length-Werten.

$$\text{ØLength} = 0,999 \cdot \text{ØLength} + 0,001 \cdot \text{Length}$$

Nach dem Start eines Prozesses dauert es 7000 Zyklen, bis der Mittelwert einen gültigen Wert erreicht.

Gemeinsam mit min. Length und max. Length zeigt dieser der Mittelwert, wie groß und wie gleichmäßig die Bearbeitungsdauer eines Prozesszyklus ist. Unterschiedliche Bearbeitungsdauern kommen zustande, wenn große Datenmengen erst nach längerer Zeit ausgewertet werden oder wenn fallweise Unterscheidungen (If, Case) verschieden lang dauernde Programmabschnitte (Schleifen) beinhalten.

- max. Latency: Größte gemessene Startverzögerung eines Prozesszyklus; nur für zeitgesteuerte Prozesse verfügbar.

Eine Startverzögerung entsteht, wenn beim Auftreten eines Event-Signals gerade ein hochpriorer Prozess bearbeitet wird. Dies tritt auf, wenn die Bearbeitungsdauer eines Prozesszyklus das (zu diesem Prozess gehörende) Processdelay überschreitet. Bei mehreren hochprioreren Prozessen sind gelegentliche Startverzögerungen unvermeidbar, es sei denn, deren Processdelays sind ganzzahlige Vielfache voneinander.

Die Summe über alle Verzögerungen sollte im Mittel immer 0 sein; dies entspricht dem Einhalten einer mittleren Frequenz. Daneben ist der

Kennwert wichtig für Prozesse, deren Prozesszyklen sehr genau zum vorbestimmten Zeitpunkt ablaufen müssen.

- `max. (Latency+Length)`: Maximum aus der Summe von Startverzögerung und Dauer eines Prozesszyklus; nur für zeitgesteuerte Prozesse verfügbar.

Für ein optimales Zeitverhalten sollte dieser Wert kleiner als das `Processdelay` sein; wenn dies eingehalten werden kann, verursacht der Prozess keine Startverzögerung bei sich selbst (bei anderen Prozessen ist das aber möglich).

- `count (Length > Delay)`: Wert, der angibt, wie oft ein Prozesszyklus das `Processdelay` überschritten hat; nur für zeitgesteuerte Prozesse verfügbar. Dieser Wert sollte möglichst Null sein.

Je größer dieser Wert ist, umso häufiger hat der Prozess eine Startverzögerung bei sich selbst (und vielleicht auch bei anderen Prozessen) hervorgerufen. Das Betriebssystem versucht kontinuierlich, diese Verzögerungen wieder aufzuholen. Über den Verlust von Events sagt die Anzahl der Überschreitungen dagegen nichts aus.

- `Critical timings`: Anzahl des Zutreffens einer Bedingung, die ein möglicherweise verlorenes Event-Signal bedeutet. Dieser Wert sollte unbedingt Null sein.

Dieser Kennwert hat je nach Art und Anzahl der Prozesse eine unterschiedliche Bedeutung (siehe auch Kapitel 6.2.5 „Verschiedene Betriebszustände im Betriebssystem“, Seite 126).

Event-Signale können verloren gehen bei:

- einem einzigen zeitgesteuerten, hochpriorien Prozess (auch in Kombination mit dem extern gesteuerten Prozess)
- dem extern gesteuerten Prozess (auch in Kombination mit einem oder mehreren zeitgesteuerten Prozessen)

Bei mehreren zeitgesteuerten Prozessen können Event-Signale *nicht* verloren gehen, das Eintreten der nachstehenden Bedingung wird aber dennoch gezählt. Hier ist der Kennwert zu interpretieren als Auftreten eines sehr schlechten Zeitverhaltens, das auf jeden Fall verbessert werden muss.

Eine Anzahl verlorener Event-Signale bedeutet, dass (seit dem letzten Start des Prozesses) weniger Prozesszyklen ausgeführt wurden als Event-Signale aufgetreten sind, nämlich wahrscheinlich um die ange-

gebene Anzahl weniger. Dieser Verlust kann vom Betriebssystem nicht ausgeglichen werden.


Ein Event-Verlust wird gleich gesetzt mit dem Eintreten der Bedingung:

- bei zeitgesteuerten Prozessen:  
 $\text{max. latency} + \text{length} > 2 \times \text{Processdelay}$
- bei extern gesteuerten Prozessen:  
Wenn die Bearbeitung des Abschnitts **Event**: gerade beendet ist, steht schon ein neues externes Event-Signal an. Falls während dieser Bearbeitungszeit noch weitere Event-Signale aufgetreten sind, sind diese weiteren Events verloren.

Manchmal ist es möglich, dass trotz zutreffender Bedingung *kein* Event verloren geht. Daher liegen Sie auf der sicheren Seite, wenn Sie die Anzahl zutreffender Bedingungen so weit wie möglich reduzieren.

### 3.9.4 Fenster „Global Variables“

Das Fenster `Global Variables` zeigt an, welche globalen Variablen (`Par_1 ... Par_80`, `FPar_1 ... FPar_80`) und Felder (`Data_1 ... Data_200`) in einem Quelltext oder in einem Projekt verwendet werden.

Um die Anzeige zu starten oder zu aktualisieren, müssen Sie die Schaltfläche `Scan Global Variables`  im Parameterfenster drücken (siehe Verwendete globale Variablen und Felder anzeigen, Seite 38).

Das Fenster ist Teil des Info-Bereichs (siehe Seite 64).

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.

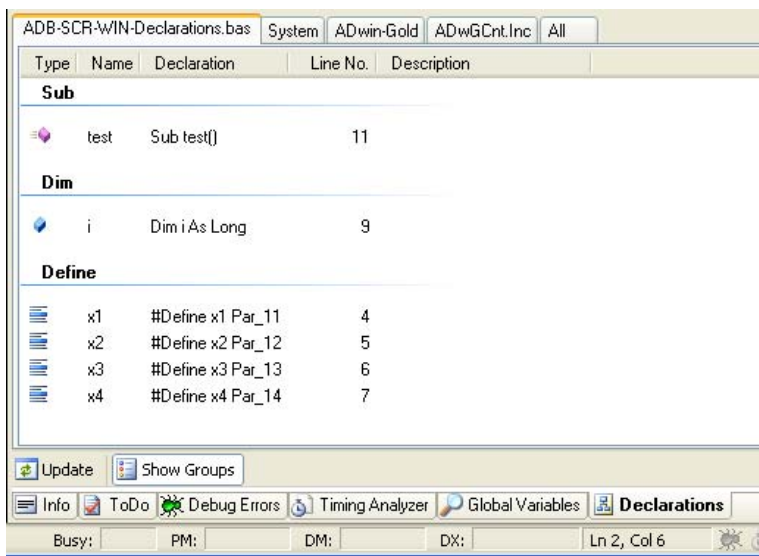
Wenn Sie den Quelltext ändern, wird das Fenster nicht automatisch aktualisiert. Verwenden Sie dazu Schaltfläche **Scan Global Variables**  im Parameterfenster.

### 3.9.5 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 64).



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 *ADbasic* implementiert sind und die zu den aktuellen Compiler-Einstellungen passen.

Die globalen Variablen **PAR** und **FPAR** werden nicht angezeigt. Beachten Sie hierzu auch das Fenster „Global Variables“ (Seite 70) und die



Funktion „Verwendete globale Variablen und Felder anzeigen“ (Seite 38).

- `ADwin-Gold`, `ADwin-light-16`: Befehle für Hardware-Zugriffe, die in *ADbasic* 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 49) aktiv ist.

## 3.10 ADtools

*ADtools* ist eine Sammlung kleiner Hilfsprogramme, mit denen Sie die globalen Variablen (**Par**, **FPar**) und Felder (**Data**) von *ADwin*-Systemen und *TiCo*-Prozessoren anzeigen und auch ändern können. Die Programme unterstützen Sie beim Entwickeln von Prozessen, indem sie z.B. Werte und Zustände anzeigen, diese mit praktischen Werkzeugen verändern oder einfache Messwert-Verläufe darstellen.

Starten Sie die *ADtools* einfach aus der senkrechten Leiste am rechten Rand. 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:



TDigit

ermöglicht das Anzeigen und Eingeben von Werten für globale Variablen und Felder.



TGraph

stellt den Inhalt globaler Felder in Kurvenform dar.



TButton

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



TLed

zeigt einen Variablenwert an durch Leuchten, Blinken, Flackern oder akustischen Alarm und kann zur Überwachung dienen.



TMeter

Analog-Zeigerinstrument zur Anzeige von globalen Variablen und Feldern.



TPoti

Potentiometer zur Anzeige und zum Verändern von globalen Variablen und Feldern.



TProcess

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



TPar\_FPar

ermöglicht das Anzeigen und Eingeben von allen oder ausgewählten Variablen.



TFIFO

ermöglicht das kontinuierliche Speichern von Daten eines FIFO-Feldes in eine Datei.



TBin

zeigt bis zu 5 ganzzahlige Variablen binär (als DIL-Schalter) und hexadezimal an. Variablenwerte können auch geändert werden.



TString

stellt den Inhalt von bis zu 5 String-Feldern dar.



ADtools

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

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 *ADbasic*-Programm aufbauen, strukturieren und welche Variablen Ihnen dabei zur Verfügung stehen.

### 4.1 Programmaufbau

Sie geben ein *ADbasic*-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 Ihr spezielles *ADwin*-System.

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

*ADbasic* 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 4 Abschnitten, die bei der Ausführung auf dem *ADwin*-System unterschiedliche Aufgaben übernehmen, sowie aus den erforderlichen Deklarationen. Halten Sie die Reihenfolge der Abb. 11 beim Programmaufbau ein. 

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

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

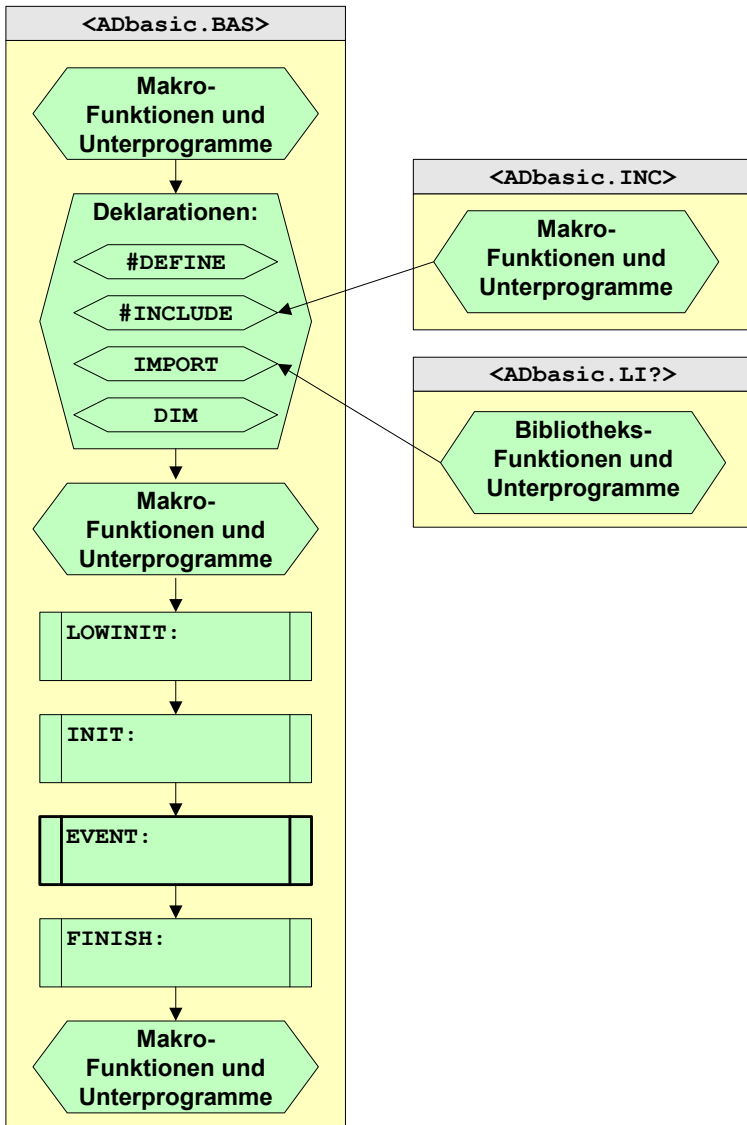



Abb. 11 – Aufbau eines ADbasic-Programms

### 4.1.1 Die Programmabschnitte


Die Programmabschnitte (siehe Abb. 11) beginnen jeweils mit einem Kennwort.


- **LowInit:** ist nur bei hochprioren Prozessen einsetzbar.

Dieser Abschnitt wird bei jedem Start des Prozesses einmalig durchlaufen und dient zur Initialisierung, z.B. von Variablen oder Datenleitungen. Er wird immer vor dem Abschnitt **Init:** ausgeführt (falls vorhanden) und immer mit niedriger Priorität, Stufe 1.

Dieser Abschnitt ist für umfangreiche Initialisierungs-Sequenzen geeignet, da er (wegen seiner niedriger Priorität) unterbrochen werden kann. 

- **Init:** entspricht dem Abschnitt **LowInit:** (Initialisierung, einmaliger Durchlauf). Er wird aber mit der von Ihnen eingestellten Priorität bearbeitet (Menüpunkt „Options / Process“).

Dieser Abschnitt kann, wenn Sie hohe Priorität eingestellt haben, nicht unterbrochen werden und sollte dann möglichst kurz sein. 

- **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. Dieser Abschnitt wird immer mit niedriger Priorität, Stufe 1 ausgeführt. 

Die Abschnitte **LowInit:**, **Init:** und **Finish:** sind optional, der Abschnitt **Event:** muss immer vorhanden sein.

### 4.1.2 Benutzerdefinierte Befehle und Variablen

#### Symbolische Namen

Mit der Anweisung **#Define** können Sie symbolische Namen definieren (siehe Seite 162). 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 *ADbasic* müssen Sie nur lokale Variablen und alle Felder vor der Benutzung mit `Dim` deklarieren (siehe Seite 164). Die globalen Variablen `Par_n` und `FPar_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 vom PC 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 100). Makros müssen in jedem Fall außerhalb der Programmabschnitte definiert werden (siehe Abb. 11 auf Seite 76).

## 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 101).

## 4.2 Variablen und Felder

### 4.2.1 Übersicht

Datenstruktur	Name	Datentyp	Bemerkung
Globale Variablen und Felder			
Variable (Skalar)	Par_1...Par_80	Long	Vordefiniert, nicht deklarierbar, Speicherbereich DM
	FPar_1...FPar_80	Float	
System-Variable	Processdelay	Long	nicht deklarierbar, Speicherbereich DM
	Process_Error	Long	
	Processn_running	Long	
Ein- oder zweidi- mensionales Feld (Vektor)	Data_1 [] [] ...	Long, Float,	Name Data_ nicht änderbar, nur Deklara- tion von Feldnummer und Dimension.
	Data_200 [] []	String, FIFO	
Lokale Variablen und Felder			
Variable (Skalar)	frei wählbar	Long, Float	muss deklariert wer- den
Eindimensiona- les Feld (Vektor)	frei wählbar	Long, Float, String	muss deklariert wer- den

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

Alle Datentypen haben eine Länge von 32 Bit.

### 4.2.2 Datenstrukturen

In *ADbasic* 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 4.3.4 auf Seite 90).

Die maximale Anzahl an Variablen bzw. die Größe eines Felds ist nur durch die Speichergröße des ADwin-Systems begrenzt.

Der Compiler unterscheidet

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

Auf globale Datenstrukturen können sowohl alle Prozesse als auch PC-Anwendungen zugreifen, z. B. zum Austausch von Daten.

System-Variablen zählen zu den globalen Variablen (siehe Seite 85).

- Lokale Variablen und Felder (siehe Seite 86):

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** und **FPar\_1 ... FPar\_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...200).



Variablen und Feldelemente haben nach der Deklaration keinen definierten Wert und sollten deshalb mit einem sinnvollen Wert (z. B. Null) initialisiert werden. Ausnahme: Nach dem Einschalten des ADwin-Systems werden globale Variablen automatisch mit Null initialisiert.

### 4.2.3 Datentypen

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



Der Compiler verarbeitet folgende Datentypen:

- **LONG** Ganzzahliger 32 Bit-Wert mit dem Wertebereich:  
 $-2^{147483648} \dots +2^{147483647} = -2^{31} \dots +2^{31}-1$
- **FLOAT** Fließkomma-Wert (32 Bit) mit dem Wertebereich:  
 $-3,402823 \cdot 10^{+38} \dots -1,175494 \cdot 10^{-38}$  (negative Werte, 32 Bit)  
 $+1,175494 \cdot 10^{-38} \dots +3,402823 \cdot 10^{+38}$  (positive Werte, 32 Bit)  
 Der Wertebereich entspricht nicht dem „IEEE-Floating point“-Format.
- **FLOAT** ab T11: Fließkomma-Wert (40 Bit) mit dem Wertebereich:  
 $-3,402823668 \cdot 10^{+38} \dots -1,175494351 \cdot 10^{-38}$  (negative Werte, 40Bit)  
 $+1,175494351 \cdot 10^{-38} \dots +3,402823669 \cdot 10^{+38}$  (positive Werte, 40 Bit)  
 Der Wertebereich entspricht nicht dem „IEEE-Floating point“-Format.  
 Die Rechengenauigkeit 40 Bit gilt ausschließlich in folgenden Bereichen:

- Berechnungen innerhalb des ADwin-Systems.
- Auswertung von Konstanten durch den Compiler.

Die 40 Bit-Genauigkeit kann auf dem PC nicht genutzt oder angezeigt werden, weil zwischen PC und ADwin-System – aus Gründen der Geschwindigkeit – nur 32 Bit-Werte übertragen werden.

Im Speicher belegt eine 40 Bit-Float-Variable 64 Bit.

- **STRING** ASCII-Zeichenfolge, die in Feldern so gespeichert wird, dass jedes Feldelement ein Zeichen enthält (Details siehe Kapitel 4.3.5 auf Seite 93). Ein einzelnes Zeichen entspricht einem ganzzahligen 8 Bit-Wert im Bereich 0...255.

Die veralteten Datentypen **Short** und **Integer** – bei Prozessoren vor T9 verwendet – wurden durch den Datentyp **Long** ersetzt. Aus Gründen der Kompatibilität akzeptiert der Compiler die Datentypen weiterhin, ersetzt sie aber durch **Long**.

Bei der Kombination von ganzzahligen und Fließkomma-Werten kommt es zu einer Typkonvertierung. Dies kann unter bestimmten Umständen zu einem anderen als dem erwarteten Berechnungsergebnis führen. Näheres finden Sie im Abschnitt „Typkonvertierung“ auf Seite 98.

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 angeben möchten. Die folgenden Beispiele weisen einer Variablen `x` den Wert 930 zu.



Bei Fließkomma-Werten wird der Punkt „.“ als Dezimaltrennzeichen verwendet (englische Schreibweise).

1. Dezimale Schreibweise:

`x = 930`

LONG

`x = 930.0`

Float



Beachten Sie den Unterschied: Die Zahl „930“ hat den Datentyp `Long`, die Zahl „930.0“ dagegen den Datentyp `Float`. Dies ist wichtig, wenn Sie beide Datentypen in einem Berechnungsausdruck miteinander verwenden (siehe Kapitel 4.4.2).

2. Exponential-Schreibweise:

`x = 93E1`

LONG

`x = 9.3E2`

Float

Hierbei steht „9.3E2“ für  $9,3 \times 10^2$ , d.h. nach dem „E“ folgt der (max. 2-stellige) Exponent zur Basis 10.

3. Binäre Schreibweise (angehängtes „b“):

`x = 111010001b`

LONG

4. Hexadezimale Schreibweise (angehängtes „h“):

`x = 3A2h`

LONG

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 Prozesse und der PC zugreifen; daher eignen sie sich gut zum Datenaustausch zwischen den Prozessen oder zwischen den Prozessen und dem PC (siehe auch Kapitel 6.3.1 „Datenaustausch zwischen Prozessen“). Ihnen stehen 80 ganzzahlige

Variablen, 80 Fließkomma-Variablen sowie bis zu 200 Felder (Arrays) vom Datentyp **Long** oder **Float** zur Verfügung. Alle Variablen und Feldelemente haben eine Länge von 32 Bit.

Die ebenfalls global verfügbaren System-Variablen sind auf Seite 85 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: Nach dem Einschalten des ADwin-Systems werden globale Variablen 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.
- **FPar\_1, FPar\_2, ..., FPar\_80** mit dem Datentyp **Float** für Fließkommawerte.

### Beispiel

```
Par_5 = 700           'Parameter 5 erhält den
                      'Wert 700.
Par_72 = ADC(1)       '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** und **FPar\_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 dem PC auszutauschen (siehe auch Kapitel 6.3.1 „Datenaustausch zwischen Prozessen“). Ihnen stehen bis zu 200 globale Felder (Arrays) vom Datentyp **Long** oder **Float** 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...200). Die Namen für Data-Felder sind also:

**Data\_1, Data\_2, ..., Data\_200.**

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
REM Feld 3 mit mit 7*5 Elementen vom Typ Float
REM deklarieren.
Dim Data_3[7][5] As Float
```

Näheres zu 2-dimensionalen Feldern steht in Kapitel 4.3.3 auf Seite 89.

Die maximale Größe der Felder richtet sich nur nach dem verfügbaren Speicherplatz. Beispielsweise kann auf einem ADwin-System mit 16MiB Speicher ein Feld mit bis zu 4 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

Rem Diese Anweisung weist dem 1. (!) Element aus dem
Rem 2dimensionalen Feld Data_3 den Wert 300 zu.
Data_3[1][1] = 300
```

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 *ADbasic*-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 *ADwin*-Systems zu erhalten, stehen Ihnen die folgenden System-Variablen zur Verfügung. Diese Variablen sind global, d.h. für alle -Prozesse und vom PC 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 238). Die Variable kann nur gelesen werden.

#### **Process\_Error**

Zeigt bei aktiviertem Debug-Modus den zuletzt aufgetretenen Fehler des Prozesses *n* an (mit *n* = 1...16, siehe Seite 237). Die Variable kann nur gelesen werden.

#### **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 234) 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, weil sonst das ADwin-System in einen instabilen Zustand geraten kann.



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 ADbasic-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**) und Fließkomma-Werte (**Float**) verfügbar, jeweils in 32 Bit.



#### Beispiel

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

Dim wert1, wert2 As Float 'Definiert die
                           'Variablen 'wert1' und
                           ''wert2'
                           'mit dem Datentyp Float
```

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 Float 'Definiert ein Feld der
                       'Länge 100 mit Namen 'wert'
                       'und dem Datentyp Float
```



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 DDrAm_Extern`. Bei dem Prozessor T11 ist ein zusätzlicher Speicherbereich mit `At EM_Local` verfügbar.

Wenn Sie bei der Deklaration keinen Zusatz verwenden, werden Variablen im internen Speicher DM angelegt sowie Felder im externen Speicher DX.

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 wegen seiner Größe vorwiegend für Felder geeignet.

In Abb. 12 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 &lt;VarType&gt;</code> oder <code>Dim var As ... At DM_Local</code>
	Zusatz (EM)	<code>Dim var As ... At EM_Local</code>
	extern (DX)	<code>Dim var As ... At DDrAm_Extern</code>
Feld	intern (DM)	<code>Dim array[5] As ... At DM_Local</code>
	Zusatz (EM)	<code>Dim array[5] As ... At EM_Local</code>
(global oder lokal)	extern (DX)	<code>Dim array[5] As ...</code> oder <code>Dim array[5] As ... At DDrAm_Extern</code>

Abb. 12 – Festlegung des Speicherbereichs bei Deklarationen

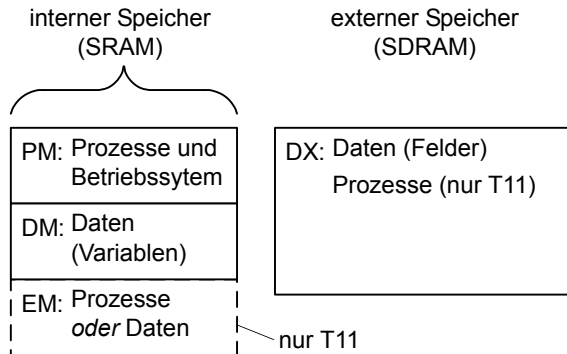
Die globalen Variablen `Par_1...Par_80` und `FPar_1...FPar_80` sind vordefiniert und stehen immer im internen Speicher DM zur Verfügung. Sie brauchen und können diese daher nicht neu (z.B. für den externen Speicher) deklarieren.



### 4.3.2 Speicherbereiche

Der Prozessor des ADwin-Systems verwendet einen schnellen internen Speicher (SRAM) und einen großen externen Speicher (SDRAM).

Je die Hälfte des internen Speichers steht als Programmspeicher PM und als Datenspeicher DM zur Verfügung. Der Prozessor T11 besitzt außerdem einen internen Zusatzspeicher EM, der entweder als Programm- oder als Datenspeicher genutzt werden kann.



#### – 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 auf).

#### – Zusatzspeicher intern (EM)

Der interne Zusatzspeicher EM ist nur beim Prozessor T11 vorhanden. Der Zusatzspeicher kann entweder als Datenspeicher oder als Programmspeicher genutzt werden.

#### – Externer Speicher (DX, SX)

Der externe Speicher belegt das externe SDRAM und nimmt die globalen und lokalen Felder auf.

Beim T11 kann der externe Speicher auch Prozesse bis zu einem Megabyte Größe aufnehmen.



Sie können auf Daten im internen Speicher DM deutlich schneller zugreifen (etwa Faktor fünf) als auf Daten im externen Speicher DX.

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, EM und DX noch zur Verfügung steht (angegeben in Bytes).

### 4.3.3 2-dimensionale Felder

Globale Felder `Data_n` können mit 1 oder 2 Dimensionen deklariert werden. Die grundsätzlichen Eigenschaften sind in Kapitel 4.2.6 „Globale Felder (Arrays)“ auf Seite 83 beschrieben.

Die 2-dimensionale Schreibweise kann (im Vergleich zu 1-dimensionalen Feldern) eine Problemlösung vereinfachen. Gleichzeitig aber verlangsamt sie den Datenzugriff und sie erfordert zusätzlichen Programmspeicher.



Der Geschwindigkeitsverlust und der zusätzliche Speicherbedarf sind umso größer, je öfter ein Programm auf 2-dimensionale Felder zugreift.

In folgenden Fällen ist es erforderlich, auf die Daten eines 2-dimensionalen Felds so zuzugreifen, als wäre es 1-dimensional deklariert:

- Im PC, wenn die Daten eines 2D-Felds vom *ADwin*-System dorthin übertragen und verarbeitet werden.  
Umgekehrt können Daten eines 1D-Felds vom PC ins *ADwin*-System übertragen werden, auch wenn das Zielfeld in *ADbasic* 2-dimensional deklariert ist.
- Innerhalb eines Library-Moduls (`Lib_Sub`, `Lib_Function`), dem ein 2D-Feld als Argument übergeben wird.

Bei der beschriebenen Art des Zugriffs ist die Reihenfolge der Daten im Speicher von Bedeutung. Als Beispiel sei ein 2D-Feld deklariert mit

```
Dim Data_1[3][2] As Float
```

Im Datenspeicher werden diese 3×2 Elemente linear abgelegt. Die folgende Tabelle zeigt, mit welchem Element-Index der 1D-Zugriff auf die Elemente des Beispiel-Felds erfolgt.

Feld-Index 2D	[1][1]	[1][2]	[2][1]	[2][2]	[3][1]	[3][2]
Feld-Index 1D	[1]	[2]	[3]	[4]	[5]	[6]
Speicherstelle	n	n+1	n+2	n+3	n+4	n+5

Dem Element **Data\_1[3][1]**, das im Hauptprogramm verwendet wird, würde z.B. in einem Library-Modul das 5. Element des übergebenen Felds entsprechen:

```

Rem Hauptprogramm
Data_1[3][1] = 17
setpar1(Data_1)           'setzt Par_1 = 17

Rem Library-Modul
Lib_Sub setpar1(BYREF array[] As Long)
    Par_1 = array[5]       'entspricht Data_1[3][1]
Lib_EndSub

```

Beachten Sie bitte: Diese Art des Zugriffs ist nur in den oben genannten beiden Fällen zulässig. In allen anderen Fällen muss die 2-dimensionale Schreibweise angewendet werden.



Für die Zuordnung von 2D-Elementen zu 1D-Elementen gilt allgemein:

$$\text{DATA\_n}[i][j] \hat{=} \text{DATA\_n}[s \cdot (i - 1) + j]$$

wobei **s** die 2. Dimension von **Data\_n** bei der Deklaration ist. Im obigen Beispiel ist **s = 2**.

#### 4.3.4 Die Datenstruktur FIFO

Für Anwendungen, in denen große Datenmengen kontinuierlich übertragen werden sollen, empfehlen wir die Verwendung eines globalen Felds **Data\_n** mit der Datenstruktur FIFO: Ein Ringspeicher, der nach dem Prinzip „First In, First Out“ verwaltet wird.



Ein FIFO ist nicht zu verwechseln mit der Datenstruktur Ringspeicher im *TiCo*-Prozessor ([ringbuffer](#)). Der *TiCo*-Ringspeicher ist im Handbuch *TiCo-Basic* beschrieben.

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).

Sie können nur eindimensionale globale Felder (`Data_n`) als Ringspeicher deklarieren. Mögliche Datentypen sind `Long` oder `Float`.

**Beispiel**

```
Dim Data_5[1003] As Long As FIFO
```

Diese Anweisung deklariert das globale Feld mit der Nummer 5 als FIFO-Ringspeicher mit 1003 Elementen vom Typ `Long`. Beachten Sie beim T11 die besondere Wahl der Feldgröße, siehe Seite 173.



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

Da ein FIFO-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.

Nach der Deklaration eines FIFO-Felds sollten Sie die Zeiger mit dem Befehl `FIFO_Clear` initialisieren.



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

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

**Beispiel**

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

Um sicherzustellen, dass noch Platz im FIFO ist, sollten Sie vor dem Schreiben die Funktion `FIFO_Empty` verwenden. In gleicher Weise prüft die Funktion `FIFO_Full` vor dem Lesen, ob noch nicht gelesene Werte vorhanden sind.

## Beispiel



```
Dim free,used,value1 As Long
Dim Data_1[1003] As Long As FIFO
Rem Sind noch Elemente zum Beschreiben frei?
free = FIFO_Empty(1)
If (free > 0) Then
    Data_1 = wert1
EndIf
Rem Können noch benutzte Elemente gelesen werden?
used = FIFO_Full(1)
If (used > 0) Then
    Par_7 = Data_1
EndIf
```

### 4.3.5 Strings

Mit Zeichenketten (Strings) lassen sich Steuerzeichen und Texte z.B. von anderen Prozessüberwachungs-Geräten über eine RS-232-Schnittstelle zum ADwin-System transferieren, umwandeln und verarbeiten.

Zur String-Verarbeitung stehen eine Reihe von Befehlen zu Verfügung:

<b>Asc</b>	ASCII-Nummer eines Zeichens bestimmen
<b>Chr</b>	Zeichen zu einer ASCII-Nummer bestimmen
<b>FloatToStr</b>	Float-Wert in einen String wandeln
<b>Float40ToStr</b>	40 Bit Float-Wert in einen String wandeln
<b>LongToStr</b>	Long-Wert in einen String wandeln
<b>StrComp</b>	2 Strings auf Gleichheit prüfen
<b>StrLeft</b>	Zeichen linksbündig aus einem String kopieren
<b>StrLen</b>	Länge eines Strings bestimmen
<b>StrMid</b>	Zeichenfolge aus einem String kopieren
<b>StrRight</b>	Zeichen rechtsbündig aus einem String kopieren
<b>ValF</b>	String in einen Float-Wert wandeln
<b>ValI</b>	String in einen Long-Wert wandeln
+ (String-Addition) Operator, um Strings aneinander zu hängen	

Für die meisten Befehle müssen Sie die Library-Datei <String.LI\*> einbinden (das Zeichen \* bezeichnet den Prozessortyp: 9 für T9, A für T10, B für

T11). Die Library-Datei finden Sie nach der Installation im Library-Verzeichnis (Standard: <C:\ADwin\ADbasic\LIB>).

Eine String-Variable hat einen ähnlichen Aufbau wie ein Feld, d. h. jedes Feld-element enthält ein Zeichen. Die Dimensionierung eines Strings für 5 Zeichen lautet:

```
Import String.LI9
Dim text[5] As String
```

Durch die Dimensionierung wird im Speicher ein Feld für den String reserviert, das wie folgt aufgebaut ist:

<code>text[1]</code>	Länge des Strings in Zeichen (5)
<code>text[2]</code>	Das 1. Zeichen
<code>text[3]</code>	Das 2. Zeichen
<code>text[4]</code>	Das 3. Zeichen
<code>text[5]</code>	Das 4. Zeichen
<code>text[6]</code>	Das 5. Zeichen
<code>text[7]</code>	Das String-Ende-Zeichen, abschl./term. Null (00h)

Jedes Element belegt im Speicher 4 Bytes. Das erste und das letzte Element des Strings reserviert der *ADbasic*-Compiler automatisch. Verwenden Sie keinesfalls das Element 0, hier also `text[0]`.

Nach der Dimensionierung sind die Elemente nicht initialisiert. Ihren Inhalt erhalten sie erst bei einer Zuweisung.

### Normale Zuweisung

Die übliche Zuweisung von Werten an eine String-Variable erfolgt durch eine Zeichenkette in doppelten Hochkommas (") und wird vorwiegend für Texte verwendet. Für jedes der Zeichen legt *ADbasic* im Speicher die entsprechende ASCII-Nummer ab (Tabelle im Anhang).



### Beispiel

```
text = "HALLO"
```

Element-Index	Speicherinhalt	Bedeutung
<code>text[1]</code>	<code>05h</code>	Länge des Strings in Zeichen (5)
<code>text[2]</code>	<code>48h</code>	ASCII-Nummer für "H"
<code>text[3]</code>	<code>41h</code>	ASCII-Nummer für "A"
<code>text[4]</code>	<code>4Ch</code>	ASCII-Nummer für "L"
<code>text[5]</code>	<code>4Ch</code>	ASCII-Nummer für "L"
<code>text[6]</code>	<code>4Fh</code>	ASCII-Nummer für "O"
<code>text[7]</code>	<code>00h</code>	String-Ende-Zeichen

Die Wertzuweisung in Hochkommas sollten Sie nur für die Zeichen mit den ASCII-Nummern `20h...7Fh` (= sichtbare Zeichen im einfachen ASCII-Zeichensatz, siehe auch Anhang) einsetzen, mit Ausnahme der folgenden Zeichen. Diese müssen Sie per Escape-Sequenz zuweisen (s.u.):

- einfaches Hochkomma ('): `\x27`
- doppeltes Hochkomma ("): `\x22`
- Backslash (\): `\x5C`

## Zuweisung per Escape-Sequenz

Wenn Sie nicht nur Texte verarbeiten, sondern auch Zahlenwerte oder Steuerzeichen in einen String einbinden möchten, sollten Sie dafür Escape-Sequenzen einsetzen. Mit jeder Escape-Sequenz übergeben Sie eine einzelne Zahl an den *ADbasic*-Compiler, der sie unverändert im Speicher ablegt.

Geben Sie die Escape-Sequenz innerhalb von doppelten Hochkommas in der Schreibweise `"\xhh"` an, wobei `hh` die zu übergebende Zahl in hexadezimaler Schreibweise ist (siehe ASCII-Zeichensatz im Anhang). Jede solche Sequenz besteht aus genau 4 Zeichen.

## Beispiel

```
text = "\x48\x41\x4C\x4C\x4F"
```



Der Speicherinhalt ist identisch mit dem vorherigen Beispiel.

Da Sie auf diese Weise einem String beliebige Zahlen von `00h` bis `FFh` hinzufügen können, eignen sich Escape-Sequenzen besonders für die Zuweisung nicht darstellbarer Zeichen (wie Line Feed, Carriage Return, ...).

Zusätzlich zur Schreibweise `\xhh` gibt es für häufig verwendete (Steuer-) Zeichen spezielle Escape-Sequenzen:

Sequenz	ASCII-Nummer	Bedeutung
<code>\\</code>	5C	Backslash (\)
<code>\t</code>	09	Tabulator (TAB)
<code>\n</code>	0A	Line Feed / Zeilenvorschub (LF)
<code>\r</code>	0D	Carriage Return / Wagenrücklauf (CR)

Sie können bei der Zuweisung von Werten an eine String-Variable die oben vorgestellten Schreibweisen beliebig kombinieren.



### Beispiel

```
text = "HA\x4C\x40"
```

Der Speicherinhalt ist wieder identisch mit dem bekannten Beispiel.



Sie dürfen in einen String kein String-Ende-Zeichen - z. B. mit der Escape-Sequenz `\x00` - einfügen (Beispiel: `text = "HA\x00LLO"`). Der *ADbasic*-Compiler verarbeitet dies zwar korrekt, jedoch können sich Fehler bei der weiteren Verarbeitung des Strings ergeben.

### Nicht empfohlene Arten der Zuweisung

Sie können auch Zeichen mit den ASCII-Nummern `00h...1Fh` oder `80h...FFh` in eine String-Zuweisung einfügen, z. B. Umlaute wie [ß], [Ö] oder [?], mit "copy and paste" aus anderen Anwendungen oder durch die Tastenkombination [ALT] + Nummer. Wir empfehlen stattdessen ausdrücklich die Zuweisung per Escape-Sequenz!

Der Compiler ist zwar in der Lage, solche Zeichen zu verarbeiten. Die Zeichen haben aber entweder keine eindeutige ASCII-Nummer (länderspezifisch) oder sie können - möglicherweise schon im Editor - unerwünschte Aktionen (Zeilenumbruch o.ä.) und Programmfehler verursachen.



Wir empfehlen Ihnen, jegliche Steuer- und Sonderzeichen nur als Escape-Sequenzen in einen String einzufügen.



## 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. 13).

Bitte beachten Sie, dass die automatische Typkonvertierung in manchen Fällen die Auswertung des Berechnungsausdrucks beeinflusst (siehe unten).

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

Abb. 13 – 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
```

#### 4.4.2 Typkonvertierung

Sie können in *ADbasic* im allgemeinen die Datentypen **Long** und **Float** (siehe auch Kapitel 4.2.3 „Datentypen“) gemeinsam verwenden, ohne auf passende Datentypen zu achten. Die Daten vom Typ **Long** werden, falls erforderlich, automatisch in den Typ **Float** konvertiert.



