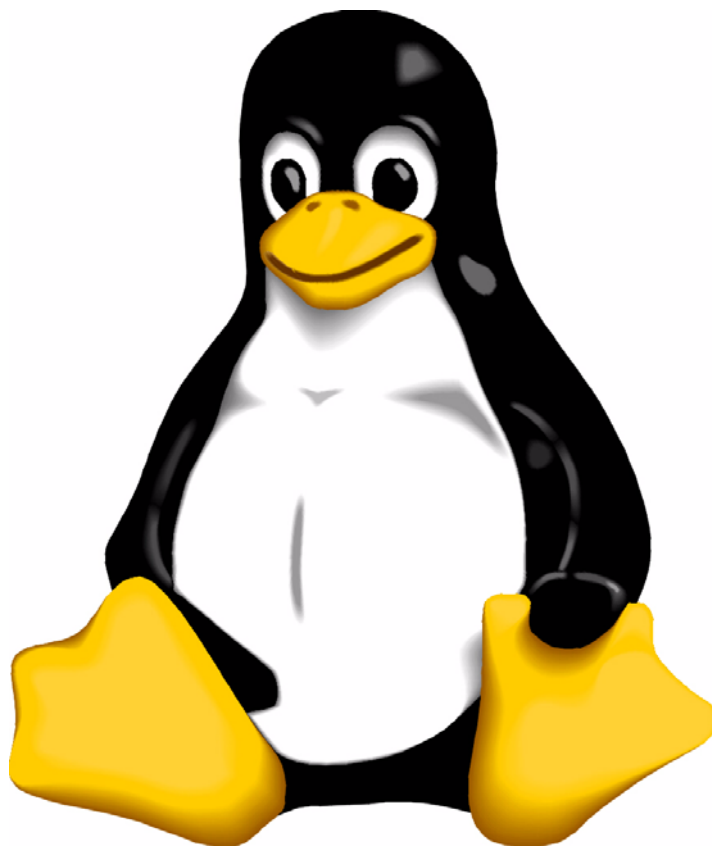


# ***ADwin für Linux***

## **Handbuch**



**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
Typografische Konventionen .....	IV
1 Zu diesem Handbuch .....	1
2 <b>ADwin</b> Linux .....	2
3 <b>ADwin</b> Linux installieren und einrichten .....	3
3.1 Wine installieren .....	3
3.2 ADwin-Software installieren .....	4
3.3 Cross-Compiler installieren für T12 / T12.1 .....	5
3.4 ADwin-System konfigurieren .....	6
3.5 Bibliothek in C einbinden .....	10
4 <b>ADbasic</b> -Quelltexte kompilieren .....	11
4.1 ADbasic- und TiCoBasic-Compiler .....	11
4.2 Aufruf ADbasic-Compiler .....	11
4.3 Aufruf TiCoBasic-Compiler .....	14
4.4 Beispiele für ADbasic .....	16
5 Methoden und Funktionen von <b>ADwin</b> Linux .....	17
5.1 Systemsteuerung und -information .....	17
5.2 Prozess-Steuerung .....	20
5.3 Übertragung von globalen Variablen .....	23
5.4 Übertragung von Datenfeldern (Arrays) .....	26
5.5 Steuerung und Fehlerbehandlung .....	35
Anhang .....	A-1
A.1 Fehlermeldungen .....	A-1
A.2 Index der Methoden und Properties .....	A-2

## Typografische Konventionen



Das „Achtung“-Zeichen steht bei Informationen, die auf Folgeschäden durch Fehlbedienung an der Hard- oder Software, am Messaufbau oder an Personen hinweisen.



Einen „Hinweis“ finden Sie bei

- Informationen, die für einen fehlerfreien Betrieb unbedingt beachtet werden müssen.
- Tipps und Ratschlägen für einen effizienten Betrieb.



Das Zeichen „Information“ verweist auf weiterführende Informationen in dieser Dokumentation oder andere Quellen wie Handbücher, Datenblätter, Literatur etc.