Verwechseln Sie die Typkonvertierung nicht mit den Befehlen **Cast\_FloatToLong** oder **Cast\_LongToFloat**, die eine ganz andere Funktion haben (siehe Seite 153).

Beachten Sie bitte folgende Besonderheiten:

- Abschneiden von Nachkommastellen



Wenn ein Fließkomma-Wert einer ganzzahligen Variablen zugewiesen wird, dann werden die Nachkommastellen abgeschnitten und gehen verloren.

- Konvertierung *aller* ganzzahligen Werte

Wenn in einem Berechnungsausdruck ein Fließkomma-Wert enthalten ist, werden *vor* der Auswertung des Ausdrucks *alle* ganzzahligen Werte des Ausdrucks automatisch konvertiert. Dies gilt auch dann, wenn ein ganzzahliger Berechnungsausdruck

- einer Fließkomma-Variablen zugewiesen wird oder
- als Argument eines *ADbasic*-Befehls dient, der einen Fließkomma-Wert erwartet.

### Beispiel

```
Par_1 = 2 / 4 * 3      'Ergebnis: Par_1=0, weil
                       '2/4 = 0
```



Nachkommastellen werden bei rein ganzzahligen Berechnungen immer abgeschnitten und gehen verloren.



aber:

```
FPar_1 = 2 / 4 * 3      'Ergebnis: FPar_1=1,5
Par_1 = 2 / 4.0 * 3      'Ergebnis: Par_1=1
                       '(abgeschnitten!)
```

Hier erzwingen die Fließkomma-Variable **FPar\_1** und der Fließkomma-Wert **4.0** die Konvertierung aller ganzzahligen Werte.

- Automatische Konvertierung verhindern

Auch das Setzen von Klammern verhindert nicht die automatische Konvertierung in **Float**. Sollen Berechnungen unbedingt in **Long** durchgeführt werden, so müssen Sie hierfür eine eigene Zeile programmieren.



### Beispiel

```
Par_1 = 2
Par_2 = 5
Rem hier wird konvertiert:
FPar_3 = (Par_2 / Par_1) + 0.2 'FPar_3 = 2.7
Rem hier dagegen nicht:
Par_9 = Par_2 / Par_1 'Par_9 = 2
                       '(abgeschnitten)
FPar_4 = Par_9 + 0.2 'Ergebnis: FPar_1 = 2.2
```



- Konvertierung von Argumenten

Folgende Ausdrücke werden immer getrennt ausgewertet (und ggf. konvertiert):

- Jeder Parameter eines Befehls.  
Zusätzlich kann durch den Datentyp des Parameters ein Abschneiden von Nachkommastellen auftreten (Parameter-Datentyp siehe jeweilige Befehlsbeschreibung).
- Jedes an Funktionen und Unterprogramme übergebene Argument.
- Jede einzelne Bedingung (Boolescher Ausdruck) bei **If...Then** oder **Do...Until**, die mit **And** und **Or** verknüpft werden kann.

### Beispiel

```
Par_1 = 2 : FPar_2 = 5.5
```



```

Rem Beide Bedingungen sind erfüllt, Par_1 wird nicht in
Rem Float gewandelt: Par_3 = 1
If ((Par_1 / 4 * 3 = 0) And (FPar_2 * 1.1 > 5.5)) Then
    Par_3 = 1
EndIf

Rem Die Bedingung mit Float beeinflusst die
Rem Long-Berechnung nicht: Par_3 = 0
If (FPar_2 * 1.1 > 5.5) Then Par_3 = Par_1 / 4 * 3

```

## 4.5 Bedingungen, Schleifen und Module

Wenn Sie Programme schreiben, strukturieren Sie diese in *ADbasic* 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 187: `Function ... EndFunction`; Seite 268: `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“ *ADbasic*-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 *ADbasic* 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 76).



### 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. 14), 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 203: `Lib_Function ... Lib_EndFunction`; Seite 208: `Lib_Sub ... Lib_EndSub`).

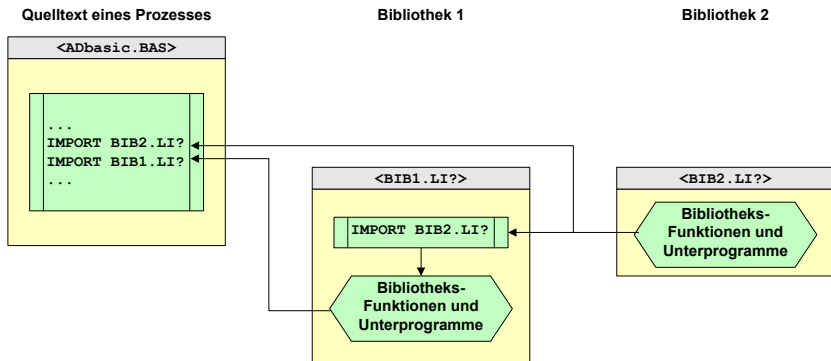


Abb. 14 – Verschachteltes Einbinden von Bibliotheken





## 5 Prozesse optimieren


Ihr *ADwin*-System ist dafür ausgelegt, Regel-, Steuer- und Messaufgabe schnell und präzise auszuführen. Je nach Anforderung kann es erforderlich werden, dass Sie Ihr *ADbasic*-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. Wir verweisen an dieser Stelle auch auf das „*ADbasic* Tutorial“, in dem Sie weitere Beispiele für die Optimierung von Prozessen finden.

### 5.1 Bearbeitungszeit messen

Als Grundlage für eine Optimierung ist es wichtig, die Bearbeitungszeit eines Prozesszyklus oder von Programmabschnitten zu messen. Sie verwenden hierzu die internen Zähler Ihres *ADwin*-Systems.

Der Prozessor des *ADwin*-Systems verfügt über zwei interne Zähler, jeweils einen für hochpriorie und für niederpriorie Prozesse, die in unterschiedlichen Zeittakten hochgezählt werden (siehe Abb. 16 auf Seite 122). Mit dem Befehl **Read\_Timer()** können Sie den aktuellen Zählerstand feststellen; es wird automatisch der Zähler ausgelesen, der der Priorität des laufenden Prozesses entspricht.

Nach dem Einschalten der Spannungsversorgung werden beide 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**

Rem **WICHTIG**: Keinesfalls Float-Variablen verwenden!

```
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 bei einem hochprioren Prozess mit dem Prozessor T10  $37 \times 25\text{ns} = 925\text{ns}$  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 Ihres ADwin-Systems werden über dessen Steuer- und Datenregister kontrolliert. Diese Funktionen können Sie sehr schnell ausführen lassen, wenn Sie mit den Befehlen **Peek** und **Poke** *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

```
FPar_1 = Sqrt(Par_2)      'mit Par_2=17  
FPar_1 = Sqrt(17)
```



Für die erste Berechnung muss zur Laufzeit der Wert der Variablen **Par\_2** ermittelt, die Wurzel berechnet und **FPar\_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*-System zwei ADC befinden, die gleichzeitig zwei verschiedene Kanäle konvertieren können. Dies geschieht im folgenden Beispiel:

#### Beispiel

```
Rem Beispiel für Gold  
Rem Multiplexer der ADC auf Kanäle 1 und 2 setzen  
Set_Mux(000000b)  
...                               'Einschwingzeit abwarten  
Start_Conv(11b)                  'Wandlung an beiden ADC  
                                'starten  
Wait_EOC(11b)                    'Wandlungsende abwarten  
Par_1 = ReadADC(1)               'Auslesen von ADC1  
Par_2 = ReadADC(2)               'Auslesen von ADC2
```



Auf dem *ADwin-light-16*-System befindet sich nur ein ADC.



### 5.2.4 Wartezeit genau einstellen

Mit einer Wartezeit kann man leicht einen genauen Zeitabstand zwischen 2 Befehlen einstellen, z. B. um zwischen **Set\_Mux** und **Start\_Conv** die Einschwingzeit des Multiplexers zu überbrücken.

Der Befehl zum Einstellen der Wartezeit ist abhängig vom Prozessortyp:

- Prozessoren T9 und T10:

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

Das Überbrücken einer Multiplexer-Einschwingzeit von 14µs auf einem Pro I-Modul könnte folgendermaßen aussehen:

```
Set_Mux(2,00000b)      'Mux auf Kanal 1 setzen
Rem Hier z.B. wird eine Berechnung eingeschoben, die
Rem 8µs dauert, um die freie Prozessorzeit zu nutzen.
Sleep(60)               'Restliche Zeit (6µs) bis
                        '14µs warten
Start_Conv(2)           'Wandlung starten
```

- Prozessor T11:

Für die Wartezeit gibt es 3 mögliche Befehle:

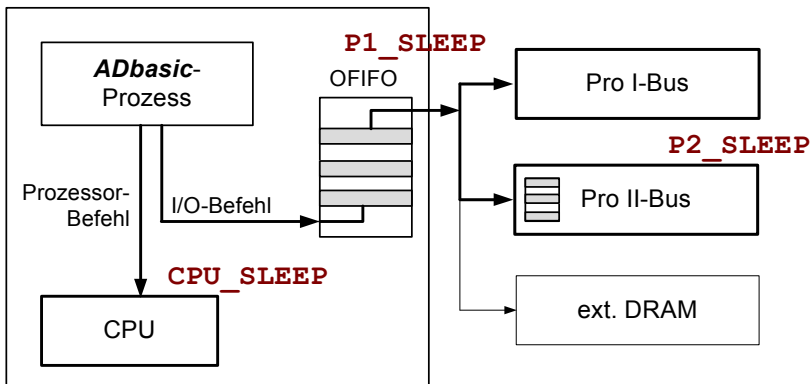
- **P1\_Sleep** lässt den Pro I-Bus warten, aber auch Pro II-Bus und ext. DRAM.
- **P2\_Sleep** lässt den Pro II-Bus warten.
- **CPU\_Sleep** lässt den Prozessor warten (entspricht **Sleep**).

Wenn die Wartezeit einen Zeitabstand zwischen Ein- / Ausgabebefehlen für Pro I-Module überbrücken soll, ist **P1\_Sleep** der richtige Befehl; für Pro II-Module ist es **P2\_Sleep**. Der Befehl **CPU\_Sleep** kann nur in wenigen Fällen sinnvoll eingesetzt werden.

Das Überbrücken einer Multiplexer-Einschwingzeit von 14µs auf einem Pro I-Modul könnte folgendermaßen aussehen:

```
Set_Mux(2,00000b)    'Mux auf Kanal 1 setzen
P1_Sleep(1400)        '14µs auf dem Pro I-Bus
                        'warten.
                        'Beachten Sie die
                        'Zeiteinheit.
Start_Conv(2)         'Wandlung starten
Rem Die Berechnung folgt erst jetzt und wird beim T11
Rem automatisch parallel zu den I/O-Befehlen
    durchgeführt
Rem Achtung: Verwenden Sie für die Berechnung möglichst
    nur
Rem Variablen aus dem internen Speicher. Anderenfalls
    ist es
Rem möglich, dass die Berechnung doch erst ausgeführt
    wird,
Rem wenn die I/O-Befehle abgearbeitet sind.
```

## Prozessor T11



Warum gibt es mehrere Befehle für die Wartezeit? Der Prozessor T11 bearbeitet Prozessor-Befehle und I/O-Befehle<sup>2</sup> quasi-parallel (siehe Skizze oben). Das ist besonders schnell, führt aber auch zu paralleler,

- 
2. I/O-Befehle sind Befehle, die über den Zwischenspeicher OFIFO auf externe Geräte zugreifen. Externe Geräte sind (für die CPU) die Module am Pro I- oder Pro II-Bus und der externe Speicher.

also getrennter Zeitsteuerung und damit zu den 3 Befehlen für die Wartezeit.

Die quasi-parallele Bearbeitung wird durch den 5-stufigen Zwischenspeicher `OFIFO` möglich: Das Betriebssystem legt die I/O-Befehle in `OFIFO` ab (falls dort noch Platz ist) und kann sofort den nächsten Befehl verarbeiten. Im obigen Beispiel werden auf diese Weise die Befehle `Set_Mux`, `P1_Sleep` und `Start_Conv` an `OFIFO` übergeben; die nachfolgende Berechnung kann bereits in der CPU ablaufen, während z. B. die Wartezeit auf dem Pro I-Bus<sup>3</sup> noch aktiv ist.



Beachten Sie bitte: Eine Berechnung, die quasi-parallel in der CPU ablaufen soll, darf nur Variablen aus dem internen Speicher verwenden. Ein Zugriff auf das externe DRAM, den üblichen Speicherort für Felder, ist für das Betriebssystem ein I/O-Befehl und durchläuft daher den Zwischenspeicher `OFIFO`.

### 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_Mux` 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.

---

3. Genauer gesagt lässt der Befehl `P1_SLEEP` nicht den Pro I-Bus warten, sondern den Zwischenspeicher `OFIFO`.

### Beispiel



Rem Beispiel für Gold Rev. B (für T11 nicht geeignet)

#### 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 mit dem Prozessor T11

Dieser Abschnitt beschreibt, wie Sie die besonderen Eigenschaften des Prozessors T11 nutzen, um einen Prozess schneller ablaufen zu lassen, insbesondere durch optimierten Speicherzugriff.

Falls Sie dennoch an die Leistungsgrenzen des Prozessors T11 stoßen, sind weitere Optimierungen möglich, allerdings nur im Zusammenhang mit Ihrer

Anwendung. Bitte wenden Sie sich dazu an unseren Support (siehe Innenseite des Handbuchs).

### Internen Speicher verwenden

Verwenden Sie für zeitkritische Vorgänge möglichst nur Variablen und Felder im internen Speicher (DM oder EM). Während normal deklarierte Variablen automatisch im internen Speicher abgelegt werden, müssen Felder (ob lokal oder global) dafür wie folgt deklariert werden:

```
Dim DataLocal[100] As Long At DM_Local  
Dim Data_5[2000] As Float At DM_Local
```

Im Vergleich zum internen Speicher ist ein Zugriff auf den externen Speicher beim Prozessor T11 aus 2 Gründen wesentlich langsamer. Zum einen wird der Zugriff als I/O-Befehl über den Zwischenspeicher `OFIFO` übertragen (siehe Seite 108), wobei Verzögerungen auftreten können. Zum anderen ist die Verwaltung des externen Speichers langsamer als die Verwaltung des internen Speichers.

### Zugriff auf externen Speicher

Verwenden Sie beim Datenzugriff auf den externen Speicher – soweit im Programm machbar – möglichst immer Datenblöcke, greifen Sie also nicht einzeln auf Werte zu. Bei blockweiser Datenübertragung kann der Prozessor eine beschleunigte Zugriffsart einsetzen, so dass er z. B. einen Block von 20 Werten schneller übertragen kann als 3 einzelne Werte.

Die Blockdatenübertragung ist beispielsweise sinnvoll einsetzbar, wenn viele Messwerte in kurzer Zeit eingelesen werden: Zunächst wird ein Datenpaket im schnellen, internen Speicher abgelegt. Sobald der Messvorgang eine nicht zeitkritische Phase erreicht, werden die Daten mit dem Befehl `MemCpy` in den externen Speicher übertragen und der interne Speicher steht wieder bereit für das nächste Datenpaket.

## 5.3 Debugging und Analyse

Die Bedienoberfläche beinhaltet mit Debug- und Timing-Modus praktische Hilfsmittel zur Fehlersuche und Programmanalyse. Alle Modi werden über das Menü „Debug“ aktiviert (siehe Kapitel 3.7.6 auf Seite 53) und entfalten ihre Hilfstätigkeit für solche Prozesse, die bei aktiviertem Modus kompiliert werden.



Beachten Sie: Das Aktivieren der Hilfs-Modi erzeugt zusätzlichen Programm-Code. Dadurch verlängert sich die Ausführungszeit des Programms und der



Speicherbedarf wird erhöht – zum Teil beträchtlich. Nutzen Sie diese Hilfsmittel daher nur zum Entwickeln und Testen von Programmen.

### 5.3.1 Laufzeitfehler erkennen (Debug-Modus)

Der Debug-Modus ist ein Hilfsmittel zum Aufspüren von folgenden Laufzeitfehlern in *ADbasic*-Programmen:

- Division durch Null
- Wurzel aus einer negativen Zahl
- Zugriff auf zu große / zu kleine Elementnummern eines Felds

Ohne Debug-Modus werden diese Laufzeitfehler ignoriert, d.h. das Ergebnis der entsprechenden Zeile ist undefiniert, wird aber dennoch für den weiteren Programmablauf verwendet. Dies kann, je nach Programm, unerwünschte Folgen haben, im schlimmsten Fall bis zum Absturz des *ADwin*-Systems.

Sie aktivieren die Option „Debug mode“ im Menü „Debug“; kompilieren Sie anschließend den zu prüfenden Quelltext. Sobald ein Laufzeitfehler auftritt, wird er automatisch im Fenster „Debug Errors“ angezeigt. Außerdem wird der Fehler so korrigiert, dass ein stabiler Betriebszustand erhalten bleibt.

Gefundene Fehler sollten in jedem Fall bereinigt werden; auch die automatische Fehlerkorrektur des Debug-Modus ist nur ein Hilfsmittel für die Fehlersuche, nicht für den Dauerbetrieb.



Die Einzelheiten zum Aktivieren und zur Anzeige der Laufzeitfehler sind im Abschnitt „Option Debug mode“ auf Seite 53 beschrieben.

### 5.3.2 Zeitverhalten prüfen (Timing-Modus)

Das *ADwin*-System ist so konzipiert, dass beim Auftreten eines Event-Signals für einen hochprioren Prozess (extern oder vom internen Zähler) sofort der entsprechende Prozesszyklus gestartet wird. Prozesse mit solch „gutem“ Zeitverhalten sind deterministisch und erledigen ihre Aufgaben genau zur vorbestimmten Zeit.

Das Prüfen des Zeitverhaltens von Prozessen erfordert einen gewissen Aufwand, zumal anschließend noch die Änderungen anstehen, um ein gutes Zeitverhalten zu erreichen. Dieser Aufwand lohnt sich dann, wenn höhere Frequenzen oder zusätzliche Aufgaben erforderlich sind, die Prozessor-Auslastung aber an die Grenzen stößt. Ein anderes Beispiel wäre, dass Prozesszyklen nicht genau genug zum vorbestimmten Zeitpunkt starten, die Messung dies aber erfordert.

Im Timing-Modus werden Informationen erzeugt, anhand derer Sie ausgewählte, hochpriorie Prozesse auf „gutes“ Zeitverhalten prüfen können. Für diese Prozesse werden bei jedem Prozesszyklus 7 Kennwerte berechnet, die im Fenster „Timing Analyzer“ angezeigt werden können (siehe Abb. 10).

Ein gutes Zeitverhalten von Prozessen zeichnet sich dadurch aus, dass folgende Situationen *nicht* (oder nur selten) auftreten:

1. Ein Event-Signal startet einen Prozesszyklus nicht sofort, sondern erst um eine gewisse (nicht genau definierte) Zeit später.
2. Ein Event-Signal startet überhaupt keinen Prozesszyklus, sondern geht „verloren“. Auch mehrere verlorene Event-Signale sind möglich.

Im ersteren Fall versucht das Betriebssystem, die Verzögerung wieder auszugleichen, indem es vorhandene Auslastungslücken des Prozessors nutzt, bis wieder alle Prozesszyklen zum vorbestimmten Zeitpunkt starten. Im letzteren Fall kann das Betriebssystem keinen Ausgleich schaffen: Es gehen tatsächlich Event-Signale, und damit auch Prozesszyklen verloren (siehe auch Kapitel 6.2.5 „Verschiedene Betriebszustände im Betriebssystem“, Seite 126).

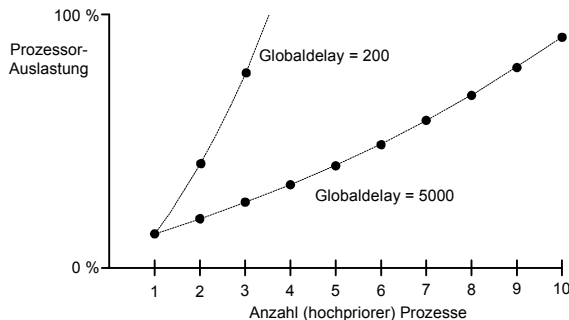
Sie erreichen ein insgesamt optimales Zeitverhalten in 2 Schritten:

1. Anzahl und Priorität von Prozessen prüfen
2. Optimales Zeitverhalten eines Prozesses anstreben, insbesondere bei einem hochpriorien Prozess.

### **Anzahl und Priorität von Prozessen prüfen**

In einem hochpriorien Prozess sollten nur zeitkritische Aufgaben bearbeitet werden, alle anderen Aufgaben dagegen in einem oder mehreren niederpriorien Prozessen (oder sogar nachträglich auf dem PC).

Verwenden Sie möglichst wenige, am besten nur einen einzigen hochpriorien Prozess. Es ist oft möglich, mehrere Prozesse zu einem einzigen Prozess zusammenzufassen; bei gleichem Processdelay sollte man das auf jeden Fall tun. Der Aufwand – insbesondere bei kürzerem Processdelay der Prozesse – ist lohnend, weil dann bei insgesamt gleicher Aufgabenstellung eine wesentlich geringere Prozessorauslastung erreichbar ist. Die unten stehende Grafik zeigt dies qualitativ.



Bei mehreren hochpriorigen, zeitgesteuerten Prozessen ist es zudem nicht vermeidbar, dass immer wieder einzelne Prozesszyklen verzögert starten (außer die Processdelays sind ganzzahlige Vielfache voneinander).

### Optimales Zeitverhalten eines Prozesses

Das optimale Zeitverhalten hat ein hochprioriger Prozess, wenn er folgende Merkmale erfüllt:

- Alle Prozesszyklen des Prozesses haben etwa die gleiche Bearbeitungsdauer
- Die Bearbeitungsdauer für einen Prozesszyklus ist möglichst klein.
- Das Processdelay des Prozesses ist größer als die längste Bearbeitungsdauer aller Prozesszyklen.

Dabei muss die insgesamte Prozessor-Auslastung durch hochpriorige Prozesse den niedripriorigen und dem Kommunikationsprozess genügend Prozessor-Zeit für ihre Arbeit lassen.

Um Informationen über das Zeitverhalten von interessanten Prozessen zu bekommen, gehen Sie vor wie folgt:

1. Aktivieren Sie die Timing-Option mit `Debug ▶ Timing Analyzer`.
2. Kompilieren (und starten) Sie Ihren *ADbasic*-Quelltext.

Zu jedem Quelltext, den Sie bei aktiver Timing-Option kompilieren, werden automatisch Informationen über das Zeitverhalten erzeugt. Betrachten Sie wegen der zusätzlichen Berechnungszeiten nur mög-

lichst wenige Prozesse auf einmal, um das Zeitverhalten insgesamt nicht zu stark zu verändern (s.u.).

3. Deaktivieren Sie die Option `Debug ▶ Timing Analyzer` wieder, damit weitere Prozesse nicht unnötig Timing-Informationen erzeugen.

4. Zeigen Sie das Fenster „Timing Analyzer“ im Info-Bereich an.



Beachten Sie bitte, dass das Zeitverhalten auf dem ADwin-System sehr von der Anzahl und auch von der Art der Prozesse abhängt, dass also auch entsprechend unterschiedliche Kennwerte entstehen. Dies liegt unter anderem an der Verwaltung der Prozesse durch das Betriebssystem (siehe Kapitel 6.2.5 „Verschiedene Betriebszustände im Betriebssystem“).



Die Berechnung der Informationen erfolgt zur Laufzeit und benötigt etwa 60 Taktzyklen (T9, T10 und T11) pro Prozesszyklus und Prozess zusätzlich. Die Kennwerte im Fenster werden laufend aktualisiert und beziehen sich auf den Zeitraum seit dem jeweils letzten Start der Prozesse.

Eine Erklärung der Werte finden Sie bei der Beschreibung zum Fenster „Timing Analyzer“, Seite 67.

Die (geringfügige) Änderung des Zeitverhaltens durch den Timing-Modus ist leider nicht vermeidbar und entsteht auch, wenn die Kennwerte nicht angezeigt werden. Dies kann in Grenzfällen zu zusätzlichen Startverzögerungen (latency) führen, die dann auch in den entsprechenden Kennwerten wiedergegeben werden; bei kurzen Prozessen mit kleinem Processdelay kann manchmal auch eine Prozessor-Auslastung über 100% erreicht werden, d.h. die Kommunikation zum PC reißt ab.

Beachten Sie auch, dass das Kompilieren vieler hochpriorer Prozesse mit der Timing-Option einen niederprioreren Prozess erheblich verlangsamen kann.

## 6 Prozesse im Betriebssystem

Das ADwin-System stellt alle Möglichkeiten zur Verfügung, um komplexe Anlagen zu regeln, zu steuern und Messungen durchzuführen. Mit *ADbasic* programmieren Sie Prozesse, die diese Möglichkeiten nutzen: Sie legen darin fest, wie und wann Ihr System analoge und digitale Daten verarbeitet und nach außen gibt.

Nach dem Start des Prozesses wird das Programm<sup>4</sup> im ADwin-System (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.

Nur einer der 10 möglichen Prozesse kann von einem externen Event gesteuert werden, alle anderen Prozesse müssen zeitgesteuert ablaufen.

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

- Die Initialisierung in den Abschnitten **LowInit:** und/oder **Init:**.
- Die eigentliche Funktion des Prozesszyklus im zentralen Abschnitt **Event:**.
- Die Schlussbearbeitung im Abschnitt **Finish:**.

Vom PC aus können Sie die Prozesse eines Systems steuern, d.h. Sie können die Prozesse starten, stoppen oder deren Processdelay ändern. Dies können Sie sowohl aus *ADbasic* als auch aus Entwicklungsumgebungen wie C++ oder Visual Basic tun. Mit der Bootloader-Option können Prozesse außerdem beim Starten der ADwin-Hardware automatisch gestartet werden. Näheres zum Programmieren des Bootloaders siehe Handbuch „ADwin-Bootloader“.

---

4. genauer: der Programmabschnitt **Event:**.

## 6.1 Prozessverwaltung

### 6.1.1 Prozessarten

Auf einem *ADwin*-System können mehrere Prozesse gleichzeitig ablaufen. Das Betriebssystem sorgt dafür, dass die Prozesszyklen nach bestimmten Regeln aufgerufen und vom Prozessor abgearbeitet werden, ohne sich gegenseitig zu blockieren (siehe auch Kapitel 6.2.5).

Wenn wir im folgenden von einem „Prozess“ sprechen, ist damit einer der von Ihnen programmierten Prozesse 1...10 gemeint.

Sie können jedem Prozess eine bestimmte Priorität zuweisen, und dadurch das zeitliche Zusammenspiel der Prozess-Abarbeitung bestimmen. Es gibt:

- Prozesse mit der Priorität „Hoch“
- Prozesse mit der Priorität „Niedrig“

Niederpriore Prozesse werden weiter in die Stufen -10 (niedrig) bis +10 (hoch) unterteilt.

Weisen Sie einem Prozess seine Priorität über das Menü „Options \ Process Options“ zu.

Prozess	Funktion	Priorität
1...10	Benutzerdefinierte Prozesse mit frei definierbarer Funktion und Priorität	Niedrig Stufe <i>n</i> / Hoch
11, 12	Vordefinierte Ein-Ausgabe-Prozesse	Hoch
15	Prozess zur Steuerung der Blink-LED bei <i>ADwin-Pro</i> - und <i>ADwin-Gold</i> -Systemen	Niedrig, Stufe 1
Kommunikation	Kommunikation zwischen <i>ADwin</i> -System und PC: Befehls- und Datenübertragung	Mittel

Abb. 15 – Übersicht aller Prozesse

Die Standardprozesse 11 und 12 werden nur in Verbindung mit den Treibern für die Bedienoberflächen Labview und Testpoint benötigt. Für diesen Fall können Sie diese Prozesse beim Booten mit dem Betriebssystem in das *ADwin*-System übertragen, entweder aus der Bedienoberfläche (Näheres finden Sie in der Treiber-Dokumentation) oder aus *ADbasic*. In *ADbasic* stellen Sie hierzu im Menü „Options / Compiler“ die Option „Load Standard processes“ auf „Yes“.

Für alle anderen Anwendungen können Sie die Übertragung der Standardprozesse beim Booten unterbinden (Einstellung „NO“).

Der Kommunikationsprozess (siehe Seite 120) ist Teil des Betriebssystems. Er empfängt Kommandos des PC und tauscht Daten zwischen dem ADwin-System und dem PC aus, allerdings nur auf Anforderung des PC.

Wenn Sie mehr als einen Prozess mit derselben Prozess-Nummer auf das System übertragen, wird nur der zuletzt übertragene ausgeführt, weil der vorher übertragene Prozess überschrieben wird.



### 6.1.2 Prozesse mit der Priorität „Hoch“

Prozesse mit der Priorität „Hoch“ werden vom Betriebssystem bevorzugt behandelt:

- Vom Aufruf des Prozesszyklus durch einen Event bis zu dessen Bearbeitungsbeginn vergehen maximal 300ns.
- Ein hochpriorer Prozesszyklus ist nicht unterbrechbar und wird immer vollständig abgearbeitet. Während dieser Zeit werden alle Prozesszyklen mit geringerer Priorität unterbrochen.

Weder ein anderer hochpriorer Prozesszyklus noch ein Stopp-Befehl kann einen laufenden, hochprioren Prozesszyklus unterbrechen. In beiden Fällen muss auf das Ende der Bearbeitung gewartet werden.

Bei zeitgesteuerten hochprioren Prozessen kann die Zykluszeit (Processdelay) je nach Prozessor in Schritten zu 25ns oder 3,3ns eingestellt werden (siehe Abb. 16 auf Seite 122).

Lassen Sie einen zeitkritischen Messprozess mit hoher Priorität laufen und alle anderen mit niedriger Priorität, damit der Prozessor die zeitkritischen Prozesszyklen ungestört bearbeiten kann.



Die Abschnitte **LowInit:** und **Finish:** eines Prozesses werden – falls vorhanden – immer mit niedriger Priorität und Prioritätsstufe 1 ausgeführt, auch wenn Sie dem Prozess die Priorität „Hoch“ gegeben haben.



### 6.1.3 Prozesse mit der Priorität „Niedrig“

Prozesszyklen mit der Priorität „Niedrig“ werden sofort unterbrochen, wenn ein Prozesszyklus mit höherer Priorität aufgerufen wird und zwar so lange, wie dieser bearbeitet wird.

Niederpriorie Prozesse werden in die Prioritätsstufen -10 (niedrig) bis +10 (hoch) unterteilt. Prozesszyklen mit niedrigerer Stufe können jederzeit von solchen mit höherer Stufe unterbrochen werden. Der Prozessor T11 beachtet

die Prioritätsstufen bei der Prozessverwaltung sehr strikt (siehe Kapitel 6.2.3 auf Seite 124).

Niederpriore Prozesszyklen mit gleicher Prioritätsstufe nehmen an einem Zeitscheibenverfahren teil. Dabei teilt das Betriebssystem den Prozesszyklen die Rechenzeit abwechselnd und in gleichen Zeitscheiben zu. Eine Zeitscheibe hat im Mittel eine Dauer von 2ms (Prozessor T9) oder 1ms (Prozessoren T10, T11).

Niederpriore Prozesse müssen immer zeitgesteuert sein. Die Zykluszeit (Processdelay) kann in diskreten Schritten eingestellt werden; Schrittweite siehe Abb. 16 auf Seite 122.

Prozesse mit niedriger Priorität beeinflussen also das Zeitverhalten eines hochprioren Prozesses prinzipiell nicht, wohl aber umgekehrt.

#### 6.1.4 Kommunikationsprozess

Der Kommunikationsprozess hat eine Prioritätsstufe zwischen den Prioritäten „Hoch“ und „Niedrig“. Er kann daher niederpriore Prozesszyklen jederzeit unterbrechen und wird selbst von hochprioren Prozesszyklen unterbrochen.

Wenn der PC mit dem ADwin-System kommuniziert, muss der Kommunikationsprozess spätestens nach 250ms antworten (time out), sonst kann die Kommunikation zwischen PC und ADwin-System abreißen. Sie erhalten dann die Meldung „ADwin-System meldet sich nicht“ und müssen das System durch Booten initialisieren. Dieses „time out“ ist unabhängig von der Schnittstelle USB oder Ethernet.

Der Grund für ein Abreißen der Kommunikation ist, dass dem Kommunikationsprozess nicht genügend Rechenleistung zur Verfügung steht. Dies kann verursacht werden durch

- ein zu kleines Processdelay der hochprioren Prozesse oder
- eine zu lange Bearbeitungszeit eines hochprioren Prozesszyklus.

Weiteres zum Thema Kommunikation finden Sie in Kapitel 6.3.2 auf Seite 128.

#### 6.1.5 Speicherfragmentierung

Das Betriebssystem des ADwin-Systems sorgt dafür, dass Prozesse, Felder und Variablen an der passenden Stelle im Speicher abgelegt werden und korrekt aufgerufen werden. Daher ist die Speicherverwaltung in aller Regel für den Benutzer problemlos und bedarf keiner weiteren Erklärung.

Unter bestimmten Bedingungen können Sie jedoch die Fehlermeldung „Not enough memory or memory access error. Please reboot the ADwin



system.<sup>5</sup> erhalten. Der Grund hierfür ist oft eine Speicherfragmentierung, die entsteht, wenn Prozesse oder Datenfelder mehrmals und mit wachsender Größe in den Speicher geladen werden; ein typischer Vorgang z.B. für das Entwickeln neuer Prozesse.

Eine einfache Lösung ist, das ADwin-System zu booten und die Daten neu zu laden.

Bei einer Speicherfragmentierung liegt freier Speicher zerstückelt zwischen belegten Speicherblöcken. Wenn nun ein neuer Datenblock wie ein Prozess oder ein Feld geladen wird – und zwar nur als unzerteilte Einheit – kann es vorkommen, dass der Datenblock in keines der freien Speicherfragmente passt. Sie erhalten die obige Fehlermeldung und müssen den Speicher neu organisieren, um genügend große freie Speicherbereiche zu erhalten.

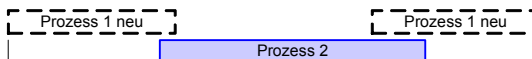
Booten und Neuladen ist hier geeignet, weil die Datenblöcke ohne Lücke hintereinander abgelegt werden und der freie Speicher damit ein zusammenhängender Bereich ist.

Beispiel: Zwei (recht große) Prozesse sind bereits im Speicher geladen. Prozess 1 soll durch neuen, längeren Code ersetzt werden, aber der neue Datenblock passt weder vor noch nach Prozess 2 in den Speicher und Sie erhalten eine Fehlermeldung. Nach dem Booten und Laden in neuer Reihenfolge kann Prozess 1 nun beliebig oft neu geladen werden, ohne dass der Speicher fragmentiert wird.

Erstes Laden



Prozess 1 kann nicht neu geladen werden



Booten und in anderer Reihenfolge neu laden



Als Alternative können Sie auch den Prozess 2 manuell löschen und beide Prozesse neu laden. Der Vorteil ist, dass damit die Werte der globalen Variablen und Felder erhalten bleiben; bei einem globalen Datenfeld bleiben die Werte nur erhalten, wenn die Größe des Felds unverändert bleibt. Die Schwierigkeit beim manuellen Löschen liegt darin, dass Sie bei vielen Prozessen von Anfang an die Übersicht behalten müssen, welcher Prozess im Speicher an welcher Stelle steht.

---

5. Bei den Prozessoren T9 und T10 erscheint keine Fehlermeldung, sondern das ADwin-System gibt eine Auslastung von 100% an.

Ähnlich wie bei Prozessen kann es auch im Datenspeicher zur Speicherfragmentierung kommen, wenn globale Datenfelder nacheinander mit unterschiedlicher Größe geladen werden, z.B. beim Entwickeln eines Prozesses. Beim Laden des Prozesses werden dann die Speicherbereiche der (jetzt anders dimensionierten) Felder freigegeben und jeweils ein neuer Speicherbereich gesucht, was zur Speicherfragmentierung führt. Booten und Neuladen des Prozesses ist auch hier eine einfache Lösung.

Grundsätzlich können globale Felder in *ADbasic* auch einzeln mit `Clear Data` gelöscht werden (siehe Kapitel 3.7.7 auf Seite 56), um genügend große freie Speicherbereiche zu erhalten. Weil aber bei einer Fragmentierung des Datenspeichers meist nicht bekannt ist, in welcher Reihenfolge die globalen Felder im Datenspeicher liegen, ist das Booten hier vorzuziehen.

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“).

## 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 (in früheren *ADbasic*-Versionen: *Globaldelay*). Sie können das *Processdelay* jedes Prozesses über den Wert der Systemvariablen `Processdelay` festlegen (siehe auch Seite 234).

Die Dauer eines Zähler-Taktzyklus ist abhängig von der Prozesspriorität und dem Prozessortyp:

Prozessor	Priorität	
	Hoch	Niedrig
T9	25ns	100µs
T10	25ns	50µs
T11	3,3ns	3,3ns = 0,003µs

Abb. 16 – Dauer eines Zähler-Taktzyklus (Einheit des *Processdelay*)

Ein *Processdelay* mit dem Wert 1000 beispielsweise bedeutet in einem hoch-prioriten Prozess mit dem Prozessor T9 einen regelmäßigen Aufruf im Zeitab-

stand von  $1000 \times 25\text{ns} = 25000\text{ns} = 25\mu\text{s}$ , bei einem niederprioren Prozess dagegen von  $1000 \times 100\mu\text{s} = 100000\mu\text{s} = 100\text{ms}$ . Dies kann z.B. eingestellt werden mit der Programmzeile:

`Processdelay = 1000`

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).

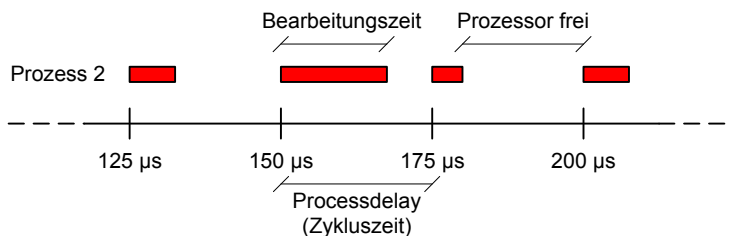


Abb. 17 – Processdelay und Bearbeitungszeit bei hochprioren Prozesszyklen

### 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 Zeitlich exakter Aufruf von Prozesszyklen

Wenn Sie (wie in Abb. 17) genau einen hochprioren Prozess haben, so wird dieser exakt im Zeitraster aufgerufen und abgearbeitet.

Stellen Sie sicher, dass die Bearbeitungszeit eines hochprioren Prozesszyklus seine Zykluszeit (im Beispiel unten:  $25\mu\text{s}$ ) nie überschreitet. Anderenfalls können – weil dieser Prozesszyklus nicht unterbrechbar ist – andere Prozesszyklen nur unvollständig oder gar nicht mehr bearbeitet werden, z.B. der wichtige Kommunikations-Prozess.

Gibt es mehrere hochprioriäre Prozesse, kann der jeweils aktive Prozesszyklus das Zeitraster der übrigen Prozesszyklen beeinflussen. In Abb. 18 kann z. B. der geplante Aufruf im Prozess 1 erst verzögert geschehen, wenn die Bearbeitung des aktiven Prozesses 2 beendet ist.

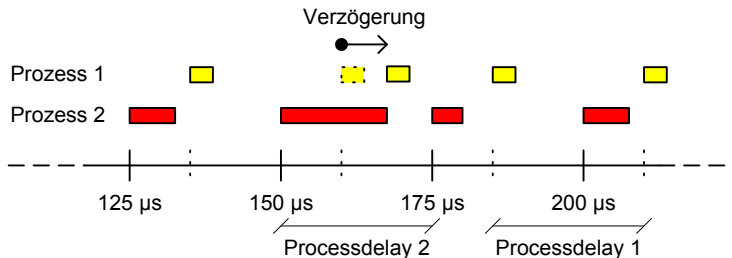


Abb. 18 – Verzögerung eines hochprioriären Prozesszyklus



Halten Sie die Ausführungszeit von hochprioriären Prozesszyklen so gering wie möglich. Lassen Sie zeitintensive Schleifen oder Berechnungen, deren Ergebnis nicht sofort weiter verarbeitet wird, immer in Prozesszyklen mit niedriger Priorität ablaufen.

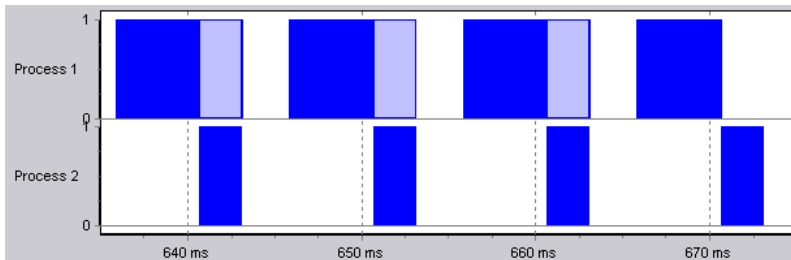
Ein niedriprioriärer Prozess ist abhängig vom Zeitverhalten aller anderen Prozesszyklen mit höherer oder gleicher Priorität. Jede Unterbrechung verringert das Zeitfenster, in dem der niedriprioriäre Prozesszyklus Rechenleistung nutzen kann; im Extremfall wird er überhaupt nicht aufgerufen.

### 6.2.3 Niederprioriäre Prozesse beim T11

Der Prozessor T11 verwaltet niedriprioriäre Prozesse strikt nach deren Prioritätsstufe. Im Vergleich dazu haben Prioritätsstufen bei T9 und T10 nur geringe Bedeutung. Der Vorrang von Kommunikationsprozess und hochprioriären Prozessen vor allen niedriprioriären Prozessen bleibt davon unberührt.

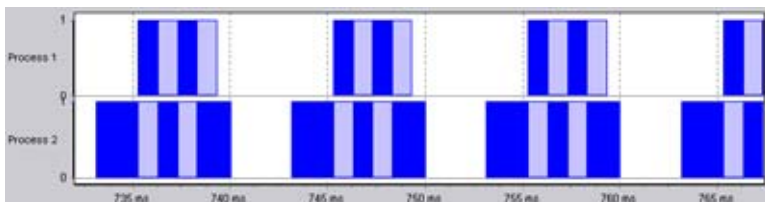
Bei niedriprioriären Prozessen ist zu unterscheiden zwischen:

- Prozesse mit unterschiedlicher Prioritätsstufe: Alle Prozesse mit niedrigerer Prioritätsstufe werden unterbrochen, sobald und solange ein Prozess mit höherer Prioritätsstufe bearbeitet wird.



Hier hat Prozess 2 die höhere Prioritätsstufe und unterbricht deswegen immer wieder Prozess 1.

- Prozesse mit gleicher Prioritätsstufe: Die Prozesse nehmen am Zeitscheibenverfahren teil. Das Betriebssystem teilt den Prozesszyklen die Rechenzeit innerhalb der Prioritätsstufe abwechselnd und in gleichen Zeitscheiben (1 ms) zu.



Im Beispiel ist der Wechsel zwischen den beiden Prozessen gut zu sehen. Es zeigt auch die Regel, dass ein Prozess – hier Prozess 1 – bei jedem Aufruf sofort eine Zeitscheibe Rechenzeit erhält.

In einem seltenen Sonderfall wird das Zeitscheibenverfahren ausgehebelt: Ein Prozess erhält sehr viel Rechenzeit, wenn er sehr häufig aufgerufen wird und sein Prozesszyklus zugleich kürzer als eine Zeitscheibe dauert. Bei jedem Aufruf unterbricht dieser Prozess andere Prozesse mit gleicher Prioritätsstufe und „stiehlt“ ihnen damit die Rechenzeit.

### 6.2.4 Auslastung des ADwin-Systems

Die Auslastung des Prozessors auf dem ADwin-System 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 3.8.5). 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.

Bitte beachten Sie beim Prozessor T11: Trotz einer angezeigten Auslastung unter 90% kann eine Überlastung vorliegen, so dass z. B. manche Prozesszyklen verspätet bearbeitet werden. Die Überlastung liegt dann nicht beim Prozessor vor, sondern auf dem Pro I- oder Pro II-Bus, deren Auslastung nicht angezeigt werden kann.

### 6.2.5 Verschiedene Betriebszustände im Betriebssystem

Das Betriebssystem unterscheidet beim Zeitverhalten für hochpriorer Prozesse zunächst 2 Betriebszustände, je nachdem, ob nur 1 oder ob mehrere zeitgesteuerte (hochpriorer) Prozesse aktiv sind. Ob zusätzlich ein extern gesteuerter Prozess läuft, ist hierbei nicht von Belang. Der extern gesteuerte Prozess wird vom Betriebssystem separat organisiert und kann daher als 3. Betriebszustand verstanden werden.

#### Einzelner zeitgesteuerter Prozess

Das Betriebssystem verwendet bei einem einzelnen, zeitgesteuerten Prozess (u. a.) Hardware-Bausteine, um die Event-Signale des internen Zählers zu verarbeiten. Dadurch kann das Betriebssystem ein eintreffendes Event-Signal dieses Prozesses sehr schnell bearbeiten.

Die Hardware-Bausteine können zwischenspeichern, ob ein Event-Signal eingetroffen ist, allerdings nicht wieviele Event-Signale es waren. Ist ein Event-Signal eingetroffen, aktiviert das Betriebssystem den nächsten Prozesszyklus zum festgelegten Zeitpunkt (siehe Processdelay, Kapitel 6.2.1), wenn nicht gerade ein hochpriorer Prozesszyklus bearbeitet wird. In diesem Fall aktiviert das Betriebssystem den nächsten Prozesszyklus sofort nach dem aktuell bearbeiteten Prozesszyklus.



Wenn mehrere Event-Signale eintreffen, während gerade ein hochpriorer Prozesszyklus bearbeitet wird, werden nicht entsprechend viele Prozesszyklen aufgerufen, sondern nur ein einziger; es gehen also Event-Signale verloren. Achten Sie deshalb streng darauf, dass die Prozesszyklen kürzer sind als die Zykluszeit (Processdelay) des Prozesses.

### Mehrere zeitgesteuerte Prozesse

Bei mehreren zeitgesteuerten Prozessen werden eintreffende Event-Signale vom Betriebssystem selbst verwaltet. Dieser Betriebsmodus ist wegen des erforderlichen Verwaltungsaufwands zwar langsamer, dafür wird aber die Anzahl aller eintreffenden Event-Signale für jeden Prozess zwischengespeichert. Damit ist gewährleistet, dass zu jedem Event-Signal ein Prozesszyklus gestartet wird, wenn auch ggf. später als zum eigentlich festgelegten Zeitpunkt.

Meistens sind die Zeitraster für den Start der Prozesszyklen derart, dass immer wieder Event-Signale während der Bearbeitung eines anderen Prozesszyklus auftreten. Anders gesagt, die Processdelay-Werte sind keine ganzzahligen Vielfachen voneinander. Wir empfehlen daher, mit möglichst wenigen Prozessen auszukommen; meistens ist es möglich (und erzielt außerdem eine geringere Prozessorauslastung), mehrere Prozesse zu einem einzigen Prozess zusammenzufassen.

Beachten Sie immer, dass die Prozessorauslastung stark von der Zahl der laufenden Prozesse abhängt. So benötigt eine Aufgabe, die von 2 (oder sogar mehr) Prozessen ausgeführt wird, in jedem Fall mehr Prozessorzeit als die gleiche Aufgabe mit einem einzelnen Prozess. Dies fällt umso stärker ins Gewicht je kürzer das Processdelay der Prozesse ist (siehe auch Kapitel 5.3.2 auf Seite 113).



Beispiel: Prozess 1 und 2 mit sehr kleinem Processdelay erzeugen als Einzelprozess je 10% Auslastung; beide Prozesse gemeinsam erzeugen 55% Auslastung.

### Extern gesteuerter Prozess

Der Betriebsmodus für den extern gesteuerten Prozess ist unabhängig von den zeitgesteuerten Prozessen immer der gleiche. Der externe Prozess wird vom Betriebssystem verwaltet wie ein einzelner zeitgesteuerter Prozess (s.o.), d.h. eintreffende Event-Signale werden sehr schnell bearbeitet, jedoch können auch Event-Signale verloren gehen.

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 (verlorene Events finden, siehe Seite 67). 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 *ADbasic*-Prozessen über globale Variablen (**Par\_1** ... **Par\_80**, **FPar\_1** ... **FPar\_80**) oder über globale Felder (**Data\_n**) austauschen. Auf diesem Weg ist auch der Datenaustausch mit Programmen im PC möglich.



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“).

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 ADwin-System

Vom PC aus können Sie aus Anwendungen und Entwicklungsumgebungen Prozesse im *ADwin*-System steuern, sowie Daten von dort lesen anfordern oder dorthin senden. Grundsätzlich kann ein *ADwin*-System nie selbstständig mit dem PC kommunizieren, sondern reagiert nur auf eine Anfrage des PC.

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



Jede Kommunikation zum *ADwin*-System läuft unter Windows über die sogenannte „ADwin32.dll“ (dynamic-link library), im System übernimmt diese Aufgabe der Kommunikationsprozess (Seite 120).

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

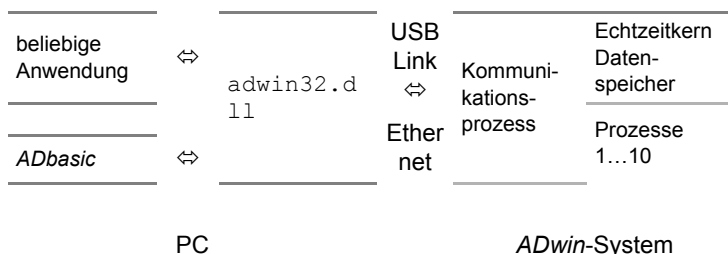


Abb. 19 – Kommunikation zwischen PC und *ADwin*-System

Die „ADwin32.dll“ übernimmt folgende Aufgaben:

- Kommunikation mit dem angesprochenen *ADwin*-System über die verbindende Schnittstelle: USB, 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 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 USB, TCP/IP). Auf diese Informationen greift die „*ADwin32.dll*“ zurück, um mit dem System kommunizieren zu können.

### 6.3.4 Kommunikation mit Entwicklungsumgebungen

Vom PC aus greifen Sie aus einer Bedienoberfläche auf das *ADwin*-System zu. Sie können diese mit einer der gängigen Entwicklungsumgebungen selbst gestalten, wie z. B. ActiveX, Java, Visual Basic, C++, Delphi oder C#.NET, oder eine fertige Bedienoberfläche wie etwa TestPoint, DIAdem oder MATLAB verwenden.

Für jede dieser Möglichkeiten erhalten Sie bei uns eine passende Treiber-Software, mit der Sie auf das *ADwin*-System zugreifen können. Wenn Sie spezielle Wünsche haben, fragen Sie bei uns an. Sie erhalten bei uns auch maßgeschneiderte Messdaten-Auswertungsprogramme.

Unter Windows ermöglicht eine DLL- oder ActiveX-Schnittstelle die Kommunikation mit dem System auch aus mehreren Programmen gleichzeitig (siehe auch „Kommunikation zwischen PC und *ADwin*-System“ auf Seite 128).



Die speziellen Befehle für Ihre Bedienoberfläche werden in den Unterlagen zu der jeweiligen Treiber-Software genau erläutert.

Sie können aus Ihrer Bedienoberfläche:

- kompilierte Programme (Binärdateien) in das *ADwin*-System übertragen. Sie kompilieren ein Programm in *ADbasic* mit **Build ▶ Make Bin File** (siehe Kapitel 3.7.4 auf Seite 42).
- Prozesse im *ADwin*-System starten, steuern und anhalten.
- Daten vom *ADwin*-System anfordern oder dorthin senden.

Obwohl das *ADwin*-System autark arbeitet, können Sie aus der Bedienoberfläche jederzeit auf globale Variablen und Felder zugreifen, ohne zeitkritische Prozesse zu verzögern. Auf dem beschriebenen Weg können alle Prozesse mit dem PC (und auch untereinander) schnell Daten miteinander austauschen.

## 7 Befehlsreferenz

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.

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

In Kapitel 7.3 und Kapitel 7.4 sind *ADbasic*-Befehle für die Anwendung der FFT-Library sowie die Mathematik-Befehle 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** oder **3.14159** sowie Berechnungsausdrücke ohne Variablen.

Konstante Zeichenketten werden in doppelten Hochkommas angegeben wie **"dieser Text"**.

**VAR** Variable oder Feldelement.

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

**FIFO** FIFO-Feld (als FIFO deklariertes **Data\_n**-Feld).

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

**LONG** ganze Zahl

**FLOAT** Fließkomma-Zahl

**STRING** Zeichenkette

**LOGIC** logischer Ausdruck in einer Bedingung

Falls ein Argument nicht den erwarteten Datentyp hat, wird der Datentyp des Arguments gewandelt (siehe „Typkonvertierung“ auf Seite 98).

- 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 Befehle L16, Gold, Pro

Die Befehle in diesem Abschnitt sind gültig für alle Typen von ADwin-Hardware: *ADwin-light-16*, *ADwin-Gold*, *ADwin-Gold II*, *ADwin-Pro*, *ADwin-Pro II*.

## + (Addition)

Der Operator „+“ addiert je zwei Werte (siehe auch „+ (String-Addition)“).

### Syntax

```
val = val_1 + val_2
```

### Parameter

`val_1`                      Summand 1

FLOAT

LONG

`val_2`                      Summand 2

FLOAT

LONG

### Bemerkungen

Beachten Sie, dass die Verknüpfung von verschiedenen Variablentypen mit dem „+“-Operator zu einer Typ-Konvertierung führt. Bei der Wandlung vom Typ `Long` in den Typ `Float` kann es zu Rundungsabweichungen kommen, die das Berechnungsergebnis beeinflussen.

### Siehe auch

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

### Beispiel

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

## + (String-Addition)

Der Operator „+“ verknüpft je zwei Strings zu einem neuen String (siehe auch „+ (Addition)“).

### Syntax

```
Import String.LI*          '*.LI9 für T9, *.LIA für
    T10,
                               '*.LIB für T11

val = val_1 + val_2
```

### Parameter

val_1	Zeichenkette 1
val_2	Zeichenkette 2

STRING

STRING

### Bemerkungen

Wenn Sie Strings „addieren“ und einem weiteren String zuweisen, muss dieser Ziel-String größer oder gleich der Summe aller ASCII-Zeichen der Teil-Strings deklariert sein.

### Siehe auch

"" String, Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, Vall

### Beispiel

```
Import string.li9
```

```
Rem 3 Strings dimensionieren: 10, 5, 4 Zeichen
```

```
Dim res_str[10] As String
```

```
Dim str_1[5] As String
```

```
Dim str_2[4] As String
```

#### Init:

```
str_1 = "ADwin"           '5 Zeichen
```

```
str_2 = "Gold"            '4 Zeichen
```

#### Event:

```
res_str = str_1 + "-" + str_2 'Strings "addieren"
```

```
Par_1 = StrLen(res_str) 'Par_1 = 10 (Anzahl der  
                        'Zeichen)
```

## - (Subtraktion)

Der Operator „-“ subtrahiert je zwei Werte.

### Syntax

```
val = val_1 - val_2
```

### Parameter

`val_1`                      Minuend

FLOAT

LONG

`val_2`                      Subtrahend

FLOAT

LONG

### Bemerkungen

Beachten Sie, dass die Verknüpfung von verschiedenen Variablentypen mit dem „-“-Operator zu einer Typ-Konvertierung führt. Bei der Wandlung vom Typ `Long` in den Typ `Float` kann es zu Rundungsabweichungen kommen, die das Berechnungsergebnis beeinflussen.

Wenn Sie „-“ als Vorzeichen einer Variablen verwenden (unärer Operator), kann es in manchen Fällen zu unerwarteten Ergebnissen kommen, die Sie durch Klammersetzung vermeiden können (siehe auch Kapitel 4.4.1 auf Seite 97).

### 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

FLOAT

LONG

val\_2                      Multiplikator 2

FLOAT

LONG

### Bemerkungen

Beachten Sie, dass die Verknüpfung von verschiedenen Variablentypen mit dem „\*“-Operator zu einer Typ-Konvertierung führt. Bei der Wandlung vom Typ **Long** in den Typ **Float** kann es zu Rundungsabweichungen kommen, die das Berechnungsergebnis beeinflussen.

### 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

FLOAT

LONG

`val_2` Divisor

FLOAT

LONG

### Bemerkungen

Beachten Sie, dass die Verknüpfung von verschiedenen Variablentypen mit dem „/“-Operator zu einer Typ-Konvertierung führt (siehe Kapitel 4.4.2 auf Seite 98). Bei der Wandlung vom Typ `Long` in den Typ `Float` kann es zu Rundungsabweichungen kommen, die das Berechnungsergebnis beeinflussen.

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“ auf Seite 97).

### Siehe auch

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

### 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

Float

Long

`val_2` Exponent

Float

Long

### Bemerkungen

Beachten Sie, dass die Verknüpfung von verschiedenen Variablentypen mit dem Potenz-Operator zu einer Typ-Konvertierung führt. Bei der Wandlung vom Typ `Long` in den Typ `Float` kann es zu Rundungsabweichungen kommen, die das Berechnungsergebnis beeinflussen.

Wenn Basis und Exponent ganzzahlige Variablen (nicht aber Konstanten) sind, wird die Potenzfunktion intern dennoch mit Float-Arithmetik ausgeführt. Bei großen Ergebniswerten führt dies zu den bei Float üblicherweise zu erwartenden Rechenungenauigkeiten bei großen Zahlen.

Beispiel:

```
Par_2 = 31           ' variable
Par_1 = 2^Par_2      ' = 7FFFFFFE2h
```

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“ auf Seite 97). 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:

$y = a + b \cdot x + c \cdot x^2 + d \cdot x^3 + e \cdot x^4$  *'langsame Variante'*

$y = a + x \cdot (b + x \cdot (c + x \cdot (d + x \cdot e)))$  *'schnelle Variante'*

### Siehe auch

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

### Beispiel

**Par\_1** = 9 ^ 4

*'Par\_1 = 6561'*

### #... (Präprozessor-Anweisung)

Ein *ADbasic*-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 197).

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

FLOAT

LONG

STRING

`expr` Berechnungsausdruck

FLOAT

LONG

STRING

### Bemerkungen

Wenn das Datenformat des Berechnungsausdrucks nicht mit dem der Variablen oder des Felds übereinstimmt, wird es in das passende Datenformat gewandelt oder die Zuweisung wird als ungültig zurückgewiesen. Bei der Umwandlung kann es zu Rundungsabweichungen kommen, die das Berechnungsergebnis beeinflussen.

### 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 *ADbasic* kommen diese Operatoren nur in Bedingungen vor.

### Syntax

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

### Parameter

val\_1            Operand

FLOAT

LONG

val\_2            Operand

FLOAT

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
```



## AbsF

**AbsF** liefert den Betrag einer Float-Variablen.

### Syntax

```
ret_val = AbsF(value)
```

### Parameter

value                      Argument

FLOAT
-------

ret\_val                    Betrag des Arguments

FLOAT
-------

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 150ns, beim T10 bis zu 75ns, beim T11 bis zu 17ns.

### Siehe auch

AbsI

### Beispiel

```
Dim val_1, val_2 As Float
```

**Event:**

```
val_1 = -5.3
```

```
val_2 = AbsF(val_1)    'Ergebnis: val_2 = 5.3
```

## 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

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 75ns, beim T10 bis zu 50ns, beim T11 bis zu 17ns.

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

### Siehe auch

AbsF, Mod

### 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 fval\_1 As Float

Dim val4 As Long

fval\_1 = 3.14

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

If ((fval\_1 < 9.1) And (fval\_1 > 3.1)) Then

    val4 = 1

Else

    val4 = 0

EndIf

*'Ergebnis: val4 = 1*

## ArcCos

**ArcCos** liefert den Arcus-Cosinus eines Arguments.

### Syntax

```
ret_val = ArcCos(val)
```

### Parameter

<code>val</code>	Argument (-1 ... +1)	Float
<code>ret_val</code>	Arcus-Cosinus des Arguments im Bogenmaß (0... $\pi$ )	Float

### Bemerkungen

Für `val` < -1 wird der Wert  $\pi$  (3,14159...) zurückgegeben, für `val` > 1 der Wert 0 (Null).

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 2,9 $\mu$ s, beim T10 bis zu 1,45 $\mu$ s, beim T11 bis zu 0,68 $\mu$ s.

### Siehe auch

Sin, Cos, Tan, ArcSin, ArcTan

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 0.5  
val_2 = ArcCos(val_1)  
Rem Ergebnis: val_2 = 1.0472
```

## ArcSin

**ArcSin** liefert den Arcus-Sinus eines Arguments.

### Syntax

```
ret_val = ArcSin(val)
```

### Parameter

<code>val</code>	Argument (-1 ... +1)	<span style="border: 1px solid black; padding: 2px;">FLOAT</span>
<code>ret_val</code>	Arcus-Sinus des Arguments im Bogenmaß ( $-\pi/2 \dots +\pi/2$ )	<span style="border: 1px solid black; padding: 2px;">FLOAT</span>

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 2,8µs, beim T10 bis zu 1,4µs, beim T11 bis zu 0,67µs.

### Siehe auch

Sin, Cos, Tan, ArcCos, ArcTan

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 0.5  
val_2 = ArcSin(val_1)  
Rem Ergebnis: val_2 = 0.5236
```

## ArcTan

**ArcTan** liefert den Arcus-Tangens eines Arguments.

### Syntax

```
ret_val = ArcTan(val)
```

### Parameter

<code>val</code>	Argument (gesamter Wertebereich, siehe „Zahlenwerte eingeben“ auf Seite 82).	<code>Float</code>
<code>ret_val</code>	Arcus-Tangens des Arguments im Bogenmaß ( $-\pi/2 \dots \pi/2$ )	<code>Float</code>

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,8µs, beim T10 bis zu 0,9µs, beim T11 bis zu 0,42µs.