`C:\ADwin\...`

Dateinamen und -verzeichnisse sind in spitzen Klammern und im Schrifttyp Courier New angeben.

`Programmtext`

Programmanweisungen und Benutzer-Eingaben sind durch den Schrifttyp Courier New gekennzeichnet.

`Var_1`

Elemente eines Quelltextes wie Befehle, Variablen, Kommentar und sonstiger Text werden im Schrifttyp Courier New und farbig dargestellt.

In einem Datenwort (hier: 16 Bit) werden die Bits wie folgt nummeriert:

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

## 1 Zu diesem Handbuch

Dieses Handbuch enthält Informationen für den Einsatz der Schnittstellen-Bibliothek *ADwin* Linux sowie des *ADbasic*- und *TiCoBasic*-Compilers.

Es wird ergänzt durch

- das Handbuch „*ADwin* Installation“, das die Hardware-Schnittstellen-Installation zu allen *ADwin*-Systemen beschreibt.
- die Handbücher „*ADbasic*“ und „*TiCoBasic*“, die die Befehlsreferenzen der Compiler enthalten. Sie beschreiben außerdem
  - den Aufbau von *ADbasic*-/*TiCoBasic*-Programmen,
  - die Optimierung von *ADbasic*-/*TiCoBasic*-Programmen,
  - das Laufzeitverhalten von Prozessen.

Zum Programmieren steht Ihnen – bisher nur unter Windows – eine komfortable Bedienoberfläche als Echtzeit-Entwicklungstool zur Verfügung. Sie können stattdessen einen Editor Ihrer Wahl verwenden.

- nur bei *ADwin-Pro*-Systemen: Das Software-Handbuch mit der Befehlsreferenz zur Programmierung der Pro-Module.
- das Hardware-Handbuch für Ihr *ADwin*-System.
- Handbücher zu *ADwin*-Treibern für Programmpakete und Programmiersprachen

Es wird vorausgesetzt, dass Sie den Umgang mit der von Ihnen verwendeten Entwicklungsumgebung oder Programmiersprache beherrschen.

### Bitte beachten Sie folgende Hinweise

Damit Ihr *ADwin*-System sicher arbeitet, halten Sie sich an die Informationen dieser und weiterführender Dokumentationen, auf die hier verwiesen wird.

Der Hersteller des in dieser Dokumentation beschriebenen Systems geht davon aus, dass an dem Gerät nur qualifiziertes Personal arbeitet.

*Qualifiziertes Personal sind Personen, die aufgrund ihrer Ausbildung, Erfahrung und Unterweisung sowie ihrer Kenntnisse über einschlägige Normen, Bestimmungen, Unfallverhütungsvorschriften und Betriebsverhältnisse von dem für die Sicherheit der Anlage Verantwortlichen be-rechtigt worden sind, die jeweils erforderlichen Tätigkeiten auszuführen und die dabei mögliche Gefahren erkennen und vermeiden können. (Definition für Fachkräfte nach VDE 105 und IEC 60364).*

Diese Produktdokumentation und Unterlagen, auf die verwiesen wird, müssen stets verfügbar sein und konsequent beachtet werden. Für Schäden, die durch Missachtung der Informationen in dieser bzw. der weiterführenden Dokumentation entstehen, übernimmt die Firma *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, keine Haftung.

Diese Dokumentation ist einschließlich aller Abbildungen urheberrechtlich geschützt. Reproduktion, Übersetzung sowie elektronische und fotografische Archivierung und Veränderung bedürfen der schriftlichen Genehmigung der Firma *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch.

Fremdprodukte werden ohne Vermerk auf mögliche Patentrechte genannt, deren Existenz nicht auszuschließen ist.

Hotline-Adresse siehe vordere Umschlagseite, innen.



### Einschränkung der Anwendergruppe

### Verfügbarkeit der Unterlagen



### Rechtliche Grundlagen

Änderungen vorbehalten.

## 2 ADwin Linux

ADwin Linux ist eine Bibliothek. Sie stellt Schnittstellen-Funktionen zur Verfügung, damit Sie z. B. aus C, C++, Matlab oder LabView mit einem ADwin-System kommunizieren können.