### Siehe auch

Sin, Cos, Tan, ArcSin, ArcCos

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 0.5  
val_2 = ArcTan(val_1)  
Rem Ergebnis: val_2 = 0.4636
```

## Asc

**Asc** ermittelt die zugehörige ASCII-Nummer zu einem einzelnen Zeichen oder zum ersten Zeichen einer Zeichenfolge.

### Syntax

```
ret_val = Asc(string)
```

### Parameter

**string**      Zeichenfolge

STRING
--------

**ret\_val**      ASCII-Nummer (0...255) des (ersten)  
Zeichens

LONG
------

### Bemerkungen

- / -

### Siehe auch

"" String, + (String-Addition), Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, ValI

### Beispiel

```
Dim text[10] As String
```

**Init:**

```
text="Hallo"
```

**Event:**

```
Par_1=Asc(text)
```

```
'Par_1 = 48h = 72
```

```
Par_2=Asc("?")
```

```
'Par_1 = 3Fh = 63
```



## Cast\_FloatToLong

**Cast\_FloatToLong** wandelt den Datentyp eines Arguments von Float (Fließkomma) in Long (ganze Zahl).

### Syntax

```
ret_val = Cast_FloatToLong(var)
```

### Parameter

<code>var</code>	Bitmuster mit Datentyp Float (Fließkomma)	<div>FLOAT</div>
<code>ret_val</code>	Identisches Bitmuster mit Datentyp Long (ganze Zahl)	<div>LONG</div>

### Bemerkungen

Es handelt sich bei dieser Funktion **nicht** um die übliche Typkonvertierung eines Zahlenwerts (siehe Kapitel 4.4.2 „Typkonvertierung“, Seite 98). Für die Zuweisung einer Fließkomma-Zahl an eine ganzzahlige Variable genügt der Operator „=“.

Die Funktion ist im Zusammenhang mit der Umkehrfunktion **CAST\_LONGTOFLOAT** sinnvoll einsetzbar, wenn ein Bitmuster eine Fließkomma-Zahl repräsentiert, jedoch mit dem Datentyp Long vorliegt. Die Wandlung lässt das Bitmuster unverändert und ändert nur den Datentyp, so dass die Zahl wieder korrekt als Fließkomma-Zahl interpretiert wird (siehe auch Kapitel 4.2.3 auf Seite 80).

Ein Anwendungsbeispiel tritt bei der Datenübertragung auf: CAN- oder RSxxx-Busse übertragen nur 8 Bit-Datenpakete mit ganzzahligem Datentyp. Der Datentyp eines 32-Bit Fließkomma-Werts muss deshalb mit **Cast\_FloatToLong** von Float in Long gewandelt und der Wert anschließend in 4 separate 8 Bit-Pakete aufgeteilt werden. Beim Empfänger werden die Datenpakete wieder zusammengesetzt und mit **Cast\_LongToFloat** wieder vom Datentyp Long in Float gewandelt.

### Siehe auch

Cast\_LongToFloat

## Cast\_LongToFloat

**Cast\_LongToFloat** wandelt den Datentyp eines Bitmusters von Long (ganze Zahl) in Float (Fließkomma).

### Syntax

```
ret_val = Cast_LongToFloat(val)
```

### Parameter

<code>val</code>	Bitmuster mit Datentyp Long (ganze Zahl)	LONG
<code>ret_val</code>	Identisches Bitmuster mit Datentyp Float (Fließkomma)	FLOAT

### Bemerkungen

Es handelt sich bei dieser Funktion **nicht** um die übliche Typkonvertierung eines Zahlenwerts (siehe Kapitel 4.4.2 „Typkonvertierung“, Seite 98). Für die Zuweisung einer Fließkomma-Zahl an eine ganzzahlige Variable genügt der Operator „=“.

Die Funktion ist sinnvoll einsetzbar, wenn ein Bitmuster eine Fließkomma-Zahl repräsentiert, jedoch mit dem Datentyp Long vorliegt. Die Wandlung lässt das Bitmuster unverändert und ändert nur den Datentyp, so dass die Zahl wieder korrekt als Fließkomma-Zahl interpretiert wird (siehe auch Kapitel 4.2.3 auf Seite 80).

Ein Anwendungsbeispiel tritt bei der Datenübertragung auf: CAN- oder RSxxx-Busse übertragen nur 8 Bit-Datenpakete mit ganzzahligem Datentyp. Der Datentyp eines 32-Bit Fließkomma-Werts muss deshalb mit **Cast\_FloatToLong** von Float in Long gewandelt und der Wert anschließend in 4 separate 8 Bit-Pakete aufgeteilt werden. Beim Empfänger werden die Datenpakete wieder zusammengesetzt und mit **Cast\_LongToFloat** wieder vom Datentyp Long in Float gewandelt.

### Siehe auch

Cast\_FloatToLong

## Chr

**Chr** weist einer String-Variablen ein Zeichen mit einer bestimmten ASCII-Nummer zu.

### Syntax

```
Import String.LI*      '*.LI9 für T9, *.LIA für
                        T10,
                        '*.LIB für T11

Chr(vascii,dest_text)
```

### Parameter

<code>vascii</code>	ASCII-Nummer (0...255) des zugewiesenen Zeichens	LONG
<code>dest_text</code>	String-Variable, der das Zeichen zugewiesen wird	STRING

### Bemerkungen

Wenn eine String-Variable mehr als ein Zeichen (oder Element) hat, weist **Chr** die ASCII-Nummer nur dem ersten Zeichen zu.

### Siehe auch

"" String, + (String-Addition), Asc, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, Vall

### Beispiel

```
Import String.LI9

Dim text_a[1], text_b[1] As String

Event:
Chr(13, text_a)      'Carriage Return
Chr(10, text_b)      'Line Feed
```

## Cos

**Cos** liefert den Cosinus eines Winkels.

### Syntax

```
ret_val = Cos(angle)
```

### Parameter

**angle** Winkel im Bogenmaß ( $-\pi \dots \pi$ )

FLOAT
-------

**ret\_val** Cosinus des Winkels ( $-1 \dots 1$ )

FLOAT
-------

### Bemerkungen

Wenn Sie für den Winkel Werte außerhalb von  $-\pi \dots \pi$  verwenden, nimmt der Berechnungsfehler mit wachsendem Wert zu.

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,3  $\mu$ s, beim T10 bis zu 0,7  $\mu$ s, beim T11 bis zu 0,31  $\mu$ s.

### Siehe auch

Sin, Tan, ArcCos, ArcSin, ArcTan

### Beispiel

```
Dim val_1, val_2 As Float
```

**Event:**

```
val_1 = -5.3
```

```
val_2 = Cos(val_1) 'Ergebnis: val_2 = 0.55...
```

## CPU\_Sleep

Nur für Prozessor T11: **CPU\_Sleep** lässt den Prozessor für eine bestimmte Zeit warten.

### Syntax

**CPU\_Sleep**(val)

### Parameter

val                      Anzahl ( $9...715827879 \approx 2^{30} / 1,5$ ) der zu wartenden Zeiteinheiten in 10ns. LONG

### Bemerkungen

Hinweis für *ADwin-Pro II*: Alternativ gibt es die Anweisungen **P1\_Sleep** und **P2\_Sleep** (siehe auch Kapitel 5.2.4 „Wartezeit genau einstellen“). Verwenden Sie bei Prozessoren bis T10 die Anweisung **Sleep**.

Die Wartezeit sollte grundsätzlich kleiner sein als die mit **Processdelay** eingestellte Zykluszeit.

Die Anweisung **CPU\_Sleep** kann bei einem hochprioren Prozess nicht unterbrochen werden. Bitte beachten Sie, dass sehr große Werte bei hochprioren Prozessen zu einem Abbruch der Kommunikation führen können.



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. Die Zeitspanne kann hier schwanken, weil sie von mehreren Voraussetzungen abhängt.
- Das Argument ist ein Feld.
- Das Argument ist ein Fließkomma-Wert.

### Siehe auch

IO\_Sleep, NOP, P1\_Sleep, P2\_Sleep, Processdelay, Sleep

**Beispiel****Event:**

*Rem Warten, um eine nachfolgende Messung genau 100  $\mu$ s  
Rem nach dem externen Event-Signal zu starten.*

**CPU\_Sleep**(10000)

...

## Data\_n

Mit `Dim Data_n[...]` `As ...` wird ein globales **DATA**-Feld dimensioniert. Weitere Informationen zur Dimensionierung siehe [Dim](#) auf Seite 164.

### Syntax

```
Dim Data_n[dim1] {[, Data_n[dim2]]} As <ARR_TYPE>
    {At <Mem_Type>}

Dim Data_n[dim1] {[dim2]} As <ARR_TYPE> {At
    <Mem_Type>}
```

### Parameter

<b>Data_n</b>	Name des deklarierten <b>DATA</b> -Felds mit <b>n</b> : 1...200.	
<b>&lt;ARR_TYPE&gt;</b>	Datentyp: <a href="#">Float</a> , <a href="#">Long</a> , <a href="#">String</a> .	
<b>dim1, dim2</b>	Feldgröße: Anzahl ( $\geq 1$ ) der Elemente vom Typ <a href="#">ARR_TYPE</a> im Feld.	<b>CONST</b>
<b>&lt;Mem_Type&gt;</b>	Speicher, in dem die Variablen abgelegt werden: <a href="#">DRAM_Extern</a> : externer Datenspeicher (Default). <a href="#">DM_Local</a> : interner Datenspeicher.	<b>LONG</b>
	nur ab T11 verfügbar: <a href="#">EM_Local</a> : Zusätzlicher interner Programm- oder Datenspeicher.	

### 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 *ADwin*-System.

Ein globales Feld kann auch 2-dimensional deklariert werden. Näheres ist in „2-dimensionale Felder“ auf Seite 89 beschrieben.

### Siehe auch

[Dim](#), [FIFO](#), „Globale Felder (Arrays)“ auf Seite 83, „Variablen und Felder im Datenspeicher“ auf Seite 87

**Beispiel**

*Rem Dimensioniere das globale Feld Data\_15 mit  
Rem 1000 Long-Elementen*

**Dim Data\_15**[1000] **As Long**

*Rem Dimensioniere das globale Feld Data\_5 mit  
Rem 20 x 75 Float-Elementen*

**Dim Data\_5**[20][75] **As Float**



## 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, *ohne* 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; *ohne* 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; Variablenamen 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 pi 3.141592654
```

Mit diesen Anweisungen können Sie im Quelltext anstelle von **Par\_1**, **Data\_1** und der Ziffernfolge die Namen **setpoint**, **measured** und **pi** 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 (auch Strings)
- *globale* eindimensionale Felder **Data\_n[n]** (auch FIFO-Felder)
- *globale* zweidimensionale Felder **Data\_n[n][m]**

Grundlagen zu Variablen und Datentypen finden Sie in Kapitel 4.2.3 auf Seite 80 sowie Informationen zu FIFO-Feldern unter dem Stichwort FIFO auf Seite 173.

## Syntax

```
Dim var1 {, var2, ...} As <VAR_TYPE>

Dim array1[dim1] {, array2[dim2]} As    <VAR_TYPE>
    {At <Mem_Type>}

Dim Data_n[dim1] {, Data_n[dim2]} As    <VAR_TYPE>
    {As FIFO} {At <Mem_Type>}

Dim Data_n[dim1][dim2] As <VAR_TYPE> {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...200 gewählt werden.	
<code>&lt;VAR_TYPE&gt;</code>	Datentyp: <code>Float</code> , <code>Long</code> für Felder zusätzlich: <code>String</code>	
<code>dim1, dim2</code>	Feldgröße: Anzahl ( $\geq 1$ ) der Feldelemente vom Typ <code>&lt;VAR_TYPE&gt;</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 (Default für Felder) <code>DM_Local</code> : interner Datenspeicher (Default für Variablen) nur ab T11 verfügbar: <code>EM_Local</code> : Zusätzlicher interner Programm- oder Datenspeicher	

## Bemerkungen

Die globalen Variablen **Par\_n** und **FPar\_n** dürfen nicht deklariert werden, weil sie vordefiniert sind.

Wenn Sie vom PC oder aus mehreren Prozessen auf Daten zugreifen wollen, ist dies nur über *globale* Variablen und Felder möglich.

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 ADwin-System.

String-Variablen sind *lokale* Felder vom Typ **STRING** (siehe „Strings“ auf Seite 93). Sie können nicht als FIFO deklariert werden.

## Siehe auch

Data\_n, Event:, FIFO, Finish:, Init:, LowInit:, "" String, „2-dimensionale Felder“ auf Seite 89, „Variablen und Felder im Datenspeicher“ auf Seite 87

## 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 1003 Long-Elementen als Fifo
Dim Data_15[1003] As Long As FIFO
```

```
Rem Dimensioniere das Feld TEXT mit
Rem 50 Elementen als String-Variable
Dim text[50] As String
```

## 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)
```

### 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 FIFO

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 im **Event**:-Abschnitt.

### Syntax

```
End
```

### Bemerkungen

Der Befehl **End** beendet die Ausführung des **Event**:-Abschnitts sofort. **Event**:- und beginnt die Ausführung des Abschnitts **Finish**:- (sofern vorhanden). Wenn im Abschnitt **Event**:- nach der Anweisung **End** weitere Programmzeilen folgen, werden sie nicht mehr ausgeführt.

In den anderen Programmabschnitten ist anstelle von **End** der Befehl **Exit** anzuwenden.

### Siehe auch

Exit, ProcessN\_Running, Restart\_Process, Start\_Process, Start\_Process\_Delayed, Stop\_Process

### 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:** {At <Mem\_Type>}

### Parameter

<Mem\_Type> nur für T11 ab Rev. E04: Speicherbereich, in dem der Programmabschnitt abgelegt wird.  
**PM\_Local**: interner Programmspeicher (Default)  
**EM\_Local**: Zusätzlicher interner Programm- oder Datenspeicher  
**DRAM\_Extern**: externer Datenspeicher

### Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 77.

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.

Bei dem Prozessortyp T11 kann jeder Programmabschnitt separat einem bestimmten Speicherbereich zugeordnet werden (siehe Kapitel 4.3.2 „Speicherbereiche“). Der große, aber langsamere Speicherbereich **DRAM\_Extern** sollte nur für zeitunkritische Programmabschnitte genutzt werden; meistens sind das die Abschnitte **LowInit:**, **Init:**, **Finish:**.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

### Siehe auch

Dim, LowInit:, Init:, Finish:

## **Beispiel**

```
Dim val_1 As Float
```

**Event:**

```
val_1 = -5.3
```

## Exit

**Exit** beendet einen Prozess in den Abschnitten **LowInit:**, **Init:** oder **Finish:**.

### Syntax

```
Exit
```

### Bemerkungen

Der Befehl **Exit** beendet die Ausführung des Prozesses und des Programmabschnitts sofort, weitere Programmzeilen im Abschnitt werden nicht mehr ausgeführt. Auch der Abschnitt **Finish** wird nicht mehr ausgeführt.

Verwenden Sie im Abschnitt **Event:** den Befehl **End**.

### Siehe auch

**End**, **ProcessN\_Running**, **Reset\_Event**, **Restart\_Process**, **Start\_Process**, **Start\_Process\_Delayed**, **Stop\_Process**

### Beispiel

```
Init:
  If (ADC(1) > 3000) Then 'Messen und Vergleichen
    Set_Digout(0)         'Dig. Ausgang setzen
    Exit                  'Diesen Prozess beenden
  EndIf
```

## Exp

**Exp** berechnet eine Potenz zur Basis e.

### Syntax

```
ret_val = Exp(val)
```

### Parameter

**val** Argument

FLOAT
-------

**ret\_val** Exponentialwert des Arguments zur Basis e.

FLOAT
-------

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,3µs, beim T10 bis zu 0,7µs, beim T11 bis zu 0,31µs.

### Siehe auch

LN, Log

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 5
```

```
val_2 = Exp(val_1)
```

```
'Ergebnis: val_2 =
```

```
'148,41...
```

## FIFO

Mit `Dim Data_n As FIFO` wird ein globales **DATA**-Feld als Ringspeicher dimensioniert.

### Syntax

```
Dim Data_n[m] As <ARR_TYPE> As FIFO
```

### Parameter

**Data\_n** Name des deklarierten **DATA**-Felds  
(n: 1...200).

**<ARR\_TYPE>** Festgelegter Datentyp: `Float`, `Long`

**m** Feldgröße ( $\geq 1$ ): Anzahl der Elemente vom Typ `ARR_TYPE` im Feld.

**CONST**

Beim Prozessortyp T11 ist der Wertebereich für **m** nur in Schritten einstellbar:

**LONG**

$m = 4 \times a + 3; a \geq 0$

### Bemerkungen

Sie können nur **DATA**-Felder als FIFO-Ringspeicher verwenden (siehe auch Kapitel 4.3.4 auf Seite 90). Dadurch kann ein FIFO-Feld nicht mehr gleichzeitig als „normales“ Feld verwendet werden.

FIFO-Felder (First in, first out) werden mit Datenzeigern verwaltet. Nach der Dimensionierung sollten Sie diese Datenzeiger mit dem Befehl **FIFO\_Clear** initialisieren, z.B. im Abschnitt **LowInit:** oder **Init:**. Die im FIFO enthaltenen Daten werden weder beim Initialisieren noch beim Dimensionieren verändert.

Wenn Sie schneller Daten in ein FIFO-Feld schreiben als auslesen, werden alte gespeicherte Daten überschrieben und gehen damit verloren. Zur Abhilfe können Sie die Befehle **FIFO\_Empty** und **FIFO\_FULL** verwenden.

Wenn Sie beim Prozessor T11 eine nicht erlaubte Feldgröße **m** angeben, wird das FIFO-Feld automatisch mit der nächstgrößeren erlaubten Feldgröße dimensioniert. Beispielsweise ändert der Compiler die Feldgröße `[1000]` automatisch in `[1003]`.

**Siehe auch**

Dim, Data\_n, FIFO\_Clear, FIFO\_Empty, FIFO\_Full

**Beispiel**

```
Rem Dimensioniere das globale Feld Data_20 mit  
Rem 1003 Long-Elementen als Fifo  
Dim Data_20[1003] As Long As FIFO
```

## FIFO\_Clear

**FIFO\_Clear** initialisiert den Schreib- und Lese-Zeiger eines FIFO-Felds.

### Syntax

```
FIFO_Clear(arraynum)
```

### Parameter

**arraynum**      Nummer des DATA-FIFO-Felds (1...200)

LONG
------

### Bemerkungen

Das Initialisieren des Schreib- und des Lese-Zeigers verändert die im Feld enthaltenen Daten nicht.

Die FIFO-Zeiger werden bei der Dimensionierung nicht initialisiert. Sie sollten dies im Abschnitt **LowInit:** oder **Init:** mit **FIFO\_Clear** tun.

Während des Programmlaufes ist das Initialisieren der FIFO-Zeiger sinnvoll, wenn Sie alle im Feld gesammelten Daten (z.B. wegen eines Messfehlers) verwerfen wollen.

### Siehe auch

FIFO, FIFO\_Empty, FIFO\_Full

**Beispiel**

```

Dim Data_1[20003] As Long As FIFO 'Deklaration
Dim reinit_fifo_flag As Long

Init:
    FIFO_Clear(1)           'FIFO-Zeiger
                             'initialisieren

Event:
    Rem Anzahl der freien Plätze im FIFO-Feld abfragen
    If (FIFO_Empty(1) > 1) Then
        'Analogen Eingang 1 messen und im FIFO speichern
        Data_1 = ADC(1)
    EndIf

    '
    '                               ' Programmtext
    '

    If (reinit_fifo_flag) Then 'z.B. Fehler geschehen
        FIFO_Clear(1)         'FIFO Zeiger
                              'initialisieren
    EndIf

```



**FIFO\_Empty** ermittelt die Anzahl der freien Elemente in einem FIFO-Feld.

### Syntax

```
ret_val = FIFO_Empty(arraynum)
```

### Parameter

**arraynum** Nummer des DATA-FIFO-Felds (1...200).

LONG

**ret\_val** Anzahl der freien Feldelemente.

LONG

### Bemerkungen

Wenn Sie Daten in ein FIFO-Feld schreiben wollen, sollten Sie vorher mit diesem Befehl überprüfen, ob noch genügend Platz im FIFO frei ist.

Beachten Sie beim Prozessor T11 die Dimensionierung in 4er-Schritten (siehe Seite 173).

### Siehe auch

FIFO, FIFO\_Clear, FIFO\_Full

### Beispiel

```
Dim Data_1[20003] As Long As FIFO'Deklaration
```

#### Init:

```
FIFO_Clear(1)           'FIFO-Zeiger
                        'initialisieren
```

#### Event:

```
Rem Anzahl der freien Plätze im FIFO-Feld abfragen
If (FIFO_Empty(1) > 1) Then
    Rem Analogen Eingang 1 messen und im FIFO speichern
    Data_1 = ADC(1)
EndIf
```

## FIFO\_Full

**FIFO\_Full** ermittelt die Anzahl der belegten Elemente in einem FIFO-Feld.

### Syntax

```
ret_val = FIFO_Full(arraynum)
```

### Parameter

<code>arraynum</code>	Nummer des DATA-FIFO-Felds (1...200).	LONG
<code>ret_val</code>	Anzahl der belegten Feldelemente (0...Dim). Beim Prozessortyp T11 ist der Wertebereich für Dim nur in Schritten einstellbar: $Dim = 4 \times a + 3; a \geq 0$	LONG

### Bemerkungen

Wenn Sie Daten aus einem FIFO-Feld lesen oder verwenden wollen, sollten Sie vorher mit diesem Befehl überprüfen, ob im FIFO noch Daten enthalten sind. Falls keine Daten mehr vorhanden sind, wird aus dem FIFO-Feld ein undefinierter Wert gelesen.

Beachten Sie beim Prozessor T11 die Dimensionierung in 4er-Schritten (siehe Seite 173).

### Siehe auch

FIFO, FIFO\_Clear, FIFO\_Empty

### Beispiel

```
Dim Data_1[20003] As Long As FIFO 'Deklaration

Init:
    FIFO_Clear(1)           'FIFO-Zeiger
                           'initialisieren

Event:
    Rem Abfrage, ob Daten im FIFO 'enthalten sind
    If (FIFO_Full(1) > 0) Then
        Rem Einen FIFO-Wert auf dem analogen Ausgang 1
        ausgeben
        DAC(1, Data_1)
    EndIf
```

## Finish:

Das Kennwort **Finish:** bezeichnet den Anfang des Programmabschnitts zur Schlussbearbeitung. Der Programmabschnitt hat in jedem Fall niedrige Priorität, Stufe 1.

### Syntax

**Finish:** {At <Mem\_Type>}

### Parameter

<Mem\_Type> nur für T11 ab Rev. E04: Speicherbereich, in dem der Programmabschnitt abgelegt wird.  
**PM\_Local**: interner Programmspeicher (Default)  
**EM\_Local**: Zusätzlicher interner Programm- oder Datenspeicher  
**DRAM\_Extern**: externer Datenspeicher

### Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 77.

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.

Bei dem Prozessortyp T11 kann jeder Programmabschnitt separat einem bestimmten Speicherbereich zugeordnet werden (siehe Kapitel 4.3.2 „Speicherbereiche“). Der große, aber langsamere Speicherbereich **DRAM\_Extern** sollte nur für zeitunkritische Programmabschnitte genutzt werden; meistens sind das die Abschnitte **LowInit:**, **Init:**, **Finish:**.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

### Siehe auch

Dim, LowInit:, Init:, Event:, ProcessN\_Running

## Beispiel

```
Dim val_1 As Float
```

**Finish:**

```
val_1 = -5.3
```

## FloToStr

**FloToStr** konvertiert eine Fließkomma-Zahl (float) in eine Zeichenfolge (String).

### Syntax

```
Import String.LI*      '*.LI9 für T9, *.LIA für  
    T10,                '*.LIB für T11  
  
FloToStr(val, string1[])
```

### Parameter

<code>val</code>	Zu wandelnder Wert.
<code>string1[]</code>	Erstellte Zeichenfolge im Format: {-}#.#####E{-}##

FLOAT

ARRAY

STRING

### Bemerkungen

Die Länge der Zeichenfolge variiert von 11 bis 13 Zeichen, abhängig von den Vorzeichen der Mantisse und des Exponenten.

### Siehe auch

"" String, Asc, Chr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, ValI

**Beispiel**

```
Import String.LI9          'String-Library für T9
```

```
Dim text[13] As String
Dim pi, number As Float
```

**Init:**

```
pi = 3.141592654
FPar_1 = -pi^-20
```

**Event:**

```
Rem Fließkomma-Zahl in einen String wandeln
```

```
FloToStr(FPar_1, text)
```

```
Par_1 = text[1]          'String-Länge = 13
Par_2 = text[2]          'ASCII-Zeichen 2Dh = "-"
Par_3 = text[3]          'ASCII-Zeichen 3lh = "1"
Par_4 = text[4]          'ASCII-Zeichen 2Eh = "."
Par_5 = text[5]          'ASCII-Zeichen 3lh = "1"
Par_6 = text[6]          'ASCII-Zeichen 34h = "4"
Par_7 = text[7]          'ASCII-Zeichen 30h = "0"
Par_8 = text[8]          'ASCII-Zeichen 32h = "2"
Par_9 = text[9]          'ASCII-Zeichen 35h = "5"
Par_10 = text[10]         'ASCII-Zeichen 35h = "5"
Par_11 = text[11]         'ASCII-Zeichen 45h = "E"
Par_12 = text[12]         'ASCII-Zeichen 2Dh = "-"
Par_13 = text[13]         'ASCII-Zeichen 3lh = "1"
Par_14 = text[14]         'ASCII-Zeichen 30h = "0"
Par_15 = text[15]         'String-Ende-Zeichen = 0
```

## Flo40ToStr

Nur für Prozessor T11: **Flo40ToStr** konvertiert eine Fließkomma-Zahl (float) von 40 Bit in eine Zeichenfolge (String).

### Syntax

```
Import String.LIB          '*LIB für T11
Flo40ToStr(val, stringl[])
```

### Parameter

<code>val</code>	Zu wandelnder Wert
<code>stringl[]</code>	Erstellte Zeichenfolge im Format: {-}#.#####E{-}##

FLOAT

ARRAY

STRING

### Bemerkungen

Die Länge der Zeichenfolge variiert von 13 bis 15 Zeichen, abhängig von den Vorzeichen der Mantisse und des Exponenten.

### Siehe auch

"" String, Asc, Chr, FloToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, Vall

**Beispiel**

```
Import String.LIB          'String-Library für T11
```

```
Dim text[15] As String
Dim pi, number As Float
```

**Init:**

```
pi = 3.141592654
FPar_1 = -pi^-20
```

**Event:**

*Rem Fließkomma-Zahl in einen String wandeln*

```
Flo40ToStr(FPar_1, text)
Par_1 = text[1]          'String-Länge = 15
Par_2 = text[2]          'ASCII-Zeichen 2Dh = "-"
Par_3 = text[3]          'ASCII-Zeichen 31h = "1"
Par_4 = text[4]          'ASCII-Zeichen 2Eh = "."
Par_5 = text[5]          'ASCII-Zeichen 31h = "1"
Par_6 = text[6]          'ASCII-Zeichen 34h = "4"
Par_7 = text[7]          'ASCII-Zeichen 30h = "0"
Par_8 = text[8]          'ASCII-Zeichen 32h = "2"
Par_9 = text[9]          'ASCII-Zeichen 35h = "5"
Par_10 = text[10]         'ASCII-Zeichen 35h = "6"
Par_11 = text[11]         'ASCII-Zeichen 35h = "4"
Par_12 = text[12]         'ASCII-Zeichen 35h = "7"
Par_13 = text[13]         'ASCII-Zeichen 45h = "E"
Par_14 = text[14]         'ASCII-Zeichen 2Dh = "-"
Par_15 = text[15]         'ASCII-Zeichen 31h = "1"
Par_16 = text[16]         'ASCII-Zeichen 30h = "0"
Par_17 = text[17]         'String-Ende-Zeichen = 0
```



## 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 Prozessor des **ADwin**-Systems 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**

```
Dim index As Long
Dim sinus[360] As Float 'Feld für Sinus-Werte
Dim pi As Float

Init:
  pi = 3.14159
  Rem Sinuswerte in Grad-Schritten berechnen (0° bis
    359°)
  For index = 1 To 360
    sinus[index] = (2047*Sin((index - 1) * 2*pi/360))
  Next index
  index = 1 'Zählindex initialisieren

Event:
  DAC(1, sinus[index]) 'Amplitudenwert ausgeben
  Inc index 'Zählindex erhöhen
  Rem Ab 360 Grad wieder bei 0 beginnen
  If (index > 360) Then index = 1
```

## Function ... EndFunction

`Function...EndFunction` definiert ein Funktions-Makro mit Übergabeparametern und einem Rückgabewert.

### Syntax

```
Function macro_name ({val_1, val_2, ...}) As <VAR_
TYPE>

    {Dim var As <VAR_TYPE>}

    ... 'Anweisungsblock

    macro_name = ... 'Rückgabewert zuweisen
EndFunction
```

### Parameter

`macro_name` Name der Funktion und Rückgabewert, Datentyp `<VAR_TYPE>`

`val_1, val_2` Namen der Übergabeparameter; für Felder ist die Syntax mit Dimensionsklammern erforderlich: `array[]` oder `Data_n[]`.

`<VAR_TYPE>` Datentyp der Funktion bzw. des Rückgabewerts: `Float`, `Long`; nicht aber `String`

FLOAT

LONG

STRING

### Bemerkungen

Allgemeine Informationen über Makros finden Sie in Kapitel 4.5.1 auf Seite 100.

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:** / **LowInit:**
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 **LowInit:**, **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, Lib\_Function ... Lib\_EndFunction, Lib\_Sub ... Lib\_EndSub

### Beispiel

```
Function average(w1, w2, w3) As Float
    Rem Die Funktion berechnet den Mittelwert aus den Werten
    Rem w1, w2 und w3
    Dim sum As Float
    sum = w1 + w2 + w3
    average = sum/3
EndFunction
```

Ein Aufruf der Funktion erfolgt z.B. mit den Programmzeilen:

```
x = average(x1, x2, x3)
DAC(1, average(x1, x2, x3))
```

Die gleiche Funktion mit einem Feld als Übergabe-Parameter:

```
Function average_array(array[]) As Float
    average_array=(array[1] + array[2] + array[3])/3
EndFunction
```

Der Aufruf dieser Funktion erfolgt wieder in ähnlicher Weise (allerdings *ohne* die Dimensionsklammern):

```
x = average_array(array)
DAC(1, average_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.

Prozessor	Priorität	
	Hoch	Niedrig
T9	25ns	100µs
T10	25ns	50µs
T11	3,3ns	3,3ns = 0,003µs

## 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

<SYSPAR>	value	Bedeutung
ADwin_ SYSTEM	ADWIN_CARD ADWIN_L16 ADWIN_GOLD ADWIN_GOLDII ADWIN_PRO ADWIN_PROII	Einstellung „System“ im Fenster „Compiler Options“.
Processor	T9 T10 T11	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 niederprioreres Processdelay auf 800µs setzen
#If Processor = T11 Then
    Rem T11: 800µs = 240000 x 3,3ns
    Processdelay = 240000
#Else
    #If Processor = T10 Then
        Rem T10: 800µs = 16 x 50µs
        Processdelay = 16
    #Else
        Rem T9: 800µs = 8 x 100µs (auch andere CPUs)
        Processdelay = 8
    #EndIf
#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>.LI9</code> für T9, <code>.LIA</code> für T10, <code>.LIB</code> für T11.	<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 52) 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 *ADbasic* enthalten:

<code>String.li9,</code> <code>String.liA,</code> <code>String.liB</code>	Befehle für String-Operationen.
---	---------------------------------

<code>FFT.li9, FFT.liA,</code>	Befehle für die schnelle Fourier-Transfor-
<code>FFT.liB</code>	mation.
<code>math.li9, math.liA,</code>	Mathematik-Befehle.
<code>math.liB</code>	

## Siehe auch

`#Include, Lib_Function ... Lib_EndFunction, Lib_Sub ... Lib_EndSub`

## Beispiel

```
Import String.li9      'Importiert die
                        'String-Library
                        'für den Prozessor T9
Rem Benutzerdefinierte Library für T10 importieren
Import C:\MyFiles\ADwinLibs\dig2volt.lia
```

## 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	CONST STRING
<code>path</code>	Vollständiger Pfad mit Laufwerk oder relativer Pfad.	CONST STRING

### Bemerkungen

Allgemeine Informationen über Include-Dateien finden Sie in Kapitel 4.5.2 auf Seite 101.

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 52) 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 *ADbasic* 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. Alternativ verwenden Sie einen der Textbausteine aus der Gruppe „Hardware“.



Beachten Sie bitte: Eine Zeile mit **#Include**-Anweisung darf höchstens 136 Zeichen lang sein (Zeilenlänge für alle anderen Zeilen, siehe Seite 142). 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: {At <Mem_Type>}
```

### Parameter

<Mem\_Type> nur für T11: Speicherbereich, in dem der Programmabschnitt abgelegt wird.  
**PM\_Local**: interner Programmspeicher (Default)  
**EM\_Local**: Zusätzlicher interner Programm- oder Datenspeicher  
**DRAM\_Extern**: externer Datenspeicher

### Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 77.

Der Programmabschnitt **Init:** wird einmal durchlaufen, sobald der Prozess gestartet wird und der Programmabschnitt **LowInit:** (falls vorhanden) beendet ist. Die Zeit zwischen dem letzten Befehl im Abschnitt **Init:** und dem Beginn des Abschnitts **Event:** beträgt  $1 \dots 2 \times \text{Processdelay}$ .

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.

Bei dem Prozessortyp T11 kann jeder Programmabschnitt separat einem bestimmten Speicherbereich zugeordnet werden (siehe Kapitel 4.3.2 „Speicherbereiche“). Der große, aber langsamere Speicherbereich **DRAM\_Extern** sollte nur für zeitunkritische Programmabschnitte genutzt werden; meistens sind das die Abschnitte **LowInit:**, **Init:**, **Finish:**.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.