Die Bibliothek ADwin Linux definiert Methoden, Funktionen und Systemvariablen zur Prozesssteuerung und zur Datenübertragung vom und ins ADwin-System. Die Definition umfasst jeweils den Namen einer Funktion sowie Anzahl, Datentyp und Reihenfolge der zu übergebenden Parameter.

### Kommunikation mit dem ADwin-System

Vom PC aus können Sie Prozesse im ADwin-System steuern, sowie Daten von dort anfordern oder dorthin senden. Die Prozesse selbst programmieren Sie in *ADbasic* (siehe gleichnamiges Handbuch) und kompilieren Sie mit dem mitgelieferten Compiler (siehe [Kapitel 4](#)).

Die Bibliotheks-Funktionen kommunizieren mit dem Echtzeitkern des ADwin-Systems, dem Betriebssystem. Deshalb müssen Sie nach jedem Einschalten des Systems zunächst das Betriebssystem (in Form einer Datei, z. B. `<adwin9.btl>`) dorthin laden. Nach erfolgreicher Übertragung kann das System Prozesse empfangen und ausführen, Befehle vom PC entgegen nehmen und Daten mit ihm austauschen. Die in *ADbasic* programmierten Prozesse enthalten den Programmcode zur Messung, Steuerung oder Regelung Ihrer Applikation.

Die Aufgaben des Betriebssystems sind:

- Verwaltung von bis zu 10 Echtzeit-Prozessen mit niedriger oder hoher Priorität (frei wählbar). Niedrig priorisierte Prozesse können von hoch priorisierten Prozessen unterbrochen werden, letztere können nicht von anderen Prozessen unterbrochen werden.
- Bereitstellung von globalen Variablen:
  - 80 Integer-Variablen (`PAR_1 ... PAR_80`), bereits vordefiniert
  - 80 Float-Variablen (`FPAR_1 ... FPAR_80`), bereits vordefiniert
  - 200 Datenfelder (`DATA_1 ... DATA_200`), frei definierbare Länge

Sie können die Werte dieser Variablen bzw. Datenfelder vom PC aus jederzeit lesen und ändern. Außerdem ist die Definition lokaler Variablen und Felder möglich.

- Kommunikation zwischen ADwin-System und PC .

Der Kommunikationsprozess läuft mit mittlerer Priorität auf dem ADwin-System und kann niedrig priorisierte Prozesse für kurze Zeit unterbrechen. Er interpretiert oder bearbeitet alle Befehle, die Sie vom PC an das ADwin-System richten: Steuerbefehle und Befehle für den Datenaustausch. Die folgende Tabelle zeigt Beispiele aus jeder Gruppe.

Steuerbefehle, z. B.	
<code>LOAD_PROCESS</code>	überträgt einen Prozess auf das System
<code>START_PROCESS</code>	startet einen Prozess
Befehle für den Datenaustausch, z. B.	
<code>GET_PAR</code>	liefert den aktuellen Wert eines Parameters
<code>SET_PAR</code>	ändert den Wert eines Parameters
<code>GETDATA_LONG</code>	liefert die Werte aus einem DATA-Feld

Der Kommunikationsprozess sendet niemals unaufgefordert Daten an den PC. Das stellt sicher, dass nur dann Daten zum PC übertragen werden, wenn Sie diese ausdrücklich angefordert haben.



Echtzeitkern

10 Prozesse

Datenspeicher

Kommunikation



### 3 ADwin Linux installieren und einrichten

Für die Installation der Bibliothek *ADwin* Linux benötigen Sie:

- Linux Kernel ab 2.6 für *ADwin*-Systeme mit Ethernet-Anschluss.
- Build-Umgebung je nach Linux-Distribution:
  - Debian/Ubuntu: `build-essential`
  - Red Hat (Fedora, CentOS): `@development-tools`
  - openSUSE: `devel_C_C++`
  - solus: `system.devel`
- Folgende Linux-Pakete: `cmake`, `bison`, `flex`, `gengetopt`
- Wine ab Version 1.4 für *ADbasic* und für *TiCoBasic*.