**Siehe auch**

Dim, LowInit:, Event:, Finish:, Processdelay

**Beispiel**

```
Dim val_1 As Float
Init:
    val_1 = -5.3
```



## IO\_Sleep

**IO\_Sleep** lässt Befehle für Ein- und Ausgänge des Gold II-Systems für eine bestimmte Zeit warten.

### Syntax

**IO\_Sleep** (val)

### Parameter

**val**                      Anzahl (12, 14 ...  $715827879 \approx 2^{31} / 3$ ) der zu wartenden Zeiteinheiten in 10ns.  
Nur gerade Zahlen sind möglich. Eine ungerade Anzahl von Zeiteinheiten wird automatisch um 1 verringert.

LONG

### Bemerkungen

Alternativ gibt es die Anweisung **CPU\_Sleep** (siehe auch Kapitel 5.2.4 „Wartezeit genau einstellen“).

Die Anweisung **IO\_Sleep** wird eingesetzt, wenn zwischen 2 Zugriffen auf Ein-/Ausgänge eine definierte Zeit gewartet werden muss. Die Gesamtzeit ist die Summe aus der Ausführungszeit des I/O-Zugriffs und der Wartezeit durch **IO\_Sleep**.

Die Wartezeit sollte grundsätzlich kleiner sein als die mit **PROCESSDELAY** eingestellte Zykluszeit.

Bei hochprioren Prozessen können ungeeignete Argumente zu einem Abbruch der Kommunikation führen:



- Stellen Sie sicher, dass das Argument immer einen Wert größer 12 hat, weil sonst extrem lange Wartezeiten entstehen können.
- Verwenden Sie sehr große Werte mit Bedacht, weil dann die Kommunikation mit dem PC sehr lange unterbrochen ist (Gefahr eines Timeout).

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. Die Zeitspanne kann hier schwanken, weil sie von mehreren Voraussetzungen abhängt.
- Das Argument ist ein Feld.
- Das Argument ist ein Fließkomma-Wert.

### Siehe auch

CPU\_Sleep, NOP

### Beispiel

```
#Include ADwinGoldII.inc
```

#### Event:

```
Set_Mux1(11010b)      'Mux 1: Kanal und  
                       'Verstärkung setzen  
IO_Sleep(200)          '2 µs (=200*10ns) warten  
                       '= max. Einschwingzeit  
                       'des MUX  
Start_Conv(1)          'Start AD-Wandlung ADC1  
...
```

### 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> {At <Mem_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> , <b>LONG</b> .
<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> .
<code>&lt;Mem_Type&gt;</code>	Sinnvoll nur für Prozessor T10: Speicher, in dem die Variablen abgelegt werden; nur gemeinsam mit Fel- dern verfügbar: <code>DRAM_Extern</code> : externer Datenspeicher. <code>DM_Local</code> : interner Datenspeicher.

**Bemerkungen**

Allgemeine Informationen über Library-Dateien finden Sie in Kapitel 4.5.3 auf Seite 101.

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 (nur eindimensional) deklarieren und verwenden.  
Deklarieren Sie Variablen immer am Anfang, keinesfalls außerhalb des Unterprogramms.
- globale Variablen und Felder verwenden, wenn sie als Parameter übergeben werden.
- nur eindimensionale Felder bearbeiten.  
Sie können zweidimensionale Felder als Parameter übergeben, diese werden aber innerhalb der Funktion wie eindimen-

sionale Felder behandelt (siehe auch Kapitel 4.3.3 auf Seite 89).

- dem Funktionsnamen einen Wert zuweisen, damit dieser zum Rückgabewert an die aufrufende Stelle wird.

Innerhalb einer Library-Funktion ist *nicht* erlaubt:

- Prozess-Abschnitte wie **LowInit:**, **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.

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.

Sie sollten alle Übergabeparameter immer **At <Mem\_Type>** definieren, um damit kostbare Prozessorzeit zu sparen (wobei **<Mem\_Type>** zur Deklaration des Übergabeparameters im aufrufenden Programm passen muss, siehe **Dim**). Anderenfalls muss die Library-Funktion zur Laufzeit herausfinden, in welchem Speicher die Übergabeparameter abgelegt sind.



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.LI9
#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> {At <Mem_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> .
<code>&lt;Mem_Type&gt;</code>	Sinnvoll nur für Prozessor T10: Speicher, in dem die Variablen abgelegt werden; nur gemeinsam mit Feldern verfügbar: <code>DRAM_Extern</code> : externer Datenspeicher. <code>DM_Local</code> : interner Datenspeicher.

### Bemerkungen

Allgemeine Informationen über Bibliotheken (Libraries) finden Sie in Kapitel 4.5.3 auf Seite 101.



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 (nur eindimensional) deklarieren und verwenden.  
Deklarieren Sie Variablen immer am Anfang, keinesfalls außerhalb des Unterprogramms.
- globale Variablen und Felder verwenden, wenn sie als Parameter übergeben werden.
- nur eindimensionale Felder bearbeiten.  
Sie können zweidimensionale Felder als Parameter übergeben, diese werden aber innerhalb der Funktion wie eindimensionale Felder behandelt (siehe auch Kapitel 4.3.3 auf Seite 89).

Innerhalb eines Library-Unterprogramms ist *nicht* erlaubt:

- Prozess-Abschnitte wie `LowInit:`, `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.

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.

Betrifft nur Prozessor T10: Sie sollten alle Übergabeparameter immer `At <Mem_Type>` definieren, um damit kostbare Prozessorzeit zu sparen (wobei `<Mem_Type>` zur Deklaration des Übergabeparameters im aufrufenden Programm passen muss, siehe `Dim`). Anderenfalls muss



das Library-Unterprogramm zur Laufzeit herausfinden, in welchem Speicher die Übergabeparameter abgelegt sind.

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.LI9
#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 []

```

## LN

**LN** liefert den natürlichen Logarithmus (zur Basis e) eines Arguments.

### Syntax

```
ret_val = LN(val)
```

### Parameter

`val`                      Argument

FLOAT
-------

`ret_val`                natürlicher Logarithmus des Arguments

FLOAT
-------

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,45µs, beim T10 bis zu 0,7µs, beim T11 bis zu 0,37µs.

### Siehe auch

Log, Exp

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 5.3
```

```
val_2 = LN(val_1)                'Ergebnis: val_2 = 1.667...
```

## LngToStr

**LngToStr** konvertiert einen ganzzahligen Wert in einen String.

### Syntax

```
Import String.LI*      '*.LI9 für T9, *.LIA für  
    T10,  
                        '*.LIB für T11  
  
LngToStr(value, stringl[])
```

### Parameter

<code>value</code>	Wert, der gewandelt werden soll
<code>stringl[]</code>	Ergebnis-Zeichenfolge

LONG

ARRAY

STRING

### Bemerkungen

Die erzeugte String-Länge ist nicht konstant, sondern richtet sich nach der zu konvertierenden Zahl und dem Vorzeichen. Es sind String-Längen von 1 bis 11 Zeichen möglich.

Informationen zum String-Aufbau finden Sie in Kapitel 4.3.5 auf Seite 93.

### Siehe auch

"" String, + (String-Addition), Asc, Chr, FloToStr, Flo40ToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, Vall

**Beispiel**

```

Import String.LI9
Dim digits[11] As String 'Ergebnis-String
Dim a As Long

```

**Init:**

```

a = -1234567890

```

**Event:**

```

LngToStr(a,digits)      'zum String konvertieren
Par_1=digits[1]         'String-Länge = 11
Par_2=digits[2]         'ASCII-Zeichen 45 = "-"
Par_3=digits[3]         'ASCII-Zeichen 49 = "1"
Par_4=digits[4]         'ASCII-Zeichen 50 = "2"
Par_5=digits[5]         'ASCII-Zeichen 51 = "3"
Par_6=digits[6]         'ASCII-Zeichen 52 = "4"
Par_7=digits[7]         'ASCII-Zeichen 53 = "5"
Par_8=digits[8]         'ASCII-Zeichen 54 = "6"
Par_9=digits[9]         'ASCII-Zeichen 55 = "7"
Par_10=digits[10]       'ASCII-Zeichen 56 = "8"
Par_11=digits[11]       'ASCII-Zeichen 57 = "9"
Par_12=digits[12]       'ASCII-Zeichen 48 = "0"
Par_13=digits[13]       'String-Ende-Zeichen = 0

```

## Log

**Log** liefert den dekadischen Logarithmus (zur Basis 10) eines Arguments.

### Syntax

```
ret_val = Log(val)
```

### Parameter

val                    Argument

FLOAT

ret\_val                dekadischer Logarithmus des Arguments

FLOAT

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,5µs, beim T10 bis zu 0,75µs, beim T11 bis zu 0,38µs.

### Siehe auch

LN, Exp

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 5.3
```

```
val_2 = Log(val_1)     'Ergebnis: val_2 = 0.724...
```

## LowInit:

Das Kennwort **LowInit:** bezeichnet den Anfang eines Programmabschnitts zur Initialisierung. Der Programmabschnitt hat in jedem Fall niedrige Priorität, Stufe 1.

### Syntax

```
LowInit: {At <Mem_Type>}
```

### Parameter

<Mem\_Type> nur für T11 ab Rev. E04: Speicherbereich, in dem der Programmabschnitt abgelegt wird.  
**PM\_Local:** interner Programmspeicher (Default)  
**EM\_Local:** Zusätzlicher interner Programm- oder Datenspeicher  
**DRAM\_Extern:** externer Datenspeicher

### Bemerkungen

Zur Übersicht der Programmabschnitte siehe Kapitel 4.1.1 auf Seite 77.

Der Abschnitt wird bei jedem Start des Prozesses einmalig durchlaufen und dient zur Initialisierung, z.B. von Variablen oder Datenleitungen. **LowInit:** wird immer vor dem Abschnitt **Init:** ausgeführt (falls vorhanden).



Der Abschnitt **LowInit:** ist für umfangreiche Initialisierungs-Sequenzen geeignet, da er (wegen seiner niedriger Priorität) unterbrochen werden kann.

Bei dem Prozessortyp T11 kann jeder Programmabschnitt separat einem bestimmten Speicherbereich zugeordnet werden (siehe Kapitel 4.3.2 „Speicherbereiche“). Der große, aber langsamere Speicherbereich **DRAM\_Extern** sollte nur für zeitunkritische Programmabschnitte genutzt werden; meistens sind das die Abschnitte **LowInit:**, **Init:**, **Finish:**.

Beim Prozessormodul Pro-CPU T11 kann der Speicherbereich erst ab Rev. E04 festgelegt werden.



## Siehe auch

Dim, Init, Event, Finish:

## Beispiel

```
Dim val_1 As Float
LowInit:
    val_1 = -5.3
```

## Max\_Float

**Max\_Float** liefert den größeren von 2 Float-Werten.

### Syntax

```
ret_val = Max_Float(val1, val2)
```

### Parameter

<code>val_1</code>	Vergleichswert 1	<div>FLOAT</div>
<code>val_2</code>	Vergleichswert 2	<div>FLOAT</div>
<code>ret_val</code>	Der größere der beiden Vergleichswerte.	<div>FLOAT</div>

### Bemerkungen

- / -

### Siehe auch

AbsF, Max\_Long, Min\_Long, ValF

### Beispiel

**Event:**

```
FPar_10 = Max_Float(FPar_1, FPar_2)
```

## Min\_Float

**Min\_Float** liefert den kleineren von 2 Float-Werten.

### Syntax

```
ret_val = Min_Float(val1, val2)
```

### Parameter

val\_1            Vergleichswert 1

Float

val\_2            Vergleichswert 2

Float

ret\_val          Der kleinere der beiden Vergleichswerte.

Float

### Bemerkungen

- / -

### Siehe auch

AbsF, Max\_Long, Min\_Long, ValF

### Beispiel

**Event:**

```
FPar_10 = Min_Float(FPar_1, FPar_2)
```

## Max\_Long

**Max\_Long** liefert den größeren von 2 ganzzahligen Werten.

### Syntax

```
ret_val = Max_Long(val1, val2)
```

### Parameter

<code>val_1</code>	Vergleichswert 1	LONG
<code>val_2</code>	Vergleichswert 2	LONG
<code>ret_val</code>	Der größere der beiden Vergleichswerte.	LONG

### Bemerkungen

- / -

### Siehe auch

Absl, Max\_Float, Min\_Long, Vall

### 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

val\_1            Vergleichswert 1

LONG

val\_2            Vergleichswert 2

LONG

ret\_val          Der kleinere der beiden Vergleichswerte.

LONG

### Bemerkungen

- / -

### Siehe auch

Absl, Max\_Long, Min\_Float, Vall

### Beispiel

**Event:**

```
Par_10 = Min_Long(Par_1, Par_2)
```

## MemCpy

Nur für Prozessor T11: **MemCpy** kopiert eine bestimmte Anzahl von Feldelementen aus einem Quellfeld in ein Zielfeld.

### Syntax

```
MemCpy(source[i1], dest[i2], count)
```

### Parameter

<code>source[]</code>	Name des Quellfelds.	<div>ARRAY</div> <div>LONG</div> <div>FLOAT</div> <div>STRING</div>
<code>i1</code>	Index ( $\geq 1$ ) des ersten kopierten Feldelements.	<div>LONG</div>
<code>dest[]</code>	Name des Zielfelds.	<div>ARRAY</div> <div>LONG</div> <div>FLOAT</div> <div>STRING</div>
<code>i2</code>	Index ( $\geq 1$ ) des ersten Feldelements, das beschrieben wird.	<div>LONG</div>
<code>count</code>	Anzahl ( $\geq 1$ ) der zu kopierenden Feldelemente.	<div>LONG</div>

### Bemerkungen

**MemCpy** ist die einfache und deutlich schnellere Alternative zum Kopieren von Daten in einer `For...Next`-Schleife.

Die Anweisung darf nicht zum Kopieren von und in FIFO-Felder benutzt werden.



Beachten Sie bitte: Die Datentypen von Quellfeld und Zielfeld müssen übereinstimmen und das Zielfeld muss groß genug dimensioniert sein, um alle kopierten Daten aufnehmen zu können. Der Zugriff auf zu große oder zu kleine Elementindexe des Zielfelds kann mit dem Debug-Modus überwacht werden (siehe Option Debug mode auf Seite 53). Für das Quellfeld ist die Überwachung nicht möglich.

## Siehe auch

Dim

## Beispiel

```
Dim Data_1[75], Data_2[100] As Float
```

### Event:

*Rem 70 Feldelemente aus Data\_1 nach Data\_2 kopieren*

```
MemCpy (Data_1[5], Data_2[30], 70)
```

## NOP

**NOP** (No OPeration) lässt den Prozessor für die Zeit von einem Taktzyklus warten.

### Syntax

**NOP**

### Bemerkungen

Diese Anweisung benötigt in der Regel einen Prozessor-Taktzyklus:

T9	25,0ns
T10	12,5ns
T11	3,3ns

Sie können mit dieser Anweisung erforderliche Wartezeiten überbrücken (beispielsweise nach **Set\_Mux**), wenn es keine sinnvolle andere Verwendung der Prozessorzeit gibt.

### Siehe auch

CPU\_Sleep, P1\_Sleep, P2\_Sleep, Sleep





## 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

<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 **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*

## P1\_Sleep

Nur für Prozessor T11: **P1\_Sleep** lässt den Bus des Pro I-Systems für eine bestimmte Zeit warten.

### Syntax

**P1\_Sleep** (*val*)

### Parameter

*val*

Anzahl der zu wartenden Zeiteinheiten in 10ns:  
bei Konstanten: 7...715827879 ( $\approx 2^{30} / 1,5$ )  
bei Variablen: 9...715827879

LONG
------

### Bemerkungen

Alternativ gibt es die Anweisungen **CPU\_Sleep** und **P2\_Sleep** (siehe auch Kapitel 5.2.4 „Wartezeit genau einstellen“). Verwenden Sie bei Prozessoren bis T10 die Anweisung **Sleep**.

Die Anweisung **P1\_Sleep** wird eingesetzt, wenn zwischen 2 Zugriffen auf Module am Pro I-Bus eine definierte Zeit gewartet werden muss.

Die Wartezeit sollte grundsätzlich kleiner sein als die mit **PROCESSDELAY** eingestellte Zykluszeit.



Die Anweisung **P1\_Sleep** kann bei einem hochprioren Prozess nicht unterbrochen werden. Bitte beachten Sie, dass sehr große Werte bei hochprioren Prozessen zu einem Abbruch der Kommunikation führen können.

Verwenden Sie keine Werte, die kleiner sind als die angegebenen Mindestwerte.

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. Die Zeitspanne kann hier schwanken, weil sie von mehreren Voraussetzungen abhängt.
- Das Argument ist ein Feld.
- Das Argument ist ein Fließkomma-Wert.

### Siehe auch

CPU\_Sleep, NOP, P2\_Sleep, Sleep

### Beispiel

**Event:**

```
Set_Mux(1,0)           'Multiplexer des Moduls 1 setzen
P1_Sleep(250)          '2.5 µs (=250*10ns) warten
                        '= Einschwingzeit des MUX
Start_Conv(1)          'Konvertierung starten
...
```

## P2\_Sleep

Nur für Prozessor T11: **P2\_Sleep** lässt den Bus des Pro II-Systems für eine bestimmte Zeit warten.

### Syntax

**P2\_Sleep** (*val*)

### Parameter

*val*

Gerade Anzahl ( $14 \dots 715827878 \approx 2^{30} / 1,5$ ) der zu wartenden Zeiteinheiten in 10ns. Eine ungerade Anzahl ist nicht erlaubt.

LONG

### Bemerkungen

Alternativ gibt es die Anweisungen **CPU\_Sleep** und **P1\_Sleep** (siehe auch Kapitel 5.2.4 „Wartezeit genau einstellen“). Verwenden Sie bei Prozessoren bis T10 die Anweisung **Sleep**.

Die Anweisung **P2\_Sleep** wird eingesetzt, wenn zwischen 2 Zugriffen auf Module am Pro II-Bus eine definierte Zeit gewartet werden muss.

Die Wartezeit sollte grundsätzlich kleiner sein als die mit **PROCESSDELAY** eingestellte Zykluszeit.



Stellen Sie unbedingt (insbesondere bei Variablen) sicher, dass das Argument in keinem Fall einen Wert kleiner 1 hat, weil sonst das ADwin-System in einen instabilen Zustand geraten kann. Beachten Sie bitte auch, dass sehr große Werte bei hochprioren Prozessen zu einem Abbruch der Kommunikation führen können.



Die Anweisung **P2\_Sleep** kann bei einem hochprioren Prozess nicht unterbrochen werden. Bitte beachten Sie, dass sehr große Werte bei hochprioren Prozessen zu einem Abbruch der Kommunikation führen können.

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.
- Das Argument ist ein Fließkomma-Wert.

### Siehe auch

CPU\_Sleep, NOP, P1\_Sleep, Sleep

### Beispiel

**Event:**

```
P2_Set_Mux(0)           'Multiplexer setzen
P2_Sleep(210)           '2.1 µs (=210*10ns) warten
                        '= Einschwingzeit des MUX
P2_Start_Conv(1)        'Konvertierung starten
...
```

## Peek

**Peek** liest den Inhalt einer bestimmten Speicherstelle auf dem *ADwin*-System.

### Syntax

```
ret_val = Peek(addr)
```

### Parameter

**addr** Adresse der auszulesenden Speicherstelle.

LONG
------

**ret\_val** Inhalt der Speicherstelle.

LONG
------

### Beschreibung

Sie finden eine Übersicht der Registeradressen (*Gold* und *Light-16*) in Ihrer Hardware-Dokumentation.

### Siehe auch

Poke, Read\_Timer

### Beispiel

Die unten stehende Anweisung liest den Wert der Speicheradresse `20400030h`, das ist bei *ADwin-Gold* das Datenregister des ADC1 und enthält den gewandelten Analogwert.

*Rem Speicherstellen eines ADwin-Gold auslesen*

```
val = Peek(20400030h)
```



## Poke

**Poke** schreibt einen Wert in eine bestimmte Speicherstelle auf dem ADwin-System.

### Syntax

**Poke** (addr, value)

### Parameter

addr                      Adresse der beschreibenden Speicherstelle

LONG

value                    Zu schreibender Wert

LONG

### Bemerkungen

Sie überschreiben mit **Poke** die angegebene Speicheradresse. Informationen, die zuvor dort gespeichert waren, gehen unwiderruflich verloren.

Überschreiben Sie in keinem Fall Speicheradressen, deren Funktion Ihnen unbekannt ist. Es können dadurch für den Betrieb wichtige Daten, Prozesse oder gar das Betriebssystem zerstört werden. Falls dies dennoch geschieht, gehen vorhandene Messdaten verloren. Booten Sie das ADwin-System und laden die Prozesse erneut.



Sie finden eine Übersicht der Registeradressen (*Gold* und *Light-16*) in Ihrer Hardware-Dokumentation.

### Siehe auch

Peek, Poke, Read\_Timer

### Beispiel

```
Rem Speicherstellen eines ADwin-Gold verändern
Rem DAC-Register 1 beschreiben: 3072 (=+5V im ±10V Bereich)
Poke (20400050h, 3072)
Poke (20400010h, 011b) 'Ausgabe auf allen DAC starten
Poke (204000C0h, 111100b) 'Setze Ausgänge DIO18...DIO21 auf High
```

## 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 abhängig von der Priorität des Prozesses und vom Prozessortyp:

Prozessor	Priorität	
	Hoch	Niedrig
T9	25ns	100µs
T10	25ns	50µs
T11	3,3ns	3,3ns = 0,003µs

Wählen Sie bei hochprioren Prozessen eine ausreichend großes Processdelay, um eine Überlastung des ADwin-Systems zu vermeiden (siehe auch Kapitel 6.1.4 auf Seite 120). 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 250ms 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:** / **LowInit:** einen Wert zuweisen. Sie überschreiben damit ggf. die Voreinstellung, die Sie in der Entwicklungsumgebung *ADbasic* 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.

Ein Beschreiben im Abschnitt **Event:** sollte gleich am Abschnittsanfang geschehen, weil sich das Ändern der Variablen sofort auf den Aufruf des nächsten Prozesszyklus auswirkt. Anderenfalls kann die zeitlich exakte Bearbeitung der Prozesszyklen außer Takt geraten.

### Siehe auch

Read\_Timer, Kapitel 6.2.1 „Processdelay“

### Beispiel

#### **Init:**

```
Rem Zykluszeit einstellen
Processdelay = 40000
Rem Für T9 und T10, hohe Priorität: 1 ms
Rem Für T11, hohe/niedrige: 0.133 ms
...
```

Wenn Sie eine längere Zykluszeit benötigen als mit **Processdelay** einstellbar ist, können Sie eine Hilfsvariable verwenden:

**Init:**

```
Rem Max. Zykluszeit einstellen
Processdelay = 2147483647
Rem Für T9 und T10, hohe Priorität: 53,7s
Rem Für T11, hohe+niedrige Priorität: 7,2s
Rem Hilfsvariable initialisieren
Par_1 = 0
```

**Event:**

```
Inc Par_1
Rem 100fache Zykluszeit verwenden
Rem Für T9 und T10, hohe Priorität: 89,5 Min.
Rem Für T11, hohe+niedrige Priorität: 12 Min.
If (Par_1 = 100) Then
  Par_1 = 0
  Rem Programm ausführen
  ...
EndIf
```

## Process\_Error

`Processn_Error` gibt den (in der Regel) zuletzt aufgetretenen Fehler des aktuellen Prozesses zurück.

### Syntax

```
ret_val = Processn_Error
```

### Parameter

<code>ret_val</code>	Nummer des zuletzt aufgetretenen Fehlers im Prozess: <span style="border: 1px solid black; padding: 2px;">LONG</span>
	0: kein Fehler
	1: Division durch Null
	2: Wurzel aus negativer Zahl
	10: Zugriff auf eine zu große Elementnummer eines globalen Felds.
	11: Zugriff auf eine zu kleine Elementnummer ( $\leq 0$ ) eines globalen Felds.
	12: Zugriff auf eine zu große Elementnummer eines lokalen Felds.
	13: Zugriff auf eine zu kleine Elementnummer ( $\leq 0$ ) eines lokalen Felds.
	30: FIFO-Index ist kein FIFO.

### Bemerkungen

Der Rückgabewert der Systemvariablen ist nur definiert, wenn der Debug-Modus eingeschaltet ist (siehe Option Debug mode, Seite 53). Die Variable kann nur gelesen werden.

### Siehe auch

`ProcessN_Running`, `Start_Process`, `Stop_Process`

### Beispiel

```
Event:
  Par_10 = Sqrt(Par_12)
  Rem letzten Fehler im Prozess lesen
  Par_2 = PROceSS_Error
```

## ProcessN\_Running

Die Systemvariable `ProcessN_Running` gibt den aktuellen Status eines bestimmten Prozesses zurück.

### Syntax

```
ret_val = ProcessN_Running
```

### Parameter

`n` Prozessnummer `n` (0...12, 15)

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, Exit, Restart\_Process, Start\_Process, Start\_Process\_Delayed, Stop\_Process

### 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.

LONG

### Bemerkungen

Die Systemvariable kann nur gelesen werden.

Es gibt im ADwin-System 2 Zähler (32 Bit), die in unterschiedlichen Zeiteinheiten zählen:

Prozesspriorität	T9	T10	T11
hoch	25ns	25ns	3,3ns
niedrig	100µs	50µs	3,3ns

Sie können eine Zeitdifferenz aus der Differenz von 2 Zählerständen ermitteln. Beachten Sie dabei bitte, dass ein gelesener Zählerstand nach einer bestimmten Zeit wieder erneut erreicht wird. Dieser Zählerüberlauf hängt von den oben angegebenen Zeiteinheiten ab:

Prozesspriorität	T9	T10	T11
hoch	107,4s	107,4s	14,3s
niedrig	119,3h	59,7h	14,3s

### 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>	ADbasic-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
```



## Reset\_Event

**Reset\_Event** löscht alle externen Event-Signale, die zur Ausführung anstehen.

### Syntax

**Reset\_Event**

### Bemerkungen

Die Anweisung ist nur bei dem extern gesteuerten Prozess zulässig, nicht für zeitgesteuerte Prozesse.

Wir empfehlen, die Anweisung am Ende des Abschnitts **Init:** auszuführen; beim Prozessor T11 ist dies sogar zwingend erforderlich. Damit stellen Sie sicher, dass ein zu frühes – während der Initialisierung aufgetretenes – Event-Signal nicht sofort den Hauptteil des Programms (Abschnitt **Event:**) startet.

Näheres zum Betriebsmodus des Betriebssystems bei einem extern gesteuerten Prozess siehe Abschnitt „Extern gesteuerter Prozess“ auf Seite 127.

### Siehe auch

End, Exit, ProcessN\_Running, Start\_Process, Stop\_Process

### Beispiel

```
Init:
    Rem Initialisierung
    ...
    Reset_Event           'Bisherige Event-Signale löschen

Event:
    Rem Hauptprogramm startet, sobald ein Event-Signal auftritt
    ...
```

## Restart\_Process

Nur für Prozessor T11: **Restart\_Process** startet den gleichen Prozess erneut.

### Syntax

**Restart\_Process**

### Bemerkungen

Die Anweisung ist nur im Abschnitt **Finish:** zulässig.

Alle Befehlszeilen des Abschnitts nach **Restart\_Process** werden noch ausgeführt, bevor der Prozess neu gestartet wird. Zur besseren Lesbarkeit empfehlen wir daher, die Anweisung ans Ende des Abschnitts zu stellen.



Die Anweisung kann zu einer Endlos-Schleife führen. Verwenden Sie **Restart\_Process** deshalb auf jeden Fall innerhalb einer Bedingung.

### Siehe auch

End, Exit, If ... Then ... {Else ... } EndIf, Start\_Process, Start\_Process\_Delayed, Stop\_Process

### Beispiel

**Event:**

...

**Finish:**

...

If (cond = 2) Then

*Rem Wenn Bedingung erfüllt, Prozess erneut starten*

**Restart\_Process**

EndIf

## 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

<code>var</code>	Auszuwertendes Argument (kein Berechnungsausdruck).	LONG
<code>const1a</code> , <code>const1b</code> , <code>const2a</code> , <code>const2b</code>	Wert von <code>var</code> (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

<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</code> < 0) und für ( <code>num</code> > 31)	LONG

### Bemerkungen

Verwenden Sie als Argument möglichst nur ganzzahlige Werte. Fließkomma-Werte (Typ `Float`) werden vor dem Schieben in ganzzahlige konvertiert. Die Nachkommastellen werden abgeschnitten und ggf. der Wertebereich angepasst, so dass das Berechnungsergebnis beeinflusst wird.

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

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

Verwenden Sie als Argument möglichst nur ganzzahlige Werte. Fließkomma-Werte (Typ **Float**) werden vor dem Schieben in ganzzahlige konvertiert: Die Nachkommastellen werden abgeschnitten und ggf. der Wertebereich angepasst, so dass das Berechnungsergebnis beeinflusst wird.

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
```



## Sin

**Sin** liefert den Sinus eines Arguments.

### Syntax

```
ret_val = Sin(angle)
```

### Parameter

**angle** Winkel im Bogenmaß ( $-\pi \dots \pi$ )

Float
-------

**ret\_val** Sinus des Winkels ( $-1 \dots 1$ )

Float
-------

### Bemerkungen

Wenn Sie für den Winkel Werte außerhalb von  $-\pi \dots \pi$  verwenden, nimmt der Berechnungsfehler mit wachsendem Wert zu.

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,25µs, beim T10 bis zu 0,63µs, beim T11 bis zu 0,28µs.

### Siehe auch

Cos, Tan, ArcSin, ArcCos, ArcTan

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = -5.3
```

```
val_2 = Sin(val_1) 'Ergebnis: val_2=0.83...
```

## Sleep

Nur bis Prozessor T10: **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

Bei dem Prozessor T11 muss **Sleep** ersetzt werden durch einen der Befehle **CPU\_Sleep**, **P1\_Sleep** oder **P2\_Sleep** (siehe auch Kapitel 5.2.4 „Wartezeit genau einstellen“).

Da der Befehl **Sleep** als Zählschleife ausgeführt wird, kann er bei einem hochprioren Prozess nicht unterbrochen werden.



Stellen Sie unbedingt (insbesondere bei Variablen) sicher, dass das Argument in keinem Fall einen Wert kleiner 1 hat, weil sonst das ADwin-System in einen instabilen Zustand geraten kann. Beachten Sie bitte auch, dass sehr große Werte bei hochprioren Prozessen zu einem Abbruch der Kommunikation führen können.

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.
- Das Argument ist ein Fließkomma-Wert.

### Siehe auch

CPU\_Sleep, NOP, P1\_Sleep, P2\_Sleep

### Beispiel

#### Event:

<b>Set_Mux</b> (0)	<i>'Multiplexer setzen</i>
<b>Sleep</b> (25)	<i>'2.5 <math>\mu</math>s (=25*100ns) warten</i>
	<i>'= Einschwingzeit des MUX</i>
<b>Start_Conv</b> (1)	<i>'Konvertierung starten</i>
...	

## Sqrt

**Sqrt** gibt die quadratische Wurzel eines Werts zurück.

### Syntax

```
ret_val = Sqrt(val)
```

### Parameter

**val**                      Argument

FLOAT
-------

**ret\_val**                Quadratwurzel des Arguments oder  
0 für (**val**<0)

FLOAT
-------

### Bemerkungen

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 0,9µs, beim T10 bis zu 0,45µs, beim T11 bis zu 0,26µs.