Für ein älteres *ADwin*-System mit Link-Datenverbindung oder für MAC OS ist diese Version der Bibliothek *ADwin* Linux ungeeignet. Verwenden Sie stattdessen die vorherige Version 5.00. Unter Linux benötigen Sie den Kernel 2.4.

Die Installation wird mit folgenden Schritten durchgeführt:

- [Wine installieren](#)
- [ADwin-Software installieren](#)
- [Cross-Compiler installieren für T12 / T12.1](#)
- [ADwin-System konfigurieren](#)
- [Bibliothek in C einbinden](#)

Teilweise benötigen Sie für die Installation „root“-Rechte. Wenn bei einer Eingabe „root“-Rechte erwartet werden, ist im folgenden das Zeichen # am Anfang der Eingabezeile angegeben, sonst das Zeichen \$.

Bei manchen Linux-Ausgaben ist der „root“-Account aus Sicherheitsgründen deaktiviert. Rufen Sie in diesem Fall den Befehl über das Programm `sudo` auf, also z.B. `sudo make install`.

#### 3.1 Wine installieren

Zur Installation von Wine konsultieren Sie bitte die Hilfe zur Installation von zusätzlichen Paketen für Ihre Distribution. Manche Distributionen erlauben die separate Installation von Wine für 32 Bit und 64 Bit Windows-Emulation. Da *ADbasic* und *TiCoBasic* reine 32-Bit Applikationen sind, wählen Sie in dem Fall das Paket `wine32`, unabhängig davon, ob Ihr Linux-System ein 32-Bit oder 64-Bit System ist.

Unter Ubuntu genügt normalerweise:

```
# apt-get install wine-stable
```

Unter Debian muss explizit der 32 Bit-Support eingeschaltet werden:

```
# dpkg --add-architecture i386
# apt-get update
# apt-get upgrade
# apt-get install wine wine32
```



## Inhalt des ADwin-Software-Pakets

## Archive installieren

## Umgebungsvariable

### 3.2 ADwin-Software installieren

Im *ADwin*-Software-Paket finden Sie das Verzeichnis `./LINUX`, in dem sich folgende zip-Archive befinden:

- `adwin-lib-x.y.zip`: Das Archiv (Version `x.y`) enthält:
  - *ADwin* Linux-Bibliothek (shared library).
  - Konfigurations-Programm `adconfig`.
- `adwin-compiler-x.y.zip`: *ADbasic*-Compiler.
- `adwin-labview.zip`: Der Treiber für die Bedienoberfläche LabView (vollständige Installation siehe Handbuch „LabView-Treiber“).
- `adwin-matlab.zip`: Der Treiber für MATLAB (vollständige Installation siehe Handbuch „Matlab-Treiber“).
- `adwin-doc-x.y.zip`: Alle Dokumentationen für Software und Hardware im Format pdf.

Installieren Sie die Archive in der Reihenfolge wie oben angegeben. Beginnen Sie mit `adwin-lib-x.y.zip` (bitte Hinweise weiter unten beachten).

Sie installieren ein Archiv mit den folgenden Schritten:

1. Entpacken Sie das Archiv `<filename>.zip`.
2. Führen Sie folgende Schritte aus:
 

```
$ cd adwin-lib-x.y
$ mkdir build -p
$ cd build
$ cmake .. -DCMAKE_INSTALL_PREFIX=/opt/adwin
$ cmake --build .
# make install
```

Über den Prefix `DCMAKE_INSTALL_PREFIX` werden die Dateien im Verzeichnis `/opt/adwin/` installiert (diese Einstellung wird im folgenden angenommen). Das Verzeichnis wird in die Datei `/etc/adwin/ADWINDIR` eingetragen.

3. Installieren Sie die übrigen Archive in der gleichen Weise.

Der Prefix muss nicht mehr angegeben werden, der Verzeichnispfad wird aus `/etc/adwin/ADWINDIR` ausgelesen.

Am Ende erhalten Sie die unten dargestellte Verzeichnisstruktur (siehe [Seite 5](#)).

In früheren Versionen der Linux-Bibliothek wurde in der Umgebungsvariablen `ADWINDIR` das Installationsverzeichnis abgelegt. Das ist auch weiter möglich, aber nicht erforderlich.

Um die Variable zu setzen, können Sie

- das Verzeichnis direkt angeben:
 

```
$ export ADWINDIR=/opt/adwin/
```
- oder die beim Installieren verwendete Datei benutzen:
 

```
$ export ADWINDIR=`cat /etc/adwin/ADWINDIR/`
```

Wenn Sie `bash` verwenden, sind vielleicht folgende Zeilen zusätzlich sinnvoll:

```
$ export PATH=$PATH:$ADWINDIR/bin:$ADWINDIR/sbin
$ export MANPATH=$MANPATH:$ADWINDIR/man
```

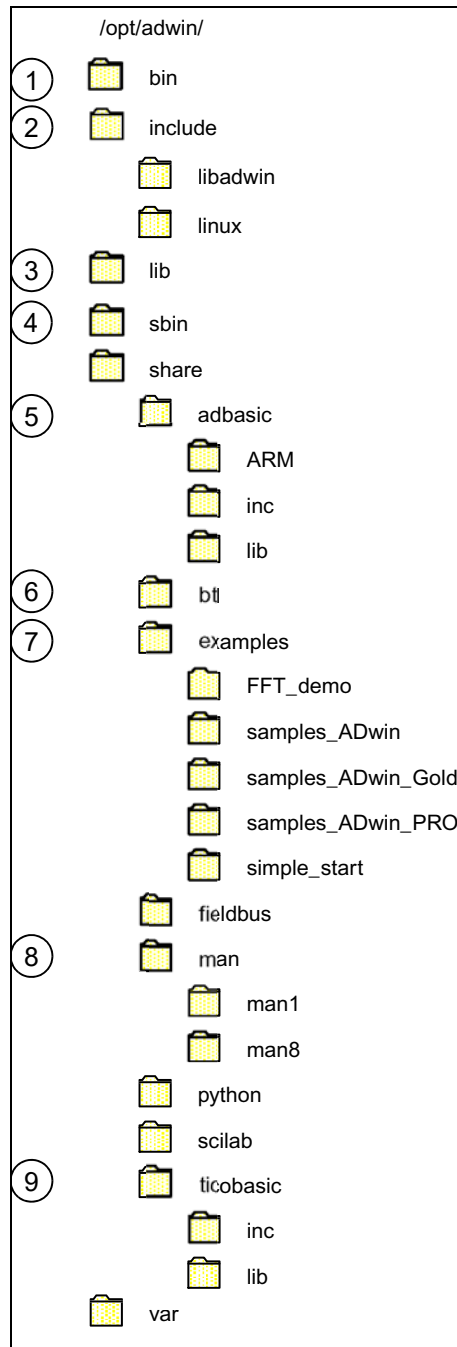
Alternativ können Sie den Pfad `MANPATH` auch systemweit in der Datei `/etc/manpath.config` eintragen.



Nach der Installation ist die rechts gezeigte Verzeichnisstruktur vorhanden.

Die wichtigsten Dateien und Programme sind:

1. Der *ADbasic*- und der *TiCoBasic*-Compiler.
2. Include-Dateien für die Programmentwicklung.
3. *ADwin*-Programmbibliothek.
4. Das Programm *ADconfig*, mit dem Sie einem *ADwin*-System eine Device No. zuweisen.
5. Include- und Library-Dateien für den *ADbasic*-Compiler.
6. Betriebssystemdateien \*.btl für *ADwin*-Prozessoren.
7. *ADbasic*-Beispiele, sortiert nach *ADwin*-Systemen.
8. Handbücher
  - man1: *ADbasic*- und *TiCoBasic*-Compiler
  - man8: *ADconfig*
9. Include- und Library-Dateien für den *TiCoBasic*-Compiler.