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 16
```

```
val_2 = Sqrt(val_1)     'Ergebnis: val_2 = 4
```

## Start\_Process

**Start\_Process** startet einen bestimmten Prozess.

### Syntax

**Start\_Process** (processnum)

### Parameter

**processnum** Nummer des zu startenden Prozesses  
(1...12, 15)

LONG

### Bemerkungen

Stellen Sie auf jeden Fall sicher, dass der zu startende Prozess bereits auf das ADwin-System übertragen wurde, bevor Sie ihn starten.



Die Anweisung hat keine Auswirkung, wenn Sie die Nummer eines Prozesses angeben, der

- bereits läuft oder
- gleich der Nummer des aufrufenden Prozesses ist.

Sie können einen Prozess mit **Start\_Process** nur aus einem anderen Prozess heraus starten (anders aber mit **Restart\_Process**). Es ist daher nicht möglich, dass ein Prozess sich selbst startet, beispielsweise im Abschnitt **Finish**.

### Siehe auch

End, Exit, Restart\_Process, Start\_Process\_Delayed, Stop\_Process

### Beispiel

**Event:**

```
If (ADC(1) > 3072) Then 'Schwellwert überschritten?
    Rem Wenn Prozess 2 nicht mehr aktiv ist: neu starten
    If (Process2_Running = 0) Then
        Start_Process(2) 'Messprozess 2 starten
    EndIf
End
EndIf
```

## Start\_Process\_Delayed

Nur für Prozessor T11: **Start\_Process\_Delayed** startet einen bestimmten Prozess (Abschnitt **Event:**) mit einer definierten Verzögerung.

### Syntax

**Start\_Process\_Delayed**(processnum, delay)

### Parameter

**processnum** Nummer (1...10) des zu startenden Prozesses.

**delay** Verzögerungszeit (>30) in Taktzyklen des Zählers.  
Beim T11 dauert 1 Taktzyklus 3,3 ns.

LONG

LONG

### Bemerkungen



Stellen Sie auf jeden Fall sicher, dass der zu startende Prozess bereits auf das ADwin-System übertragen wurde, bevor Sie ihn starten.

Die Anweisung kann nur einen zeitgesteuerten Prozess mit hoher Priorität starten; sie bleibt ohne Auswirkung, wenn Sie die Nummer eines Prozesses angeben, für den eine der folgenden Bedingungen gilt:

- Der Prozess wird vom externen Event-Signal gesteuert.
- Der Prozess hat niedrige Priorität.
- Der Prozess läuft bereits.
- Der Prozess hat die gleiche Nummer wie der aufrufende Prozess.

Sie können einen Prozess mit **Start\_Process\_Delayed** nur aus einem anderen Prozess heraus starten (anders mit **Restart\_Process**).

Der gestartete Prozess beginnt immer mit dem Abschnitt **Event:**, die Programmabschnitte **Init:** und **LowInit:** werden nicht ausgeführt.

Für den gewünschten Startzeitpunkt gilt:

- Die Verzögerung beginnt mit der Verarbeitung der Anweisung **Start\_Process\_Delayed**; die Verarbeitung der Anweisung dauert 30 Taktzyklen.
- Aus einem hochpriorigen Programmabschnitt heraus kann der Startzeitpunkt nur eingehalten werden, wenn die Verzögerungszeit **delay** größer ist als die restliche Bearbeitungszeit des Abschnitts.

In dem Programmabschnitt werden alle noch folgenden Zeilen verarbeitet, bevor der gewählte Prozess starten kann. Der Startzeitpunkt wird also durch eine zu lange Restbearbeitungszeit verzögert.

### Siehe auch

Restart\_Process, Start\_Process, Stop\_Process

### Beispiel

#### Event:

```
...
If (cond = 2) Then
    Rem Wenn Bedingung erfüllt, Prozess 2 um 100 Zyklen = 333 ns
    Rem verzögert starten.
    Start_Process_Delayed(2,100)
EndIf
Rem Es folgen keine weiteren Programmzeilen, damit der
Rem Startzeitpunkt sicher eingehalten wird.
```

## Stop\_Process

**Stop\_Process** stoppt aus einem Prozess heraus einen bestimmten anderen Prozess.

### Syntax

**Stop\_Process** (processnum)

### Parameter

**processnum** Nummer des zu stoppenden Prozesses (1...12, 15)

LONG
------

### Bemerkungen

Die Anweisung hat keine Auswirkung, wenn Sie die Nummer eines Prozesses angeben, der

- bereits gestoppt ist oder
- noch nicht auf das *ADwin*-System geladen ist

Das Unterbinden des Aufrufs von **Event**: läuft ab wie folgt:

- Der Prozess bekommt zunächst den Status "Prozess wird gestoppt" (siehe [ProcessN\\_Running](#)), bei niederprioren Prozessen erst nach einer gewissen Zeitspanne (time-out).
- Wenn der Abschnitt **Event**: gerade bearbeitet wird, während das Stopp-Signal eintrifft, wird diese Bearbeitung in jedem Fall zu Ende geführt.
- In der Regel wird der Abschnitt **Event**: noch ein weiteres Mal aufgerufen und bearbeitet.
- Falls vorhanden, wird der Abschnitt **Finish**: abgearbeitet (immer mit niedriger Priorität).
- Der Prozess ruht nun, kann aber jederzeit wieder gestartet werden.



Wenn der Prozess sich selbst stoppen soll, verwenden Sie den Befehl [End](#) oder [Exit](#).

### Siehe auch

[End](#), [Exit](#), [ProcessN\\_Running](#), [Restart\\_Process](#), [Start\\_Process](#), [Start\\_Process\\_Delayed](#)



### Beispiel

#### Event:

```
If (ADC(1) > 3072) Then 'Schwellwert überschritten?  
    Stop_Process(2)      'Messprozess 2 stoppen  
End  
EndIf
```

## "" String

Zeichenfolgen (Strings) werden in doppelten Hochkommas "" angegeben.

### Syntax

```
Import String.LI*           '*.LI9 für T9, *.LIA für T10,
                             '*.LIB für T11

Dim text[length] As String

text = "ADwin"
```

### Parameter

`text[]`                      Name der Text-Variablen

ARRAY

STRING

`length`                    Länge der Text-Variablen

CONST

LONG

### Bemerkungen

Dimensionieren Sie Text-Variablen mit `Dim ... As String` (siehe Seite 164). Setzen Sie eine Zeichenfolge, die sie der Variablen zuweisen möchten, in Hochkommas.

Näheres zu Textvariablen und dem Aufbau von Strings finden Sie unter „Strings“ auf Seite 93.

Sie können Zeichenfolgen mit den unten aufgeführten Anweisungen bearbeiten. Außerdem können Sie Zeichenfolgen mit dem „+“-Operator aneinander hängen.

### Siehe auch

+ (String-Addition), Dim, Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF, Vall

## Beispiel

```
Import String.LI9
```

*Rem Strings mit 3 und 1 Zeichen dimensionieren*

```
Dim chars[3] As String
```

```
Dim char[1] As String
```

### Init:

*Rem Den Strings eine Zeichenfolge übergeben*

```
chars = "ABC"
```

```
char = "z"
```

### Event:

```
Par_1 = chars[1]           'Par_1 = 3 Anzahl der Zeichen
```

```
Par_2 = chars[2]           'Par_2 = 65 (= "A")
```

```
Par_3 = chars[3]           'Par_3 = 66 (= "B")
```

```
Par_4 = chars[4]           'Par_4 = 67 (= "C")
```

```
Par_5 = chars[5]           'Par_5 = 0 String-Terminierung
```

*Rem Wandlung in Großbuchstaben:*

*Rem Kleinbuchstabe: a, b, c, ..., x, y, z?*

```
Par_6 = Asc(char)
```

```
If (Par_6>96 And Par_6<133) Then
```

*Rem 32 subtrahieren um in Großbuchstaben zu wandeln*

```
Chr(Par_6-32,char)
```

```
EndIf
```

## StrComp

**StrComp** überprüft zwei Zeichenketten auf Gleichheit.

### Syntax

```
Import String.LI*           '*.LI9 für T9, *.LIA für T10,
                             '*.LIB für T11

ret_val = StrComp(string1[], string2[])
```

### Parameter

string1[], Zeichenkette  
string2[]

ARRAY

STRING

CONST

ret\_val      0: Zeichenketten sind gleich  
             -1: Zeichenketten sind ungleich

LONG

### Bemerkungen

Wenn die Zeichenketten von unterschiedlicher Länge sind, wird immer ein negativer Wert zurückgegeben, auch wenn die kürzere Zeichenkette in der längeren enthalten ist.

### Siehe auch

"" String, + (String-Addition), Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrLeft, StrLen, StrMid, StrRight, ValF, Vall

### Beispiel

```
Import String.LI9

Dim text1[7], text2[7], text3[8] As String

Init:
text1 = "ADbasic"      'ADbasic richtig geschrieben
text2 = "ADbasci"      'ADbasic falsch geschrieben
text3 = "ADbasica"     'ADbasic falsch geschrieben

Event:
Par_1 = StrComp(text1, text2) 'Par_1=-1
Par_2 = StrComp(text1, text3) 'Par_2=-1
```

## StrLeft

**StrLeft** kopiert aus einer Zeichenkette von links her eine bestimmte Anzahl von Zeichen in eine zweite Zeichenkette.

### Syntax

```
Import String.LI*           '*.LI9 für T9, *.LIA für T10,  
                             '*.LIB für T11  
  
StrLeft(string1[], length, string2[])
```

### Parameter

`string1[]`      Zeichenkette, aus der kopiert wird

ARRAY

STRING

`length`          Anzahl der zu kopierenden Zeichen

LONG

`string2[]`      Zeichenkette, in die kopiert wird

ARRAY

STRING

### Siehe auch

"" String, + (String-Addition), Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLen, StrMid, StrRight, ValF, ValI

**Beispiel**

```
Import String.LI9

Rem Quell- und Ziel-String dimensionieren:
Dim text1[32], text2[13] As String

Init:
Rem Quell-String definieren
text1 = "MEGA-Echtzeit mit ADwin-Systemen"

Event:
Rem Hole 13 Zeichen von links aus dem String text1
StrLeft(text1,13,text2)
Par_1 = text2[1]      'String-Länge = 13 Zeichen
Par_2 = text2[2]      'ASCII-Zeichen 4Dh = "M"
Par_3 = text2[3]      'ASCII-Zeichen 45h = "E"
Par_4 = text2[4]      'ASCII-Zeichen 47h = "G"
Par_5 = text2[5]      'ASCII-Zeichen 41h = "A"
Par_6 = text2[6]      'ASCII-Zeichen 2Dh = "-"
Par_7 = text2[7]      'ASCII-Zeichen 45h = "E"
Par_8 = text2[8]      'ASCII-Zeichen 63h = "C"
Par_9 = text2[9]      'ASCII-Zeichen 68h = "h"
Par_10 = text2[10]    'ASCII-Zeichen 74h = "t"
Par_11 = text2[11]    'ASCII-Zeichen 7Ah = "z"
Par_12 = text2[12]    'ASCII-Zeichen 65h = "e"
Par_13 = text2[13]    'ASCII-Zeichen 69h = "i"
Par_14 = text2[14]    'ASCII-Zeichen 74h = "t"
Par_15 = text2[15]    'String-Ende-Zeichen = 0
```

## StrLen

**StrLen** gibt die Anzahl der Zeichen in einer Zeichenfolge zurück.

### Syntax

```
Import String.LI*           '*.LI9 für T9, *.LIA für T10,  
                             '*.LIB für T11  
  
ret_val = StrLen(string[])
```

### Parameter

**string[]**      Zeichenkette, deren Länge ermittelt wird

ARRAY

STRING

**ret\_val**      Anzahl der Zeichen in der Zeichenfolge

LONG

### Siehe auch

"" String, + (String-Addition), Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrMid, StrRight, ValF, Vall

### Beispiel

```
Import String.LI9  
Dim text[50] As String  
  
Init:  
    text = "MEGA-Echtzeit mit ADwin-Systemen"  
  
Event:  
    Par_1 = StrLen(text) 'String-Länge: Par_1 = 32
```

## StrMid

**StrMid** kopiert aus einer Zeichenkette ab einer anzugebenden Position eine bestimmte Anzahl von Zeichen in eine zweite Zeichenkette.

### Syntax

```
Import String.LI*      '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

StrMid(string1[], start, length, string2[])
```

### Parameter

<code>string1[]</code>	Zeichenkette, aus der kopiert wird	ARRAY
		STRING
<code>start</code>	Position des ersten Zeichens, das kopiert wird	LONG
<code>length</code>	Anzahl der zu kopierenden Zeichen	LONG
<code>string2[]</code>	Zeichenkette, in die kopiert wird	ARRAY
		STRING

### Siehe auch

"" String, + (String-Addition), Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrRight, ValF, ValI



**Beispiel**

```
Import String.LI9
```

```
Rem Quell- und Ziel-String dimensionieren:
```

```
Dim text1[32], text2[18] As String
```

**Init:**

```
Rem Quell-String definieren
```

```
text1 = "MEGA-Echtzeit mit ADwin-Systemen"
```

**Event:**

```
Rem Kopiere 18 Zeichen ab dem 6. Zeichen aus dem String text1
```

```
StrMid(text1,6,18,text2)
```

```
Par_1 = text2[1]           'String-Länge = 18 Zeichen
```

```
Par_2 = text2[2]           'ASCII-Zeichen 45h = "E"
```

```
Par_3 = text2[3]           'ASCII-Zeichen 63h = "c"
```

```
Par_4 = text2[4]           'ASCII-Zeichen 68h = "h"
```

```
Par_5 = text2[5]           'ASCII-Zeichen 74h = "t"
```

```
Par_6 = text2[6]           'ASCII-Zeichen 7Ah = "z"
```

```
Par_7 = text2[7]           'ASCII-Zeichen 65h = "e"
```

```
Par_8 = text2[8]           'ASCII-Zeichen 69h = "i"
```

```
Par_9 = text2[9]           'ASCII-Zeichen 74h = "t"
```

```
Par_10 = text2[10]          'ASCII-Zeichen 20h = " "
```

```
Par_11 = text2[11]          'ASCII-Zeichen 6Dh = "m"
```

```
Par_12 = text2[12]          'ASCII-Zeichen 69h = "i"
```

```
Par_13 = text2[13]          'ASCII-Zeichen 74h = "t"
```

```
Par_14 = text2[14]          'ASCII-Zeichen 20h = " "
```

```
Par_15 = text2[15]          'ASCII-Zeichen 41h = "A"
```

```
Par_16 = text2[16]          'ASCII-Zeichen 44h = "D"
```

```
Par_17 = text2[17]          'ASCII-Zeichen 77h = "w"
```

```
Par_18 = text2[18]          'ASCII-Zeichen 69h = "i"
```

```
Par_19 = text2[19]          'ASCII-Zeichen 6Eh = "n"
```

```
Par_20 = text2[20]          'String-Ende-Zeichen = 0
```

## StrRight

**StrRight** kopiert aus einer Zeichenkette von rechts her eine bestimmte Anzahl von Zeichen in eine zweite Zeichenkette.

### Syntax

```
Import String.LI*           '*.LI9 für T9, *.LIA für T10,  
                             '*.LIB für T11  
  
StrRight(string1[], length, string2[])
```

### Parameter

**string1[]**      Zeichenkette, aus der kopiert wird

**ARRAY**

STRING

**length**          Anzahl der zu kopierenden Zeichen

LONG

**string2[]**      Zeichenkette, in die kopiert wird

**ARRAY**

STRING

### Siehe auch

"" String, + (String-Addition), Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, ValF, ValI

**Beispiel**

```
Import String.LI9
```

```
Rem Quell- und Ziel-String dimensionieren:
```

```
Dim text1[32], text2[13] As String
```

**Init:**

```
Rem Quell-String definieren
```

```
text1 = "MEGA-Echtzeit und ADwin-Systeme"
```

**Event:**

```
Rem Hole 13 Zeichen von rechts aus dem String text1
```

```
StrRight(text1,13,text2)
```

```
Par_1 = text2[1]           'String-Länge = 13 Zeichen
```

```
Par_2 = text2[7]           'ASCII-Zeichen 41h = "A"
```

```
Par_3 = text2[8]           'ASCII-Zeichen 44h = "D"
```

```
Par_4 = text2[9]           'ASCII-Zeichen 77h = "w"
```

```
Par_5 = text2[10]          'ASCII-Zeichen 69h = "i"
```

```
Par_6 = text2[11]          'ASCII-Zeichen 6Eh = "n"
```

```
Par_7 = text2[12]          'ASCII-Zeichen 2Dh = "-"
```

```
Par_8 = text2[13]          'ASCII-Zeichen 53h = "S"
```

```
Par_9 = text2[14]          'ASCII-Zeichen 79h = "y"
```

```
Par_10 = text2[13]         'ASCII-Zeichen 73h = "s"
```

```
Par_11 = text2[13]         'ASCII-Zeichen 74h = "t"
```

```
Par_12 = text2[13]         'ASCII-Zeichen 65h = "e"
```

```
Par_13 = text2[14]         'ASCII-Zeichen 6Dh = "m"
```

```
Par_14 = text2[13]         'ASCII-Zeichen 65h = "e"
```

```
Par_15 = text2[15]         'String-Ende-Zeichen = 0
```

## 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 100.

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:/LowInit:.`
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 `LowInit:`, `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, Lib\_Sub ... Lib\_EndSub, Lib\_Function ... Lib\_EndFunction

## Beispiel

```
Sub Fast_Dac1(val_1)
  Rem Gibt val_1 auf dem analogen Ausgang 1 eines ADwin-Gold aus
  Poke(20400050h, (val_1)) 'Wert ins Ausgangsregister
  Poke(20400010h, 11011b) 'Wandlung starten
EndSub
```

Der Aufruf des Unterprogramms `Fast_Dac1` erfolgt mit der Programmzeile:

```
Fast_Dac1(NeuerWert)
```

Das gleiche Unterprogramm mit einem Feld als Übergabe-Parameter:

```
Sub Fast_Dac1(array[])  
    Rem Gibt Element 3 des Felds array auf dem  
    Rem analogen Ausgang eines ADwin-Gold aus  
    Poke(20400050h, (array[3])) 'Wert ins Ausgangsregister  
    Poke(20400010h, 11011b) 'Wandlung starten  
EndSub
```

Der Aufruf dieses Unterprogramms erfolgt wieder in ähnlicher Weise (allerdings *ohne* die Dimensionsklammern):

```
Fast_Dac1(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.

## Tan

**Tan** liefert den Tangens eines Arguments.

### Syntax

```
ret_val = Tan(angle)
```

### Parameter

**angle** Winkel im Bogenmaß ( $-\pi/2 \dots \pi/2$ )

Float
-------

**ret\_val** Tangens des Winkels ( $-1 \dots 1$ )

Float
-------

### Bemerkungen

Wenn Sie für den Winkel Werte außerhalb von  $-\pi/2 \dots \pi/2$  verwenden, nimmt der Berechnungsfehler mit wachsendem Wert zu.

Die Ausführungszeit des Befehls beträgt beim T9 bis zu 1,33µs, beim T10 bis zu 0,67µs, beim T11 bis zu 0,31µs.

### Siehe auch

Sin, Cos, ArcSin, ArcCos, ArcTan

### Beispiel

```
Dim val_1, val_2 As Float
```

#### Event:

```
val_1 = 5.3
```

```
val_2 = Tan(val_1) 'Ergebnis: val_2 = -1.50...
```

## Trace\_Mode\_Pause

**Trace\_Mode\_Pause** deaktiviert den Trace-Modus.

### Syntax

**Trace\_Mode\_Pause**

### Bemerkungen

Mit dem Befehl **Trace\_Mode\_Pause** kann in einem *ADbasic*-Programm der Trace-Modus gezielt deaktiviert werden. Mit dem Befehl **Trace\_Mode\_Resume** kann der Trace-Modus wieder aktiviert werden. Die Deaktivierung/Aktivierung betrifft nur Trace-aktive Programmzeilen, die mit ? (Fragezeichen) markiert wurden.

Beide Befehle gemeinsam erlauben, den Trace-Modus gezielt für bestimmte Programmzeilen oder -abschnitte einzuschalten. So kann der Trace-Modus z.B. nur solange aktiviert werden, wie eine bestimmte Bedingung erfüllt ist.

### Siehe auch

Trace\_Mode\_Resume

### Beispiel

```
Event:
  Par_1 = ADC(1,4)
  If (Par_1 > 32768) Then
    Trace_Mode_Resume    'Trace-Modus ein

    ...                  'Der Trace-Modus ist für diesen
    ...                  'Programmabschnitt durchgehend aktiv

    Trace_Mode_Pause     'Trace-Modus aus
  EndIf
```



## Trace\_Mode\_Resume

**Trace\_Mode\_Resume** aktiviert den Trace-Modus ab der nächsten Programmzeile.

### Syntax

**Trace\_Mode\_Resume**

### Bemerkungen

Mit dem Befehl **Trace\_Mode\_Resume** kann in einem *ADbasic*-Programm der Trace-Modus wieder aktiviert werden, wenn er zuvor mit **Trace\_Mode\_Pause** deaktiviert wurde. Die Deaktivierung/Aktivierung betrifft nur Trace-aktive Programmzeilen, die mit ? (Fragezeichen) markiert wurden.

Beide Befehle gemeinsam erlauben, den Trace-Modus gezielt für bestimmte Programmzeilen oder -abschnitte einzuschalten. So kann der Trace-Modus z.B. nur solange aktiviert werden, wie eine bestimmte Bedingung erfüllt ist.

### Siehe auch

Trace\_Mode\_Pause

### Beispiel

```
Event:
  Par_1 = ADC(1,4)
  If (Par_1 > 32768) Then
    Trace_Mode_Resume   'Trace-Modus ein

    ...                  'Der Trace-Modus ist für diesen
    ...                  'Programmabschnitt durchgehend aktiv

    Trace_Mode_Pause    'Trace-Modus aus
  EndIf
```

## ValF

**ValF** konvertiert eine Zeichenfolge (String) in eine Fließkomma-Zahl (float).

### Syntax

```
Import String.LI*          '*.LI9 für T9, *.LIA für T10,
                           '*.LIB für T11

ret_val = ValF(string[])
```

### Parameter

`string[]`      Zu wandelnde Zeichenfolge im Format:

ARRAY

STRING

Mantisse (max. 10 Zeichen)				Exponent (0...99)	
{+}	vvvvv	.	nnnnn	e	{+} nn
-		,		E	-

`ret_val`      Erzeugter Fließkomma-Wert

FLOAT

### Bemerkungen

Wenn Sie kein Vorzeichen angeben, wird ein positives Vorzeichen angenommen.

Das Zeichen "E" trennt Mantisse und Exponent. Von der Mantisse werden bei T9 und T10 bis zu 7 Zeichen (Vor- und Nachkommastellen) ausgewertet, bei T11 bis zu 10 Zeichen. Wenn Sie mehr Ziffern angeben, gehen die jeweils letzten verloren. Als Dezimalzeichen sind sowohl der Punkt als auch das Komma zulässig.

Beachten Sie den Wertebereich für Float-Werte in Kapitel 4.2.3 auf Seite 80. Werte außerhalb des Wertebereichs werden als „Unendlich“ oder Null interpretiert

Wenn Sie unzulässige Zeichen verwenden (= andere als im Format angegeben), wird die Zeichenfolge nur bis zum ersten unzulässigen Zeichen ausgewertet.

**Siehe auch**

"" String, Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, Vall

**Beispiel**

```
Import String.LI9

Dim text[20] As String

Init:
  text="-27.14159E-10"  'Zu konvertierender String
  Par_1 = text[1]      'String-Länge
  Par_2 = text[2]      'ASCII-Zeichen 2Dh = "-"
  Par_3 = text[3]      'ASCII-Zeichen 32h = "2"
  Par_4 = text[4]      'ASCII-Zeichen 37h = "7"
  Par_5 = text[5]      'ASCII-Zeichen 2Eh = "."
  Par_6 = text[6]      'ASCII-Zeichen 31h = "1"
  Par_7 = text[7]      'ASCII-Zeichen 34h = "4"
  Par_8 = text[8]      'ASCII-Zeichen 31h = "1"
  Par_9 = text[9]      'ASCII-Zeichen 35h = "5"
  Par_10 = text[10]     'ASCII-Zeichen 39h = "9"
  Par_11 = text[11]     'ASCII-Zeichen 45h = "E"
  Par_12 = text[12]     'ASCII-Zeichen 2Dh = "-"
  Par_13 = text[13]     'ASCII-Zeichen 31h = "1"
  Par_14 = text[14]     'ASCII-Zeichen 30h = "0"
  Par_15 = text[15]     'String-Ende-Zeichen

Event:
  FPar_1 = ValF(text)  'String zu Float wandeln
```

## Vall

**VallI** konvertiert eine Zeichenfolge (String) in eine ganze Zahl (Long).

### Syntax

```
Import String.LI*           '*.LI9 für T9, *.LIA für T10,
                             '*.LIB für T11

ret_val = Vall(string)
```

### Parameter

**string** []      Zu wandelnde Zeichenfolge im Format :  
 Vorzeichen: + (optional) oder –  
 Vorkommastellen: max. 10 Ziffern

ARRAY

STRING

---

{+}                    vvvvvvvvvvvv  
 –

---

**ret\_val**              Erzeugter Long-Wert

LONG

### Bemerkungen

Wenn Sie kein Vorzeichen angeben, wird ein positives Vorzeichen angenommen.

Beachten Sie den Wertebereich für Long-Werte:

-2.147.483.648 bis +2.147.483.647

Werte außerhalb dieses Bereichs werden als Null interpretiert.

Wenn Sie unzulässige Zeichen verwenden (= andere als im Format angegeben), wird nur die Zeichenfolge bis zum ersten unzulässigen Zeichen ausgewertet.

### Siehe auch

"" String, Asc, Chr, FloToStr, Flo40ToStr, LngToStr, StrComp, StrLeft, StrLen, StrMid, StrRight, ValF

**Beispiel**

```
Import String.LI9

Dim text[20] As String

Init:
    text="-1234567890"      'Zu konvertierender String
    Par_1 = text[1]         'String-Länge = 11
    Par_2 = text[2]         'ASCII-Zeichen 2Dh = "-"
    Par_3 = text[3]         'ASCII-Zeichen 31h = "1"
    Par_4 = text[4]         'ASCII-Zeichen 32h = "2"
    Par_5 = text[5]         'ASCII-Zeichen 33h = "3"
    Par_6 = text[6]         'ASCII-Zeichen 34h = "4"
    Par_7 = text[7]         'ASCII-Zeichen 35h = "5"
    Par_8 = text[8]         'ASCII-Zeichen 36h = "6"
    Par_9 = text[9]         'ASCII-Zeichen 37h = "7"
    Par_10 = text[10]        'ASCII-Zeichen 38h = "8"
    Par_11 = text[11]        'ASCII-Zeichen 39h = "9"
    Par_12 = text[12]        'ASCII-Zeichen 30h = "0"
    Par_13 = text[13]        'String-Ende-Zeichen

Event:
    Par_20 = ValI(text)     'String zu Long wandeln
```

## 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 FFT-Library

Die FFT-Library enthält *ADbasic*-Befehle für die schnelle Fourier-Transformation (Fast Fourier Transformation).

Die Library ist verfügbar für Prozessoren ab dem Typ T9.

#### Hinweise zur Anwendung der Library

Sie können die Rechenzeit deutlich verringern, wenn Sie die Felder im internen Speicher ([At DM\\_Local](#)) deklarieren. So dauert die Berechnung einer FFT mit 1024 Werten nur noch etwa 23ms anstatt 35ms (mit Prozessor T9).



Rufen Sie Anweisungen der FFT-Library nur in einem Programm mit niedriger Priorität auf oder im Programmabschnitt **LowInit:** oder **Init:**. Wenn die Berechnung einer FFT in einem hochprioritären Programm sehr lange dauert, nimmt der PC einen Fehler an und bricht die Kommunikation zum *ADwin*-System mit einer entsprechenden Fehlermeldung ab.



Im Ordner <C:\ADwin\ADbasic\lib\FFT\_doc+demo> finden Sie alle Beispiele zu den Library-Befehlen.

#### Fast-Fourier Transformationen

Die Fast-Fourier-Transformation (FFT) ist ein Algorithmus zur schnellen Berechnung einer diskreten Fouriertransformation. Die FFT ist in der Signalverarbeitung für viele Aufgaben geeignet, so. z. B. zur

- Berechnung digitaler Filter.
- Umrechnung eines Zeitverlaufs in der Schwingungsmesstechnik in eine Frequenzdarstellung.
- Berechnung von Spektrogrammen (Diagramme mit der Darstellung der Amplituden von den jeweiligen Frequenzanteilen).
- Näherungsweise Bestimmung der Frequenzen in einem abgetasteten Signal.

#### Befehle der Library

Name	Funktion
<b>FFT</b>	FFT führt eine komplexe Fast-Fourier-Transformation mit komplexen Eingangs- und Ausgangsdaten durch. 281
<b>FFT_MAG</b>	FFT_MAG gibt die Absolutbeträge komplexer Werte zurück. 285

Name	Funktion	
<b>FFT_SCALE</b>	FFT_Scale normiert das Ergebnis einer FFT-Berechnung auf die Größenordnung der einzelnen Signal- komponenten der Quelldaten.	283
<b>FFT_PHASE</b>	FFT_PHASE gibt die Phasenlagen komplexer Werte zurück.	287
<b>FFT_MAG_SCALE</b>	FFT_Mag_Scale gbt die skalierten Absolutbeträge komplexer Werte zurück.	289
<b>FFT_INIT</b>	FFT_Init initialisiert 2 Hilfsfelder für die Berechnung von Fast-Fourier-Transformationen.	290
<b>FFT_CALC</b>	FFT_Calc berechnet eine Fast-Fourier-Transforma- tion nach vorheriger Initialisierung.	291
<b>FFT_CALC_DM</b>	FFT_Calc_DM berechnet eine Fast-Fourier-Trans- formation nach vorheriger Initialisierung und ist für den Prozessor T10 optimiert.	293
<b>FFT_CALC_DX</b>	FFT_Calc_DX berechnet eine Fast-Fourier-Trans- formation nach vorheriger Initialisierung und ist für den Prozessor T10 optimiert.	295



## FFT

**FFT** führt eine komplexe Fast-Fourier-Transformation mit komplexen Eingangs- und Ausgangsdaten durch.

### Syntax

```
Import FFT.LI*           '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT(real[], img[], z_real[], z_img[], array1[],
    array2[], count)
```

### Parameter

<code>real[]</code>	Realteile der Quelldaten.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der Quelldaten.	<div>FLOAT</div> <div>ARRAY</div>
<code>z_real[]</code>	Ergebnis: Realteile (Index 1... <code>count</code> /2) der transformierten Daten. Feldgröße: $4 \times \text{count}$ .	<div>FLOAT</div> <div>ARRAY</div>
<code>z_img[]</code>	Ergebnis: Imaginärteile (Index 1... <code>count</code> /2) der transformierten Daten. Feldgröße: $4 \times \text{count}$ .	<div>FLOAT</div> <div>ARRAY</div>
<code>array1[]</code> , <code>array2[]</code>	Felder für interne Berechnungen. Feldgröße: $4 \times \text{count}$ .	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Punktzahl ( $\geq 2$ ) der Quelldaten. Die Anzahl der Punkte muss eine Potenz von 2 sein.	<div>LONG</div>

### Bemerkungen

Die Fourier-Transformation liefert korrekte Ergebnisse, wenn die Frequenzkomponenten  $f_i$  der Quelldaten innerhalb des folgenden Bereichs – bezogen auf die Abtastfrequenz  $f_{\text{Abtast}}$  – liegen:

$$0 \leq f_i \text{ und } f_i < f_{\text{Abtast}}/2$$

Die transformierten Daten, die komplexen Amplituden des Frequenzspektrums, befinden sich nur in den Elementen 1...`count`/2 der Felder `z_real` und `z_img`. Die restlichen Feldelemente (bis  $4 \times \text{count}$ )

werden für interne Berechnungen benötigt und enthalten Zwischenergebnisse.

Das Ergebnis der Transformation ist nicht normiert auf die Größenordnung der einzelnen Signalkomponenten der Quelldaten. Sie können die transformierten Daten mit der Anweisung **FFT\_Scale** normieren.

Die folgende Tabelle zeigt, wie das berechnete Frequenzspektrum den Elementen der Felder **z\_real** und **z\_img** zugeordnet ist (Normierung der Frequenzachse). Dabei ist  $t_{\text{total}}$  die gesamte Abtastzeit, in der die Quelldaten abgetastet wurden.

Im Beispiel unten ist die Abtastzeit  $t_{\text{total}} = 0,1\text{ s}$ ; dem Element [1024] ist daher die Frequenz  $(1024-1) / 0,1\text{ s} = 10230\text{ Hz}$  zugeordnet.

Elementindex	[1]	[2]	...	[i]	...	[count/2]
Frequenz [Hz]	0	$\frac{1}{t_{\text{total}}}$	...	$\frac{i-1}{t_{\text{total}}}$	...	$\frac{\text{count}/2-1}{t_{\text{total}}}$



Wenn Sie mehrere FFT mit der *gleichen* Anzahl an Quelldaten berechnen, kann die Berechnungszeit verringert werden: Rufen Sie anstelle von **FFT** zuerst die Anweisung **FFT\_Init** und anschließend mehrfach **FFT\_Calc** auf.

### Siehe auch

FFT\_Mag, FFT\_Scale, FFT\_Phase, FFT\_Mag\_Scale, FFT\_Init, FFT\_Calc, FFT\_Calc\_DM, FFT\_Calc\_DX

### Beispiel

Das Beispielprogramm (für *ADwin-Gold* und *ADwin-light-16*)

```
<C:\ADwin\ADbasic\lib\FFT_doc+demo\FFT_demo.bas>
```

liest das Analogsignal am Eingang 1 (2048 Messpunkte in 0,1s) und berechnet daraus eine FFT. Wenn beispielsweise ein Sinussignal von 1000Hz anliegt, werden die Maximalwerte in **Data\_3[101]** (Realteil) und **Data\_4[101]** (Imaginärteil) gespeichert.

## FFT\_Scale

**FFT\_Scale** normiert das Ergebnis einer FFT-Berechnung auf die Größenordnung der einzelnen Signalkomponenten der Quelldaten.

### Syntax

```
Import FFT.LI*           '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT_Scale(unscaled[], scaled[], count)
```

### Parameter

**unscaled[]** Nicht normierte Daten einer FFT-Berechnung.

Float

Array

**scaled[]** Ergebnis: Normierte Daten.

Float

Array

**ret\_val** Anzahl der Daten.

Long

### Bemerkungen

Die Anweisung arbeitet nach der Formel:

$$\text{scaled}[i] = \begin{cases} i \neq 1: \text{scaled}[i] = \text{unscaled}[i] / (\text{count}) \\ i = 1: \text{scaled}[i] = \text{unscaled}[i] / (\text{count} \cdot 2) \end{cases}$$

Wenn Sie **FFT\_Scale** direkt auf die Ergebnisfelder der Anweisung **FFT** anwenden, muss **count = count / 2** sein (**count**: Parameter von **FFT**).

**FFT\_Scale** normiert das Ergebnis einer FFT-Berechnung auf die Größenordnung der einzelnen Signalkomponenten der Originaldaten. Dagegen normiert **FFT\_Scale** nicht die Frequenzachse des Spektrums (siehe Erläuterungen hierzu unter **FFT**).

### Siehe auch

FFT, FFT\_Mag, FFT\_Phase, FFT\_Mag\_Scale

### Beispiel

Das Beispielprogramm (für alle ADwin-Systeme)

```
<C:\ADwin\ADbasic\lib\FFT_doc+demo\FFT_scale_demo.bas>
```

generiert ein Signal aus mehreren Sinussignalen, tastet das Signal ab, berechnet die FFT, den Absolutbetrag und normiert den Absolutbetrag.

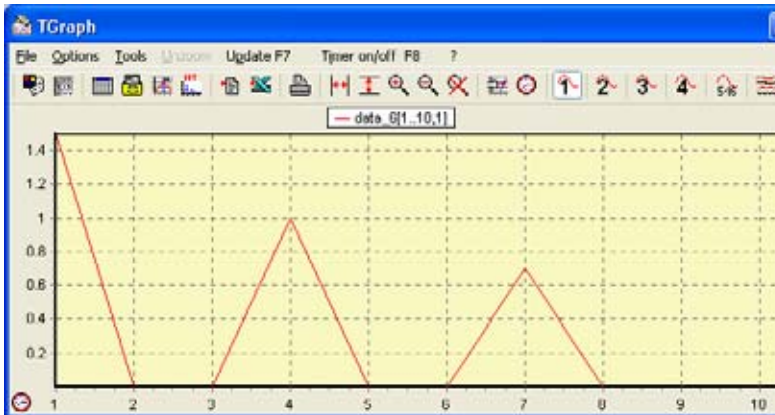
Das Quellsignal entsteht aus:

- einem Sinussignal mit 60 Hz und der Amplitude 0,7
- einem Sinussignal mit 30 Hz und der Amplitude 1,0
- einem konstanten Signal mit der Amplitude 1,5

Die Amplituden des normierten Frequenzspektrums (siehe Grafik unten, erzeugt mit Tgraph.exe) zeigen exakt die Größen der überlagerten Quellsignale:

```
Data_6[7] = 0.7      'Index 7: 60 Hz
Data_6[4] = 1        'Index 4: 30 Hz
Data_6[1] = 1.5      'Index 1: Gleichspannungsanteil
```

Alle weiteren Amplituden haben den Wert 0.



## FFT\_Mag

**FFT\_MAG** gibt die Absolutbeträge komplexer Werte zurück.

### Syntax

```
Import FFT.LI*          '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT_Mag(real[], img[], magnitude[], count)
```

### Parameter

<code>real[]</code>	Realteile der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>magnitude[]</code>	Ergebnis: Absolutbeträge der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Anzahl der komplexen Werte.	<div>LONG</div>

### Bemerkungen

Der Betrag eines komplexen Werts wird errechnet mit der Formel:

$$\text{magnitude}[i] = \sqrt{\text{real}[i]^2 + \text{img}[i]^2}$$

Die Anweisung **FFT** berechnet die Amplituden des Frequenzspektrums als komplexe Werte. Die Anweisungen **FFT\_Mag** und **FFT\_Phase** rechnen die komplexen Amplituden in Betrag und Phase um.

Wenn Sie **FFT\_Mag** direkt auf die Ergebnisfelder der Anweisung **FFT** anwenden, muss `count = count / 2` sein (`count`: Parameter von **FFT**).

### Siehe auch

FFT, FFT\_Phase, FFT\_Mag\_Scale

### Beispiel

Das Beispielprogramm (für *ADwin-Gold* oder *ADwin-light-16*)

```
<C:\ADwin\ADbasic\lib\FFT_doc+demo\FFT_mag_demo.bas>
```

liest das Analogsignal am Eingang 1 (2048 Messpunkte in 0,1s), berechnet daraus eine FFT und die Absolutbeträge. Wenn beispielsweise ein Sinussignal von 1500Hz anliegt, wird der maximale Betrag in `Data_5[151]` gespeichert.

## FFT\_Phase

**FFT\_PHASE** gibt die Phasenlagen komplexer Werte zurück.

### Syntax

```
Import FFT.LI*           '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT_Phase(real[], img[], phase[], count)
```

### Parameter

<code>real[]</code>	Realteile der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>phase[]</code>	Ergebnis: Phasenlagen der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Anzahl der komplexen Werte.	<div>LONG</div>

### Bemerkungen

Die Phasenberechnung arbeitet nach folgender Formel (Details siehe `<math.inc>`):

$$\text{phase}[i] = \begin{cases} \text{real}[i] \neq 0: & \text{phase}[i] = \text{atan}(\text{img}[i]/\text{real}[i]) \\ \text{real}[i] = 0: & \text{phase}[i] = \text{sgn}(\text{img}[i]) \cdot \pi/2 \end{cases}$$

Die Anweisung **FFT** berechnet die Amplituden des Frequenzspektrums als komplexe Werte. Die Anweisungen **FFT\_Mag** und **FFT\_Phase** rechnen die komplexen Amplituden in Betrag und Phase um.

Wenn Sie **FFT\_Phase** direkt auf die Ergebnisfelder der Anweisung **FFT** anwenden, muss `count = count / 2` sein (`count`: Parameter von **FFT**).

### Siehe auch

FFT, FFT\_Mag, FFT\_Mag\_Scale

**Beispiel**

Das Beispielprogramm (für alle ADwin-Systeme)

```
<C:\ADwin\ADbasic\lib\FFT_doc+demo\FFT_phase_demo.bas>
```

generiert 2 (um  $\pi/2$ ) phasenverschobene Sinussignale, tastet die Signale ab, berechnet daraus die FFT, die normierten Absolutbeträge und die Phasenlagen.

Das berechnete Frequenzspektrum hat folgende Werte:

```
Data_6[4] = 1 'Index 4: 30 Hz
```

```
Data_7[4] = -0.018410 'Phase etwa 0
```

```
Data_26[4] = 1 'Index 4: 30 Hz
```

```
Data_27[4] = 1.552389 'Phase etwa  $\pi/2$ 
```

Alle weiteren Amplituden haben den Wert 0 und die zugehörigen Phasenlagen sind undefiniert.



## FFT\_Mag\_Scale

**FFT\_Mag\_Scale** gibt die skalierten Absolutbeträge komplexer Werte zurück.

### Syntax

```
Import FFT.LI*          '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT_Mag_Scale(real[], img[], mag_scal[], count)
```

### Parameter

<code>real[]</code>	Realteile der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>mag_scal[]</code>	Ergebnis: Normierte Absolutbeträge der komplexen Werte.	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Anzahl der komplexen Werte.	<div>LONG</div>

### Bemerkungen

Die Anweisung **FFT\_Mag\_Scale** liefert das gleiche Ergebnis wie der Aufruf von **FFT\_Mag** und **FFT\_Scale**, arbeitet aber schneller.

Wenn Sie **FFT\_Mag\_Scale** direkt auf die Ergebnisfelder der Anweisung **FFT** anwenden, muss `count = count / 2` sein.

### Siehe auch

FFT, FFT\_Mag, FFT\_Scale

### Beispiel

Das Beispielprogramm <FFT\_scale\_demo\_opt.bas> (für alle ADwin-Systeme) entspricht dem Beispiel <FFT\_scale\_demo.bas> (siehe Seite 283FFT\_Scale), verwendet jedoch **FFT\_Mag\_Scale**.

## FFT\_Init

**FFT\_Init** initialisiert 2 Hilfsfelder für die Berechnung von Fast-Fourier-Transformationen.

### Syntax

```
Import FFT.LI*          '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT_Init(array1[], array2[], count)
```

### Parameter

<code>array1[]</code> ,	Ergebnis: Hilfswerte für interne Berechnungen. Feld-	FLOAT
<code>array2[]</code>	größe: $4 \times \text{count}$ .	ARRAY
<code>count</code>	Punktzahl ( $\geq 2$ ) der Quelldaten. Die Anzahl der Punkte muss eine Potenz von 2 sein.	LONG

### Bemerkungen

Die Anweisung **FFT\_Init** ist nur notwendig und sinnvoll, wenn im Anschluss eine der Anweisungen **FFT\_Calc**, **FFT\_Calc\_DM** oder **FFT\_Calc\_DX** aufgerufen wird.



Wenn Sie mehrere FFT mit der *gleichen* Anzahl `count` an Quelldaten berechnen, kann die Berechnungszeit verringert werden: Rufen Sie anstelle von **FFT** zuerst die Anweisung **FFT\_Init** und anschließend mehrfach **FFT\_Calc** auf.

### Siehe auch

FFT, FFT\_Calc, FFT\_Calc\_DM, FFT\_Calc\_DX

### Beispiel

Siehe Beispielprogramm <FFT\_scale\_demo\_opt.bas> (für alle ADwin-Systeme) im Ordner <C:\ADwin\ADbasic\lib\FFT\_doc+demo>.

## FFT\_Calc

**FFT\_Calc** berechnet eine Fast-Fourier-Transformation nach vorheriger Initialisierung.

### Syntax

```
Import FFT.LI*           '*.LI9 für T9, *.LIA für T10,
                        '*.LIB für T11

FFT_Calc(real[], img[], z_real[], z_img[],
          array1[], array2[], count)
```

### Parameter

<code>real[]</code>	Realteile der Quelldaten.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der Quelldaten.	<div>FLOAT</div> <div>ARRAY</div>
<code>z_real[]</code>	Ergebnis: Realteile (Index 1... <code>count</code> /2) der transformierten Daten. Feldgröße: $4 \times \text{count}$ .	<div>FLOAT</div> <div>ARRAY</div>
<code>z_img[]</code>	Ergebnis: Imaginärteile (Index 1... <code>count</code> /2) der transformierten Daten. Feldgröße: $4 \times \text{count}$ .	<div>FLOAT</div> <div>ARRAY</div>
<code>array1[]</code> , <code>array2[]</code>	Felder für interne Berechnungen. Feldgröße: $4 \times \text{count}$ .	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Punktzahl ( $\geq 2$ ) der Quelldaten. Die Anzahl der Punkte muss eine Potenz von 2 sein.	<div>LONG</div>

### Bemerkungen

Die Anweisung ist nur sinnvoll, wenn vorher **FFT\_Init** aufgerufen wurde.

Wenn Sie mehrere FFT mit der *gleichen* Anzahl `count` an Quelldaten berechnen, kann die Berechnungszeit verringert werden: Rufen Sie anstelle von **FFT** zuerst die Anweisung **FFT\_Init** und anschließend mehrfach **FFT\_Calc** auf.



Nur Prozessor T10: Anstelle von **FFT\_Calc** können die Funktionen **FFT\_Calc\_DM** oder **FFT\_Calc\_DX** genutzt werden, die die FFT in kürzerer Zeit berechnen.

**Siehe auch**

FFT, FFT\_Init, FFT\_Calc\_DM, FFT\_Calc\_DX

**Beispiel**

Siehe Beispielprogramm <FFT\_scale\_demo\_opt.bas> (für alle ADwin-Systeme) im Ordner <C:\ADwin\ADbasic\lib\FFT\_doc+demo>.

## FFT\_Calc\_DM

**FFT\_Calc\_DM** berechnet eine Fast-Fourier-Transformation nach vorheriger Initialisierung und ist für den Prozessor T10 optimiert.

### Syntax

```
Import FFT.LIA

FFT_Calc_DM(real[], img[], z_real[], z_img[],
             arr1[], arr2[], count)
```

### Parameter

<code>real[]</code>	Realteile der Quelldaten. Das Feld muss <code>At DM_Local</code> deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der Quelldaten. Das Feld muss <code>At DM_Local</code> deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>z_real[]</code>	Ergebnis: Realteile (Index 1... <code>count</code> /2) der transformierten Daten. Das Feld muss <code>At DM_Local</code> mit der Größe: $4 \times \text{count}$ deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>z_img[]</code>	Ergebnis: Imaginärteile (Index 1... <code>count</code> /2) der transformierten Daten. Das Feld muss <code>At DM_Local</code> mit der Größe: $4 \times \text{count}$ deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>array1[]</code> , <code>array2[]</code>	Felder für interne Berechnungen. Die Felder müssen <code>At DM_Local</code> mit der Größe: $4 \times \text{count}$ deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Punktzahl ( $\geq 2$ ) der Quelldaten. Die Anzahl der Punkte muss eine Potenz von 2 sein.	<div>LONG</div>

### Bemerkungen

Die Anweisung ist nur sinnvoll, wenn vorher **FFT\_Init** aufgerufen wurde.

**FFT\_Calc\_DM** hat die gleiche Funktion wie **FFT\_Calc**, berechnet eine FFT aber etwas schneller. Der Befehl **FFT\_Calc\_DM** kann nur auf dem Prozessor T10 sinnvoll eingesetzt werden.

**FFT\_Calc\_DM** darf nur verwendet werden, wenn alle übergebenen Felder im internen Speicher deklariert sind.

Mit dem Prozessor T10 dauert die Berechnung einer FFT mit 1024 Werten nur noch etwa 11ms anstatt 14ms mit **FFT\_Calc**. Für beide Zeitmessungen wurden die Felder im internen Speicher **DM\_Local** deklariert.

**Siehe auch**

FFT, FFT\_Init, FFT\_Calc, FFT\_Calc\_DX

**Beispiel**

Siehe Beispielprogramm `<FFT_scale_demo_opt.bas>` (für alle ADwin-Systeme) im Ordner `<C:\ADwin\ADbasic\lib\FFT_doc+demo>`.

## FFT\_Calc\_DX

**FFT\_Calc\_DX** berechnet eine Fast-Fourier-Transformation nach vorheriger Initialisierung und ist für den Prozessor T10 optimiert.

### Syntax

```
Import FFT.LIA

FFT_Calc_DX(real[], img[], z_real[], z_img[],
             array1[], array2[], count)
```

### Parameter

<code>real[]</code>	Realteile der Quelldaten. Das Feld sollte <code>At DRAM_Extern</code> deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>img[]</code>	Imaginärteile der Quelldaten. Das Feld sollte <code>At DRAM_Extern</code> deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>z_real[]</code>	Ergebnis: Realteile (Index 1... <code>count</code> /2) der transformierten Daten. Das Feld sollte <code>At DRAM_Extern</code> mit der Größe <code>4 × count</code> deklariert sein.	<div>FLOAT</div> <div>ARRAY</div>
<code>z_img[]</code>	Ergebnis: Imaginärteile (Index 1... <code>count</code> /2) der transformierten Daten. Das Feld sollte <code>At DRAM_Extern</code> mit der Größe <code>4 × count</code> deklariert sein.	<div>FLOAT</div> <div>ARRAY</div>
<code>array1[]</code> , <code>array2[]</code>	Felder für interne Berechnungen. Die Felder müssen <code>At DRAM_Extern</code> mit der Größe: <code>4 × count</code> deklariert werden.	<div>FLOAT</div> <div>ARRAY</div>
<code>count</code>	Punktzahl ( $\geq 2$ ) der Quelldaten. Die Anzahl der Punkte muss eine Potenz von 2 sein.	<div>LONG</div>

### Bemerkungen

Die Anweisung ist nur sinnvoll, wenn vorher **FFT\_Init** aufgerufen wurde.

**FFT\_Calc\_DX** hat die gleiche Funktion wie **FFT\_Calc**, berechnet eine FFT aber etwas schneller. Der Befehl **FFT\_Calc\_DX** kann nur auf dem Prozessor T10 sinnvoll eingesetzt werden.

**FFT\_Calc\_DX** darf nur verwendet werden, wenn alle übergebenen Felder im externen Speicher deklariert sind.

Mit dem Prozessor T10 dauert die Berechnung einer FFT mit 1024 Werten nur noch etwa 49ms anstatt 53ms mit **FFT\_Calc**. Für beide Zeitmessungen wurden die Felder im externen Speicher **DRAM\_Extern** deklariert.

**Siehe auch**

FFT, FFT\_Init, FFT\_Calc, FFT\_Calc\_DM

**Beispiel**

Siehe Beispielprogramm <FFT\_scale\_demo\_opt\_DX.bas> (für alle ADwin-Systeme) im Ordner <C:\ADwin\ADbasic\lib\FFT\_doc+demo>.



## 7.4 Mathematik-Befehle

Die Include-Datei `math.inc` enthält zusätzliche Mathematik-Befehle, die über den Befehlssatz des *ADbasic*-Compilers hinaus gehen.

Die Befehle sind verfügbar für Prozessoren ab dem Typ T9.

### Mathematik-Befehle

Name	Funktion
<b>Mod</b>	Mod berechnet den ganzzahligen Rest bei einer Division von zwei ganzen Zahlen. 298

## Mod

**Mod** berechnet den ganzzahligen Rest bei einer Division von zwei ganzen Zahlen.

### Syntax

```
#Include Math.inc
ret_val = Mod(x_param, y_param)
```

### Parameter

x_param	Dividend.	LONG
y_param	Divisor.	LONG
ret_val	Rest aus der Division von x_param / y_param.	LONG

### Bemerkungen

Die Division mit Rest arbeitet nach der symmetrischen Variante, bei der der Quotient durch Abschneiden der Nachkommastellen berechnet wird. Mit dieser Definition wird der Quotient zur Null hin gerundet und der Rest hat das gleiche Vorzeichen wie der Dividend.

Der ganzzahlige Rest bei einer Division durch Null ist gleich dem Dividend: **Mod**(x, 0) = x.

Die Ausführungszeit für die Modulo-Funktion beträgt beim T9 bis zu 3,5µs, beim T10 bis zu 1,67µs und beim T11 bis zu 0,44µs (hohe Priorität).

### Siehe auch

/ (Division), AbsI

### Beispiel

```
Par_1 = Mod(17, 3)      'Par_1 = 2
Par_2 = Mod(-9, 5)      'Par_2 = -4
Par_3 = Mod(72, Par_2)  'Par_3 = 3
```

### 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 ADbasic

Um die Tastaturkürzel für Textbausteine zu sehen, öffnen Sie die Datei <ADbasicCS.xml> in C:\ADwin\ADbasic\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-System booten.	
F6	Bibliothek erzeugen.	Build ► Make Lib File
F7	Binärdatei erzeugen.	Build ► Make Bin File
CTRL-F7	Alle Binärdateien des Projekts erzeugen.	Build ► Make All Bin Files
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.	
CTRL-A	Alles markieren.	Edit ► Select All

Tastenkürzel	Funktion	Menüaufruf
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

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
00h 0	01h 1	02h 2	03h 3	04h 4	05h 5	06h 6	07h 7
BS <sup>1</sup>	TAB <sup>2</sup>	LF <sup>3</sup>	VT	FF	CR <sup>4</sup>	SO	SI
08h 8	09h 9	0Ah 10	0Bh 11	0Ch 12	0Dh 13	0Eh 14	0Fh 15
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
10h 16	11h 17	12h 18	13h 19	14h 20	15h 21	16h 22	17h 23
CAN	EM	SUB	ESC	FS	GS	RS	US
18h 24	19h 25	1Ah 26	1Bh 27	1Ch 28	1Dh 29	1Eh 30	1Fh 31
SPC <sup>5</sup>	!	"	#	\$	%	&	'
20h 32	21h 33	22h 34	23h 35	24h 36	25h 37	26h 38	27h 39
(	)	*	+	,	-	.	/
28h 40	29h 41	2Ah 42	2Bh 43	2Ch 44	2Dh 45	2Eh 46	2Fh 47
0	1	2	3	4	5	6	7
30h 48	31h 49	32h 50	33h 51	34h 52	35h 53	36h 54	37h 55
8	9	:	;	<	=	>	?
38h 56	39h 57	3Ah 58	3Bh 59	3Ch 60	3Dh 61	3Eh 62	3Fh 63
@	A	B	C	D	E	F	G
40h 64	41h 65	42h 66	43h 67	44h 68	45h 69	46h 70	47h 71
H	I	J	K	L	M	N	O
48h 72	49h 73	4Ah 74	4Bh 75	4Ch 76	4Dh 77	4Eh 78	4Fh 79
P	Q	R	S	T	U	V	W
50h 80	51h 81	52h 82	53h 83	54h 84	55h 85	56h 86	57h 87
X	Y	Z	[	\	]	^	_
58h 88	59h 89	5Ah 90	5Bh 91	5Ch 92	5Dh 93	5Eh 94	5Fh 95
`	a	b	c	d	e	f	g
60h 96	61h 97	62h 98	63h 99	64h 100	65h 101	66h 102	67h 103
h	i	j	k	l	m	n	o
68h 104	69h 105	6Ah 106	6Bh 107	6Ch 108	6Dh 109	6Eh 110	6Fh 111
p	q	r	s	t	u	v	w
70h 112	71h 113	72h 114	73h 115	74h 116	75h 117	76h 118	77h 119
x	y	z	{		}	~	□
78h 120	79h 121	7Ah 122	7Bh 123	7Ch 124	7Dh 125	7Eh 126	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 *ADbasic* – 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 *ADbasic* (im folgenden als „*ADbasic*-Software“ bezeichnet) und die gedruckte Benutzerdokumentation mit der Bezeichnung „*ADbasic*: 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 *ADbasic*-Software nur auf einem einzelnen Computer und nur an einem Ort benutzt werden darf. Der Lizenznehmer darf die *ADbasic*-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 *ADbasic*-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 *ADbasic*-Software oder das zugehörige schriftliche Material an einen Dritten zu übergeben oder einem Dritten sonstwie zugänglich zu machen,

- b) die *ADbasic*-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 *ADbasic*-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 *ADbasic*-Software aufgezeichnet ist. Ein Erwerb von Rechten an der *ADbasic*-Software selbst ist damit nicht verbunden.
- 4.2 Die Jäger Messtechnik GmbH behält sich insbesondere alle Veröffentlichungs-, Vervielfältigungs-, Bearbeitungs- und Verwertungsrechte an der *ADbasic*-Software vor.

#### 5. VERVIELFÄLTIGUNG

- 5.1 Die *ADbasic*-Software und das zugehörige Schriftmaterial sind urheberrechtlich geschützt.

Zu Sicherungszwecken ist dem Lizenznehmer das Anfertigen einer einzigen Reservekopie der *ADbasic*-Software erlaubt. Er ist verpflichtet, auf der Reservekopie den Urheberrechtsvermerk der Jäger Messtechnik GmbH anzubringen. Der in der *ADbasic*-Software vorhandene Urheberrechtsvermerk darf nicht entfernt werden.

- 5.2 Es wird ausdrücklich untersagt, die *ADbasic*-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 *ADbasic*-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 *ADbasic*-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 *ADbasic*-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 *ADbasic*-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 *ADbasic*-Software nach eigenem Ermessen zu erstellen. Die Jäger Messtechnik GmbH ist nicht verpflichtet, Aktualisierungen der *ADbasic*-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 *ADbasic*-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 *ADbasic*-Software. Insbesondere übernimmt die Jäger Messtechnik GmbH keine Gewähr dafür, dass die *ADbasic*-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 *ADbasic*-Software, sowie der damit beabsichtigten oder erzielten Ergebnisse trägt der Lizenznehmer. Das gleiche gilt für das die *ADbasic*-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 *ADbasic*-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 *ADbasic*-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 *ADbasic* Ihren License key bereits eingegeben haben. 

Der Kommandozeilen-Aufruf hat sich im Vergleich zu *ADbasic 4* geändert, d. h. Sie müssen die Syntax vorhandener Aufrufe überprüfen. 

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>.

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**

```

ADbasicCompiler /M src.bas
[/A"dest"] [/IP"path"] [/LP"path"] [/Lx] [/Sx] [/Px]
[/ET | /EE] [/PNx] [/PH | /PL | /PLx] [/PDx] [/Ox]
[/Vx]

ADbasicCompiler /L src.bas
[/A"dest"] [/IP"path"] /LP"path" [/Lx] [/Sx] [/Px]
[/Ox]

ADbasicCompiler /MAKE"makefile"

ADbasicCompiler /H

ADbasicCompiler /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 .Txn erzeugen.
x	Prozessortyp; siehe Schalter /Px.
n	Prozessnummer; siehe Schalter /PNx.
/L	Library-Datei (Bibliothek) mit der Endung .Lix erzeugen.
x	Prozessortyp; siehe Schalter /Px.
/MAKE	Hauptschalter, Dateiname und weitere Schalter für einen Aufruf aus dem makefile entnehmen.  Der Text im makefile kann über mehrere Zeilen verteilt geschrieben werden. Schalter außerhalb des makefile 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. 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>.Txn</code> (Binärdatei) oder <code>.LIX</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>ADbasic</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>ADbasic</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>/SC</code> Karten <code>/SL</code> Light-16 <code>/SG</code> Gold; Default. <code>/SGII</code> Gold II <code>/SP</code> Pro <code>/SPII</code> Pro II
<code>/Px</code>	Prozessortyp, für den die Datei kompiliert wird: <code>/P2</code> Prozessor T2 <code>/P4</code> Prozessor T4 <code>/P5</code> Prozessor T5 <code>/P8</code> Prozessor T8 <code>/P9</code> Prozessor T9; Default <code>/P10</code> Prozessor T10 <code>/P11</code> Prozessor T11

/ET	Zeitgesteuerten Prozess erzeugen, siehe Kapitel 6 auf Seite 117. Default. Schließt /EE aus.
/EE	Extern gesteuerten Prozess erzeugen, siehe Kapitel 6 auf Seite 117. Schließt /ET aus.
/PNx	Nummer x (1...10) des Prozesses. Default: 1.
/PH	Prozess mit hoher Priorität erzeugen. Default-Einstellung. Siehe auch Kapitel 6.1.2 auf Seite 119.
/PL	Prozess mit niedriger Priorität und Prioritätsstufe 1 erzeugen (nur für zeitgesteuerten Prozess). Siehe auch Kapitel 6.1.3 auf Seite 119.
/PLx	Prozess mit niedriger Priorität und Prioritätsstufe x (-10...10) erzeugen.
/PDx	Zykluszeit (Processdelay) des Prozesses auf x einstellen. Default: 1000, für T11: 3000. Siehe auch Kapitel 6.2.1 auf Seite 122.
/Ox	Optimierungsstufe x (0, 1, 2) für die Kompilierung einstellen, siehe auch Dialogfenster „Process Options“ (Seite 46). /O0 Optimierungsstufe 0 (=keine Optimierung) /O1 Optimierungsstufe 1 (Default) /O2 Optimierungsstufe 2
/Vx	Version x des Prozesses einstellen, siehe Dialogfenster „Process Options“ (Seite 46). 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 *ADbasic* im Infofenster (siehe Kapitel 3.9.1) 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\ADbasic\ADbasiccompiler.exe /L
Z:\Myfiles\test.bas
```



Diese Kommandozeile kompiliert den Quelltext `<test.bas>` und erzeugt die Library-Datei `<test.LI9>` 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-Gold*.
- Prozessor: T9.
- Optimierungsstufe 1.

Wenn Sie sich beim Aufruf im Verzeichnis `<C:\ADwin\ADbasic>` befinden, können Sie obige Zeile kürzer schreiben:

```
ADbasiccompiler.exe /L Z:\Myfiles\test.bas
```

Die kürzeste Aufruf-Variante ergibt sich, wenn auch der Quelltext im Verzeichnis `<C:\ADwin\ADbasic>` liegt:

```
ADbasiccompiler /L test.bas
```

Wir empfehlen in jedem Fall – speziell für eine Automatisierung des Aufrufs – die vollständige Variante:

```
ADbasiccompiler /L test.bas /A"test" /LE /SG /P9 /O1
```

```
ADbasicCompiler /L Z:\Myfiles\string.bas /SP /O1
```



Der Aufruf kompiliert den Quelltext `<string.bas>` mit Optimierungs-Level 1 in eine Library-Datei für ein *Pro*-System mit Prozessor T9. Der Prozess ist zeitgesteuert, hat die Nummer 1 und hohe Priorität.

Der gleiche Aufruf, nur für den Prozessor T10, sieht so aus:

```
ADbasicCompiler /L Z:\Myfiles\string.bas /P10 /SP /O1
```



```
ADbasicCompiler /M  
C:\ADwin\ADbasic\samples_ADwin\bas_dmo6f.bas /LE /SG /P9  
/ET /PN3 /PH /O1
```

Kompiliert die Demo-Datei <bas\_dmo6f.bas> in eine Binär-Datei für ein Gold-System mit Prozessor T9; der Prozess ist zeitgesteuert, hat die Nummer 3 und hohe Priorität.



```
ADbasicCompiler /M  
C:\ADwin\ADbasic\samples_ADwin\bas_dmo6 /LE /SL /P8 /EE  
/PN2 /PH /O0
```

Kompiliert die Demo-Datei <bas\_dmo6.bas> in eine Binär-Datei für eine Light-16-Karte mit dem Prozessor T8, Optimierung ist deaktiviert; der Prozess ist zeitgesteuert, hat die Nummer 2 und niedrige Priorität.



```
ADbasicCompiler /M C:\user\my_file.bas /LE /SC /P4 /EE  
/PN5 /PL /A"your_file" /O1
```

Die Anweisung kompiliert die Datei <my\_file.bas> für eine ADwin-Karte mit Prozessor T4; der Prozess ist extern gesteuert, hat die Nummer 5 und niedrige Priorität. Die erzeugte Binärdatei hat den Namen <your\_file.T45> und befindet sich im gleichen Verzeichnis wie der Quelltext: <C:\user>.



```
ADbasicCompiler /M C:\user\my_file.bas /LE /SG /P9  
/A"Y:\somewhere\your_file" /ET /PN3 /PH /O1
```

Die Binärdatei heißt nun <your\_file.T93> 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 Obsolete Programmteile

Die Entwicklungsumgebung enthält aus Gründen der Kompatibilität auch Einstellungsmöglichkeiten für *ADwin*-Systeme mit Transputer-Prozessoren (T4, T5, T8).

### Dialogfenster „Process Options“

In diesem Dialogfenster legen Sie Compiler-Optionen für das gerade aktive Quelltextfenster fest, d.h. Sie bestimmen Eigenschaften desjenigen Prozesses, der aus dem aktiven Quelltext übersetzt und ins *ADwin*-System übertragen wird.

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).

Wenn Sie im Dialogfenster „Compiler Options“ den Prozessortyp T4, T5 oder T8 eingestellt haben, wird das unten gezeigte Dialogfenster geöffnet.

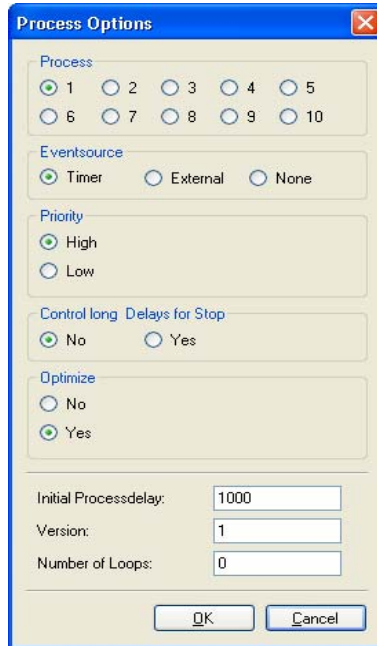


Abb. 1 – Dialogfenster „Process Options“ für Prozessoren T4...T8

- „Event“: Sie stellen hier ein, welches Event-Signal den Abschnitt **EVENT**: Ihres Prozesses starten soll.

Mit der Einstellung „Timer“ definieren Sie Impulse des internen Zählers als Event-Signal. In diesem Fall legen Sie mit der Systemvariablen **PROCESSDELAY** fest, in welchen Zeitabständen der Zähler ein Event-Signal auslöst.

Mit „Extern“ legen Sie fest, dass ein Signal am Event-Eingang Ihrer ADwin-Hardware den Prozess startet. Dies könnte beispielsweise ein Impuls eines Messaufnehmers sein. Ein solcher Prozess muss mit hoher Priorität ablaufen. Stellen Sie für diesen Fall die Option „Priority“ auf „High“.

Wie Sie bei einem ADwin-Pro-System einen externen Event-Eingang nutzen können, lesen Sie bitte in der zugehörigen Software-Dokumentation unter dem Befehl **EVENTENABLE** nach.


Mit der Einstellung „None“ startet der Prozess nach der Übertragung ins System sofort. Der Abschnitt **EVENT**: wird, unabhängig von Event-Signalen, nach der Ausführung erneut gestartet (Endlos-Schleife). Insbesondere bei einem hochprioritären Prozess müssen Sie dann dafür sorgen, dass der Prozess auch Rechenzeit für andere Aufgaben (z. B. Kommunikation mit dem PC) bereitstellt.

- „Process“: Stellen Sie die Nummer (1...10) ein, unter der der übertragene Prozess auf dem System angesprochen wird.

Wenn Sie mehrere Prozesse gleichzeitig auf einem ADwin-System ablaufen lassen, müssen Sie jedem Prozess eine eigene Nummer zuweisen.

- „Number of Loops“: Falls gewünscht, können Sie hier die Anzahl der Event-Durchläufe des Prozesses einstellen. Ist diese Anzahl erreicht, stoppt der Prozess automatisch. Eine geänderte Einstellung wird beim nächsten Start des Prozesses wirksam (also nicht bei einem laufenden Prozess), ohne dass Sie das Programm neu kompilieren müssen.

Wenn Sie den Wert „0“ eintragen, wird das Programm solange wiederholt, bis Sie den Prozess stoppen mit:

- dem Befehl **END**,
  - dem Befehl **STOP\_PROCESS** oder
  - der Schaltfläche  in der Entwicklungsumgebung.
- „Version“: Hier können Sie einen ganzzahligen Wert eingeben, um verschiedene Versionen Ihres Programms zu unterscheiden.
  - „Priority“: Stellen Sie hier die Priorität des Prozesses ein, mit der er im System laufen soll. Weitere Informationen zu diesem Thema finden Sie in Kapitel 6.1 „Prozessverwaltung“. Der Eintrag „Level“ existiert für diesen Prozessortyp nicht.
  - „Control long Delays for Stop“: Die Einstellung ist nur bei den Prozessoren T2...T8 verfügbar.  
  
Das Stoppen eines Prozesses tritt verzögert ein, wenn er nur selten (Zykluszeit > 5 Millisekunden) aufgerufen wird. Wir empfehlen Ihnen in diesem Fall, die Option zu aktivieren, denn sie beschleunigt den Stoppvorgang.
  - „Optimize“: Die wahlweise einschaltbare Optimierung kann die Prozess-Ausführungszeit um bis zu 20% verkürzen. 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 Einstellung eines niedrigeren „Level“ beheben.

- „Delay“: Stellen Sie hier das Processdelay (Zykluszeit) ein, mit dem der Prozess beginnen soll.

## A.6 Liste der Debug-Fehlermeldungen

Die folgenden Fehlermeldungen können angezeigt werden, wenn in *ADbasic* die Option *Debug mode* aktiv ist; siehe Option *Debug mode*, Seite 53.

Fehlermeldung
Division durch Null
Wurzel aus negativer Zahl
Data n: Index ist zu groß / Data n: Index ist kleiner als 1 Array index ist zu groß / Array index ist kleiner als 1 Zugriff auf nicht deklarierte Elemente eines lokalen oder globalen Felds, d.h. auf eine zu große oder zu kleine Elementnummer. Der Zusatz (inc) in der Fehlermeldung ist ein ergänzender Hinweis für unseren Support, an welcher Stelle der Fehler festgestellt wurde.
FIFO-Index ist kein FIFO Das Feld mit dieser Nummer ist nicht als FIFO oder gar nicht deklariert.
Adresse des Pro II Moduls ist >15 oder <1
P2_Burst_xxx <sup>1</sup> : "startadr" ist nicht durch 4 teilbar
P2_Burst_xxx <sup>1</sup> : Anzahl der Werte nicht durch 4 teilbar
P2_Burst_Init <sup>1</sup> : Anzahl der Werte nicht durch 4 / 8 teilbar
P2_Burst_Read_Unpacked1: Anzahl der Werte ist nicht durch 8 teilbar
P2_Burst_Read_Unpacked2: Anzahl der Werte ist nicht durch 4 teilbar
P2_Burst_Read_Unpacked8: Anzahl der Werte ist nicht durch 2 teilbar
P2_Burst_Read: Anzahl der Werte ist kleiner als 1 / als 4
P2_GetData/SetData_Long: TiCo DATA existiert nicht
P2_GetData/SetData_Long: TiCo DATA hat falschen Datentyp

Fehlermeldung
P2_GetData/SetData_Long: TiCo DATA-Index zu groß
P2_GetData/SetData_Long: TiCo DATA-Index < 1
P2_Digout_FIFO_Write: Zeitstempel-Differenz < 2
Media_Read / Media_Write: start_block + count_blocks128 > num_blocks start_block < 0 Zugriff auf ungültige Bereiche des Speichermediums, d.h. auf eine zu große oder zu kleine Blocknummer.

---

1. Gilt für **P2\_BURST\_INIT**, **P2\_BURST\_READ**, **P2\_BURST\_WRITE**

## A.7 Index

FFT · 281  
Mod · 298

## Symbole

- · 136  
# · 141  
#Define · 162  
#Else · 192  
#EndIf · 192  
#If · 192  
#Include · 197  
\* · 137  
+ · 133  
+ (String) · 134  
.NET · 130  
/ · 138  
: · 142  
< = > · 144  
= · 143  
^ · 139  
' (Rem) · 240

## Zahlen

150h, siehe Device No.  
2-dimensionale Felder · 89  
40 Bit-Genauigkeit · 81

## A

Abbruch siehe Prozess beenden  
AbsF · 145  
AbsI · 146  
Absolutwert  
    Fließkomma-Zahlen · 145  
    Ganze Zahlen · 146  
ActiveX · 130  
    aus Entwicklungsumgebung  
        nutzen · 130  
    Kommunikation zum ADwin-  
        System · 129

ADbasic  
    Demo-Modus · 10  
    Lizenzvertrag · 5  
    starten · 9  
    Verzeichnisse einstellen · 52  
ADbasic 4  
    speichern als · 7  
ADbasic 4:Umstellungen · 5  
ADbasicCompiler · 9  
Add Open Files to Project · 58  
Add to Project · 18, 58  
Addition · 133  
ADtools  
    Leiste einstellen · 53  
ADtools · 73  
ADWIN\_CARD · 192  
ADWIN\_GOLD · 192  
ADWIN\_GOLDII · 192  
ADWIN\_L16 · 192  
ADWIN\_PRO · 192  
ADWIN\_PROII · 192  
ADWIN\_SYSTEM · 192  
ADwin32.dll · 128  
aktivieren, Trace-Modus · 273  
Analyse  
    Allgemein · 112  
    Laufzeitfehler · 113  
    Zeitverhalten · 113  
And · 147  
Anhalten siehe Prozess beenden  
Anmerkungen siehe Kommentar  
Anzahl von Prozessen · 114  
Anzeige  
    aktuelle Informationen · 14  
    Auslastung: CPU, PM, EM, DM,  
        DX · 63  
    Befehlsparameter · 36  
    Prozessoptionen · 13  
    ToDo-Liste · 66  
Arcus-Cosinus: ArcCos · 149  
Arcus-Sinus: ArcSin · 150



Arcus-Tangens: ArcTan · 151  
 Arithmetische Funktionen  
   - · 136  
   \* · 137  
   + · 133  
   / · 138  
   ^ · 139  
   Dec · 161  
   Exp · 172  
   Inc · 196  
   LN · 212  
   Log · 215  
   Sqrt · 252  
 Array-Index (lokal) zu groß / < 1, siehe Laufzeitfehler, erkennen  
 Arrays, siehe Felder  
 (Dim) As · 164  
 Asc · 152  
 ASCII-Zeichensatz · 3  
 (Dim ...) At · 164  
 Auslastung  
   100%, wegen  
     Speicherfragmentierung · 120  
   Anzeige · 63  
   Definition · 125  
   Einfluss der Prozess-  
     Anzahl · 114  
 auswerten  
   Operatoren · 97  
 Autoindent · 49  
 Automatisch vervollständigen, Befehle und Variablen · 34  
 Automatische  
   Typkonvertierung · 98  
 Automatisches Einrücken · 22  
 Automatisches Formatieren · 21  
 AutoSave · 42  
 Autostart · 42

## B

Backslash im String, Escape-Sequenz · 96  
 Basis e · 172  
 Bearbeitungszeit messen · 105  
 Bedienoberfläche · 12  
 Bedingter Sprung  
   If ... Then · 190  
   SelectCase · 243  
 Befehl  
   Automatisch  
     vervollständigen · 34  
   Deklaration anzeigen · 37  
   Übergabeparameter  
     anzeigen · 36  
     zur Deklaration springen · 33  
 Befehlsreferenz · 131  
 Befehls-Separator (:) · 142  
 Befehlszeile  
   Groß-/Kleinschreibung · 75  
   max. Zeilenlänge · 75  
   Zeilenlänge  
     mit #Include · 197  
 benutzerdefinierte Befehle und Variablen  
   Übersicht · 77  
 Berechnungsausdrücke · 97  
   auswerten · 97  
   getrennte Auswertung · 99  
   symbolische Namen · 77  
   Typkonvertierung · 98  
 Betriebssystem  
   laden, siehe Booten  
   Standardverzeichnis · 11  
   Verzeichnis einstellen · 52

Bibliothek  
  Allgemeines · 101  
  Ein-/ausfalten · 22  
  Einbinden · 194  
  Erzeugen  
    aus ADbasic · 43  
    aus Kommandozeile · 9  
  erzeugen · 42  
  Funktion · 203  
  Position im Programm · 78  
  Rekursion verboten · 102  
  Unterprogramm · 208  
  Verzeichnis einstellen · 52  
Binärdatei  
  Erzeugen  
    aus Kommandozeile · 9  
  erzeugen · 42  
    aus ADbasic · 43  
  nutzen in  
    Entwicklungsumgebung · 130  
  siehe auch Bibliothek  
Binäre Schreibweise · 82  
Bits verschieben  
  nach links · 246  
  nach rechts · 247  
Bookmark · 32  
Booten · 11  
Bootloader · 117  
BTL-Datei: Verzeichnis  
  einstellen · 52  
Busy-Anzeige · 63

## C

C#.NET, C++ · 130  
Carriage Return im String, Escape-  
  Sequenz · 96  
Case, CCase, CaseElse (Select-  
  Case ...) · 243  
Cast\_FloatToLong · 153  
Cast\_LongToFloat · 154  
Chr · 155

Clear Parameter Scan · 38  
Code size · 65  
code snippets · 36  
Comment Block · 22  
Compiler  
  Aufruf · 42  
  AutoSave · 42  
  Compilermeldung, Fehler /  
    Status · 65  
  Fehlermeldung · 42  
  Kommandozeilen-Aufruf · 9  
  Optionen einstellen · 43  
Compiler-Anweisungen · 141  
  #Define · 162  
  #if ... Then · 192  
  #include · 197  
Control block · 18  
Controlblock · 32  
Cosinus: Cos · 156  
CPU\_Sleep · 157  
Cursor-Position · 63

## D

DATA\_n  
  dimensionieren · 164  
  Übersicht · 159  
Data\_n  
  Globale Felder · 83  
  Globale Felder, 2-  
    dimensional · 89  
Data-Index (global) zu groß / <1,  
  siehe Laufzeitfehler, erkennen  
Dateiname  
  Bibliothek · 43  
  Binärdatei · 43  
Datenaustausch  
  mit dem PC · 128  
  mit der  
    Entwicklungsumgebung · 130  
  zwischen Prozessen · 128

- Datenspeicher
  - siehe auch Speicher
  - 2-Dim. Felder im ~ · 89
  - Übersicht, intern, extern · 88
  - Zusatzbedarf durch
    - Debug-Modus · 113
    - Timing-Modus · 116
- Datenspeicher, intern (DM) · 88
- Datenstrukturen
  - FIFO · 90
  - Globale Felder · 83
    - Speicherfragmentierung · 120
  - Globale Felder, 2-dimensional · 89
  - Globale Variablen · 82
  - Lokale Variablen und Felder · 86
  - Übersicht · 79
- Datentypen
  - String · 93
  - Typkonvertierung · 38
  - Übersicht · 80
- Datenverlust
  - beim Booten · 11
  - FIFO · 92
- Datenwort: Nummerierung von Bits · 2
- deaktivieren
  - Trace-Modus · 272
- Debug
  - Allgemein · 112
  - Debug-Modus anwenden · 113
  - Menü · 53
  - Timing-Fenster · 67
  - Timing-Modus aktivieren · 53
  - Timing-Modus anwenden · 113
  - Trace\_Mode\_Pause · 272
  - Trace\_Mode\_Resume · 273
- Debug errors · 53
- Debug mode · 53
- Dec · 161
- Declaration Info · 37
- Declarations · 72
- DEFINE siehe #Define
- Dekadischer Logarithmus · 215
- Deklaration
  - alle anzeigen · 37, 72
  - anzeigen · 37
  - siehe Dimensionierung
  - zur ~ springen · 33
- Dekrementieren · 161
- Delphi · 130
- Demo-Modus · 10
- Device No.
  - Definition · 129
  - einstellen · 44
- Dezimale Schreibweise · 82
- Dezimaltrennzeichen · 82
- Dim · 164
- Dimensionierung
  - Befehl Dim · 164
  - Position im Programm · 78
  - Speicherbereich · 87
- Directory siehe Verzeichnis
- Disable Trace · 18
- Division
  - durch 2 · 247
  - einfache · 138
  - ganzzahliger Rest · 298
- Division durch Null, siehe Laufzeitfehler, erkennen
- DM, siehe Datenspeicher, intern
- DM\_Local
  - Dim · 164
- Do ... Until · 167
- DRAM\_Extern
  - Dim · 164
  - Event · 169
  - Finish · 179
  - Init · 199
  - LowInit · 216
- Druckeinstellungen · 51
- DX, siehe Externer Speicher

**E**

- Editor
  - General · 49
  - Print Settings · 51
  - Syntax Colors · 50
- Editor-Leiste · 20
- e-Funktion: Exp · 172
- Ein-/ausfalten, Textbereich · 22
- Einbinden
  - Include-Datei · 197
  - Library · 194
- Einrücken · 22
- Else (If ...) · 190
- EM, siehe Zusatzspeicher
- EM\_Local
  - Dim · 164
  - Event · 169
  - Finish · 179
  - Init · 199
  - LowInit · 216
- Enable Trace · 18
- End · 168
- EndFunction · 187
- EndIf (If ...) · 190
- EndSelect (SelectCase ...) · 243
- EndSub · 268
- Entwicklungsumgebung
  - Leisten und Fenster · 12
  - starten · 9
  - Tastaturkürzel · 1
- Ersetzen
  - Beispiele · 29
  - Reguläre Ausdrücke · 29
  - Text · 24
- Escape-Sequenz · 95
- Event
  - externes Signal · 117
  - externes Signal: löschen · 241
  - Signalquelle einstellen · 47
  - verlorenes Signal
    - ein Prozess

- zeitgesteuert · 126
- extern gesteuerter
  - Prozess · 127
- mehrere Prozesse
  - zeitgesteuert · 127
- prüfen · 69
- Zeitdifferenz messen · 106
- Event: · 169
- Exit · 171
- Exklusiv-ODER-Verknüpfung · 278
- Exponential-Funktion Exp · 172
- Exponential-Schreibweise · 82
- Externer Speicher (DX) · 88
- externer Speicher (SDRAM) · 88
- Externer Speicher (SX) · 88
- externes Event-Signal · 117
- Extremwert
  - Maximum Fließkomma-Zahlen · 218
  - Maximum Ganze Zahlen · 220
  - Minimum Fließkomma-Zahlen · 219
  - Minimum Ganze Zahlen · 221

**F**

- F1: Hilfe aufrufen · 17
- Falten, Textbereiche · 22
- Farbe einstellen · 50
- Farbige Darstellung · 21
- Fehler
  - durch Cut&Paste erzeugt · 41
  - geringere Optimierung
    - einstellen · 47
- Fehlermeldung
  - Laufzeitfehler · 53
  - Time out · 120
- Fehlermeldung Compiler · 65

- Felder
  - 2-dimensional · 89
  - DATA\_n · 159
  - FIFO · 173
  - globale · 83
    - erstes Element · 84
    - verwendete anzeigen · 38
  - initialisieren · 78
  - kopieren · 222
  - lokale · 86
    - erstes Element · 86
  - Speicherbereich festlegen · 87
  - Übersicht · 79
- Fenster
  - Compiler Options · 43
  - Debug errors · 53
  - Declarations · 72
  - Global Variables · 70
  - Info-Bereich · 64
  - Info-Fenster · 65
  - Parameter · 60
  - Process Options · 46
  - Projekt · 58
  - Quelltext-Informationen · 13
  - Quelltext-Statusleiste · 13
  - Statusleiste · 63
  - Timing Analyzer · 67
  - ToDo-Liste · 66
  - Toolbox · 58
  - Übersicht · 12
- FFT · 281
- FFT\_Calc · 291
- FFT\_Calc\_DM · 293
- FFT\_Calc\_DX · 295
- FFT\_Init · 290
- FFT\_Mag · 285
- FFT\_Mag\_Scale · 289
- FFT\_Phase · 287
- FFT\_Scale · 283
- FIFO
  - Aufbau der Datenstruktur · 90
  - belegte Elemente abfragen · 178
  - Datenverlust · 92
  - dimensionieren · 164
  - Elementanzahl prüfen · 92
  - freie Elemente abfragen · 177
  - initialisieren · 175
  - Übersicht · 173
- FIFO\_Clear · 175
- FIFO\_Empty · 177
- FIFO\_Full · 178
- Finden · 24
  - Beispiele · 28
  - Deklaration von Befehl/Variable · 33
  - Reguläre Ausdrücke · 29
- Finish: · 179
- Fließkomma-Zahlen
  - Dezimale Schreibweise · 82
  - Exponential-Schreibweise · 82
  - Wertebereich · 81
- Flo40ToStr · 183
- Float, siehe Fließkomma-Zahlen
- FloToStr · 181
- Font einstellen · 50
- For ... Next · 185
- formatieren · 21
- FPar\_n, globale Variablen · 82
- Fragmentierung, Speicher · 120
- Function · 187
- Funktion
  - Allgemeines zu Bibliotheken · 101
  - Allgemeines zu Makros · 100
  - Bibliothek (Lib\_Function) · 203
  - Makro · 187
  - Position im Programm · 78

**G**

- Ganze Zahlen
  - Binäre Schreibweise · 82
  - Hexadezimale Schreibweise · 82
  - Typkonvertierung · 98
  - Wertebereich · 81
- ganzzahliger Rest bei Division · 298
- Gerätenummer, Definition · 129
- gleich = · 144
- Global Variables · 70
- Globaldelay · 234
- globale Felder, siehe Felder, globale
- globale Variablen, siehe Variablen, globale
- Goto Line · 33
- Groß-/Kleinschreibung · 16
- größer als >, >= · 144

**H**

- Halt siehe Prozess beenden
- Hardware-Zugriff
  - Lesen · 232
  - Schreiben · 233
- Header · 51
- Hexadezimale Schreibweise · 82
- Hilfe
  - aufrufen · 17
- Hilfsprogramme (*ADtools*<>) · 73

**I**

- IEEE-Floating point-Format · 81
- If · 190
  - siehe auch #If · 192
- Import · 194
- Inc · 196
- Include-Datei
  - Allgemeines · 101
  - einbinden · 197
  - Verzeichnis einstellen · 52

- Indent · 22
- Indent ADbasic sections · 49
- Info-Bereich · 64
- Info-Fenster · 65
- Init: · 199
- Initialisierung · 77
  - Booten · 11
  - umfangreiche · 77
- Inkrementieren · 196
- Installation,
  - Standardverzeichnis · 11
- INTEGER · 81
- Integer
  - Typkonvertierung · 98
- Interner Speicher
  - Datenspeicher (DM) · 88
  - Gesamt (SRAM) · 88
  - Programm (PM) · 88
  - Zusatzspeicher (EM) · 88
- IO\_Sleep · 201

**J**

- Java · 130
- Jump to Declaration · 33

**K**

- kleiner als <, <= · 144
- Kommandozeilen-Aufruf · 9
- Kommentar
  - Syntax · 240
  - Zeilen in ~ ändern · 22
- Kommunikation
  - mit dem PC · 128
  - mit der
    - Entwicklungsumgebung · 130
    - Prozess im ADwin-System · 120
    - Time out · 120
    - zwischen Prozessen · 128
  - kompilieren siehe Compiler
- Konstante · 77

Kontextmenü · 18  
Projektfenster · 58  
Kontrollstrukturen · 100

## L

Language · 52  
latency (Timing-Fenster) · 67  
Laufzeitfehler  
    Anzeige · 53  
    erkennen · 113  
    siehe auch Debug-Modus  
length (Timing-Fenster) · 67  
Lib\_EndFunction · 203  
Lib\_EndSub · 208  
Lib\_Function · 203  
Lib\_Sub · 208  
Library  
    Funktion · 203  
    Import · 194  
    siehe auch Bibliothek  
    siehe Bibliothek  
    Unterprogramm · 208  
Lib-Verzeichnis einstellen · 52  
License key eingeben · 10  
Line Feed im String, Escape-  
    Sequenz · 96  
Lizenzvertrag · 5  
LN · 212  
LngToStr · 213  
Log · 215  
Logarithmus  
    dekadischer · 215  
    natürlicher · 212  
Logische Funktionen  
    And · 147  
    Not · 225  
    Or · 226  
    Shift\_Left · 246  
    Shift\_Right · 247  
    XOr · 278

Long  
    Typkonvertierung · 98  
Long, siehe Ganze Zahlen  
LowInit · 216

## M

Make Bin File, Make Lib File · 42  
Makro  
    Allgemeines · 100  
    Ein-/ausfalten · 22  
    Funktion · 187  
    Position im Programm · 78  
Mark Controlblock · 32  
Matlab · 130  
Matrix, 2-dimensional · 89  
Max\_Float · 218  
Max\_Long · 220  
Maximale Zeilenlänge  
    mit #Include · 197  
Maximum  
    Fließkomma-Zahlen · 218  
    Ganze Zahlen · 220  
MemCpy · 222  
Menü  
    auswählen · 13  
    Build · 42  
    Debug · 53  
    Edit · 41  
    File · 40  
    Help · 57  
    Leiste · 39  
    Options · 43  
    Tools · 56  
    View · 41  
    Window · 57  
Menüleiste · 39  
Messwertverlauf darstellen · 73  
Metazeichen · 29  
Min\_Float · 219  
Min\_Long · 221

Minimum  
Fließkomma-Zahlen · 219  
Ganze Zahlen · 221  
Mod · 298  
Multiplikation  
einfache · 137  
mit 2 · 246

## N

Nachkommastellen  
abschneiden · 98  
Namen, lokale Variablen · 86  
Natürlicher Logarithmus · 212  
negatives Vorzeichen · 98  
Neues in ADbasic 5 · 5  
Next (For ...) · 185  
NICHT · 225  
niederpriore Prozesse beim  
T11 · 124  
NOP · 224  
Not · 225

## O

ODER-Verknüpfung · 226  
Operatoren  
And · 147  
auswerten · 97  
negatives Vorzeichen · 98  
Or · 226  
Priorität · 97  
XOr · 278  
Optimales Zeitverhalten  
ein Prozess · 115  
mehrere Prozesse · 114

Optimierung  
Allgemein · 105  
Bearbeitungszeit messen · 105  
Konstanten statt Variablen · 107  
Polynom schneller  
berechnen · 140  
Registerzugriff · 106  
schneller messen · 107  
T11 Speicherzugriff · 111  
Wartezeit einstellen · 108  
Wartezeit nutzen · 110  
Optionen einstellen  
ADtools · 53  
Allgemein (Settings) · 49  
Compiler · 43  
Drucken · 51  
Editor · 49  
Prozess · 46  
Sprache · 52  
strukturierte  
Befehlsdarstellung · 50  
Verzeichnisse · 52  
Or · 226  
Ordner siehe Verzeichnis  
Outdent · 22

## P

P1\_Sleep · 228  
Par\_n, globale Variablen · 82  
Parameter Scan · 38  
Parameter, siehe Variablen, globale  
Parameterfenster · 60  
Parse and Indent · 49  
Peek · 232  
PM, siehe Programmspeicher  
PM\_Local  
Event · 169  
Finish · 179  
Init · 199  
LowInit · 216  
Poke · 233



- Polynom, schneller berechnen · 140
- Potenz · 139
  - in Polynom ersetzen · 140
  - zur Basis e · 172
- Präprozessor-Anweisungen · 141
- Präprozessor-Anweisungen
  - #Define · 162
  - #If ... Then · 192
  - #Include · 197
- Print layout · 51
- Priorität
  - niederpriorie Prozesse mit T11 · 124
  - Operatoren · 97
  - Prozess, siehe Prozess, Priorität von Prozessen prüfen · 114
- Probleme
  - langsamer Editor · 49
- Process\_Error · 237
- Process\_Running · 238
- Processdelay
  - Syntax · 234
  - Zeitverhalten · 122
- ProcessN\_Running · 238
- Processor · 192
- Programm verbessern, siehe Optimierung
- Programmabschnitte
  - Event: · 77
  - Finish: · 77
  - Init: · 77
  - LowInit: · 77
  - Übersicht · 77
- Programmspeicher
  - Zusatzbedarf durch
    - Debug-Modus · 113
    - Timing-Modus · 116
- Programmspeicher (PM) · 88
- Programmstruktur
  - Übersicht · 100
- Bibliothek
  - Lib\_Function · 203
  - Lib\_Sub · 208
  - Übersicht · 101
- Ein-/ausfalten · 22
- Include-Datei · 101
- Kommentar Rem · 240
- Makros
  - Funktion Function · 187
  - Übersicht · 100
  - Unterprogramm Sub · 268
- Schleife
  - Do ... Until · 167
  - For ... Next · 185
- Sprung
  - If ... Then · 190
  - SelectCase · 243
- Projekt
  - Alle Quelltextdateien übersetzen · 43
- Projektfenster · 58
- Projektverwaltung
  - Allgemeines · 38
  - Fenster · 58
  - verwendete Variablen anzeigen · 38
- Prozess
  - Anzahl · 118
  - Bearbeitungszeit · 123
  - beenden
    - andere · 256
    - sich selbst (in Event:) · 168
    - sich selbst (in Init:, LowInit:,

- Finish:) · 171
- Betriebszustände beim
  - Zeitverhalten · 126
- Fehler auslesen · 237
- Kommunikation zwischen
  - ~en · 128
- Kommunikationsprozess · 120
- mehrere · 123
- neu laden · 120
- Optimales Zeitverhalten
  - ein Prozess · 115
  - mehrere Prozesse · 114
- Optionen
  - einstellen · 46
- Optionen, Anzeige · 13
- Priorität
  - Hoch · 119
  - Kommunikation · 120
  - Niedrig · 119
  - Niedrig mit T11 · 124
  - Übersicht · 118
- prüfen auf Anzahl, Priorität · 114
- Speichernutzung · 120
- Standardprozesse 11, 12 · 119
- starten
  - anderen Prozess · 253
  - gleichen Prozess erneut · 242
  - verzögert · 254
- Status ermitteln · 238
- Zeitverhalten · 122
- Zyklus, siehe Prozesszyklus
- Prozess optimieren, siehe Optimierung
- Prozessor, siehe Processor · 192
- Prozess-Steuerung
  - End · 168
  - Exit · 171
  - Process\_Error · 237
  - ProcessN\_Running · 238
  - Reset\_Event · 241
  - Restart\_Process · 242
  - Start\_Process · 253
  - Start\_Process\_Delayed · 254
  - Stop\_Process · 256
- Prozesszyklus
  - Aufruf
    - mit Event-Signal · 117
    - Zeiteinheit · 122
    - zeitlich exakter Aufruf · 123
  - Punkt vor Strich, siehe Operatoren
- Q**
  - Quadratwurzel · 252
  - Quelltext
    - erstellen · 16
    - farbig darstellen · 21
    - formatieren · 21
    - im Projekt verwenden · 58
    - ToDo-Liste · 66
  - Quelltext-Statusleiste · 13
- R**
  - Read\_Timer · 239
  - Rechenzeit sparen
    - Konstanten statt Variablen · 107
    - Registerzugriff · 106
    - schneller messen · 107
    - Wartezeit einstellen · 108
    - Wartezeit nutzen · 110
  - Registerzugriff · 106
  - Reguläre Ausdrücke · 29
  - Rekursion bei Bibliothek · 102
  - Rem · 240
  - Reset\_Event · 241

Rest, ganzzahliger bei  
Division · 298  
Restart\_Process · 242  
Ringbuffer · 90  
Ringspeicher · 90

## S

Save All Files of Project · 58  
(Bits) schieben  
nach links · 246  
nach rechts · 247  
Schleife, Do ... Until · 167  
Schreibweise von Zahlen · 82  
Schriftart einstellen · 50  
SDRAM, siehe Externer Speicher  
SelectCase · 243  
Separator : · 142  
Shift\_Left · 246  
Shift\_Right · 247  
SHORT · 81  
Short-Cuts · 1  
Show Declarations · 37  
Show line numbers · 49  
Sinus: Sin · 249  
P2\_Sleep · 230  
Sleep · 250  
Smart format · 21  
snippets · 36

Speicher  
Auslastung · 63  
Bedarf bestimmen · 65  
Bereich festlegen · 87  
Bereiche (PM, DM, EM, DX) · 88  
Fragmentierung · 120  
siehe auch Datenspeicher  
String · 94  
Zusatzbedarf durch  
Bibliotheken · 102  
Debug-Modus · 113  
Makros · 101  
Timing-Modus · 116  
Speichern  
für ADbasic 4 · 7  
springen zu Zeile · 33  
Sprung, bedingter  
If ... Then · 190  
SelectCase · 243  
Sqrt · 252  
SRAM, siehe Interner Speicher, Gesamt  
Stack size · 65  
Start\_Process · 253  
Start\_Process\_Delayed · 254  
Starten, ADbasic · 9  
Statusleiste · 63  
Statusmeldung Compiler · 65  
Step (For ...) · 185  
Steuerzeichen in Strings · 95  
Stop\_Process · 256  
Stopp siehe Prozess beenden  
StrComp · 260  
String · 258  
Definition des Datentyps · 81  
Escape-Sequenz · 95  
nicht empfohlene Zuweisung · 96  
Steuerzeichen · 95  
Variablenaufbau · 94  
Werte normal zuweisen · 94

- String-Anweisung
  - Addition · 134
  - ASCII-Wert in Zeichen · 155
  - Dimensionierung · 258
  - Float in String · 181
  - Float in String (40 Bit) · 183
  - Länge eines Strings · 263
  - Long in String · 213
  - String in Float · 274
  - String in Long · 276
  - Teilstring
    - linker · 261
    - mittlerer · 264
    - rechter · 266
  - Vergleichen · 260
  - Zeichen in ASCII-Wert · 152
- StrLeft · 261
- StrLen · 263
- StrMid · 264
- StrRight · 266
- Strukturieren
  - Ein-/ausfalten · 22
  - Farbig Darstellung · 21
  - Programmabschnitte · 100
  - Zeilen einrücken · 22
- Sub · 268
- Subtraktion · 136
- Suchen
  - Beispiele · 28
  - Deklaration von
    - Befehl/Variable · 33
  - Reguläre Ausdrücke · 29
  - schnell · 24
  - Text · 24
- SX, siehe Externer Speicher
- symbolische Namen · 77
- Syntax
  - Colors · 50
  - Farbige Darstellung · 21

- Systemvariablen
  - Process\_Error · 237
  - Processdelay · 234
  - ProcessN\_Running · 238
  - Übersicht · 85

## T

- T10 · 192
- T11
  - Bedingung mit #If · 192
  - Fließkomma-Zahlen · 81
  - niederpriore Prozesse · 124
  - Wartezeit einstellen · 108
- T9 · 192
- Tabsize · 49
- Tabulator
  - im String, Escape-Sequenz · 96
  - Schrittweite einstellen · 49
- Tangens: Tan · 271
- Tastatur
  - Anzeige von Einstellungen · 63
  - Kürzel · 1
- Terminierung siehe Prozess beenden
- Testpoint · 130
- Text formatieren · 21
- Text schnell suchen · 24
- text snippets · 36
- Text suchen und ersetzen · 24
- Textbausteine einfügen · 36
- Textbereich ein-/ausfalten · 22
- Textmarke · 32
- Then (If ...) · 190
- Time out · 120
- Timer siehe Zähler
- Timer-Event · 117
- Timing, siehe Zeitverhalten

Timing-Modus  
  aktivieren · 53  
  anwenden · 113  
  Fenster · 67  
  zusätzliche Prozessorzeit · 116  
To (For ...) · 185  
ToDo-Liste · 66  
Toolbox · 58  
Trace\_Mode\_Pause · 272  
Trace\_Mode\_Resume · 273  
Trace-Modus  
  aktivieren · 273  
  deaktivieren · 272  
  Trace\_Mode\_Pause · 272  
  Trace\_Mode\_Resume · 273  
Transputer, Einstellungen · 15  
Trigonometrische Funktionen  
  ArcCos · 149  
  ArcSin · 150  
  ArcTan · 151  
  Cos · 156  
  Sin · 249  
  Tan · 271  
Typkonvertierung  
  ASCII-Wert in Zeichen · 155  
  automatische · 98  
  Float in Long (nur Datentyp) · 153  
  Float in String · 181  
  Float in String (40 Bit) · 183  
  Long in Float (nur Datentyp) · 154  
  Long in String · 213  
  String in Float · 274  
  String in Long · 276

## U

Übergabeparameter anzeigen · 36  
Überlastung des Prozessors · 125  
umsteigen auf ADbasic 5 · 5  
Uncomment Block · 22  
UND-Verknüpfung · 147  
ungleich <> · 144

Unmark Controlblock · 32  
Unterprogramm  
  Allgemeines zu  
    Bibliotheken · 101  
  Allgemeines zu Makros · 100  
  Bibliothek (Lib\_Sub) · 208  
  Makro (Sub) · 268  
  Position im Programm · 78  
Until (Do ...) · 167

## V

ValF · 274  
Vall · 276  
Variablen  
  Automatisch  
    vervollständigen · 34  
  Deklaration anzeigen · 37  
  Deklaration finden · 33  
  globale · 82  
    Anzeige · 60  
    kopieren, große Anzahl · 222  
    verwendete anzeigen · 38  
  Werte hexadezimal  
    anzeigen · 61  
  initialisieren · 78  
  Initialisierung beim Booten · 11  
  lokale · 86  
    erlaubte Zeichen im  
      Namen · 86  
    Länge des Namens · 86  
    Speicherbereich festlegen · 87  
  symbolische Namen · 77  
  Übersicht · 79  
  vordefinierte Namen · 79  
  siehe auch Systemvariablen  
Vergleich  
  < = > · 144  
  Strings · 260  
Verzeichnis bei  
  Standardinstallation · 11  
Verzeichnisse einstellen · 52

Visual Basic · 130

## W

Wagenrücklauf im String, Escape-Sequenz · 96

Warten

IO\_Sleep: Gold II · 201

P1\_Sleep: Pro I-Bus · 228

P2\_Sleep: Pro II-Bus · 230

Prozessor T11: CPU\_Sleep · 157

Prozessor: NOP · 224

Sleep: Prozessor bis T10 · 250

Wartezeit genau einstellen · 108

Werkzeuge (*ADtools*<>) · 73

Werkzeugleiste · 13

Wertebereich · 80

Workspace size · 65

Wurzel · 252

aus negativer Zahl, siehe Laufzeitfehler, erkennen

## X

XOr · 278

## Z

Zahlenwerte

Schreibweise · 82

Zähler

auslesen · 239

interner, Taktzyklus · 122

Zeichenkette siehe String

Zeile, zu ~ springen · 33

Zeilen einrücken · 22

Zeilen formatieren · 21

Zeilenlänge

max. Länge · 75

Zeilenlänge.

max. mit #Include · 197

Zeilennummern · 49

Zeilenvorschub im String, Escape-Sequenz · 96

Zeit

exakter Aufruf · 123

Zykluszeit · 122

Zeitdifferenz messen · 105

Zeitoptimierung, siehe Optimierung

Zeitverhalten

Änderung durch

Debug-Modus · 113

Timing-Modus · 116

Betriebszustände

allgemein · 126

ein Prozess

zeitgesteuert · 126

extern gesteuerter

Prozess · 127

mehrere Prozesse

zeitgesteuert · 127

Informationen abfragen · 115

optimales

ein Prozess · 115

mehrere Prozessen · 114

prüfen, optimieren · 113

Zusatzspeicher (EM) · 88

zuweisen, Zahlen · 82

Zuweisung (=) · 143

<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				