### Installierte Verzeichnisstruktur

### 3.3 Cross-Compiler installieren für T12 / T12.1

Bei den *ADwin*-Prozessoren T12 und T12.1 wird vom *ADbasic*-Compiler C-Code erzeugt, der anschließend automatisch von einem Cross-Compiler in *ADbasic*-Prozesse übersetzt wird. Unter Windows wird dafür der Linaro GCC Compiler arm-linux-gnueabi-4.9-2014.09 verwendet, den Sie auch manuell unter Linux installieren können. Wir haben jedoch gute Erfahrungen damit gemacht, einfach den aktuellsten Cross-Compiler der Distribution einzusetzen.

So installieren Sie den C-Compiler:

- Debian und Ubuntu
  - # apt-get install gcc-arm-linux-gnueabi

- RedHat-basierte Distributionen wie Fedora, CentOS  

```
# dnf copr enable lantw44/arm-linux-gnueabi-hf-toolchain
# dnf install arm-linux-gnueabi-hf-{binutils,gcc,glibc}
```

- openSUSE

Beziehen Sie das Paket `linaro-toolchain` von der Adresse <https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/arm-linux-gnueabi-hf/>

Erstellen Sie anschließend Links zu allen Binaries in `/usr/bin`:

```
# ln -s [BINARY-DOWNLOAD] /bin/arm-linux-gnueabi-hf*
/usr/bin/
```

Setzen Sie die Umgebungsvariable  
`CROSS_COMPILE=arm-linux-gnueabi-hf-`

### 3.4 ADwin-System konfigurieren

Jedes *ADwin*-System wird unter Linux über eine sogenannte „Device No.“ angesprochen. Zum Einrichten einer Device No. verwenden Sie das Programm `adconfig` im Verzeichnis `/opt/adwin/sbin/`:

- Melden Sie die Device No. des *ADwin*-Systems unter Linux an (nach der Installation ist noch keine Device No. angemeldet). Der PC verwendet die Anmeldungsdaten, um das *ADwin*-System anzusprechen.
- Wenn das *ADwin*-System eine Ethernet-Schnittstelle besitzt, müssen Sie auch das System konfigurieren (siehe [Seite 8](#)).

Die Konfigurationsdaten werden nicht auf dem PC gespeichert, sondern an das *ADwin*-System übertragen. Die Daten müssen mit den Anmeldungsdaten unter Linux übereinstimmen.

Sie finden im Handbuch „ADwin Installation“ eine Beschreibung der Grundlagen zum Ethernet-Betrieb.



Zum Aufruf des Programms `adconfig` benötigen Sie „root“-Rechte (`su`).

#### 3.4.1 Beispiel: Standard-Konfiguration

Mit den folgenden Zeilen richten Sie die Device No. 0x150 für ein *ADwin*-System mit Ethernet-Schnittstelle ein:

1. Melden Sie die Device No. 0x150 (dezimal:336) mit einer festen IP-Adresse (hier: 10.100.100.83) an:  

```
# adconfig add 0x150 TYPE net IP 10.100.100.83
```
2. Lassen Sie sich den angemeldeten Eintrag anzeigen  

```
$ adconfig

[0x150] UID=0 GID=0 MODE=0666 TYPE=net DESC=""
HOSTLINK=0x0 PW="" COUNT=5 TIMEOUT=1000 PORT=6543
IP=10.100.100.83
```
3. Konfigurieren Sie das *ADwin*-System mit MAC-Adresse, IP-Adresse und Subnet Mask:  

```
# adconfig config 00:50:C2:0A:22:DD IP 10.100.100.83
MASK 255.255.255.0
```

Die Device No. 0x150 ist nun eingerichtet und Sie können mit dem *ADwin*-System arbeiten.

### 3.4.2 Konfiguration ausführlich

Mit dem Programm `adconfig` können Sie folgende Funktionen durchführen:

- Liste aller unter Linux angemeldeten Device No. anzeigen:  
\$ `adconfig`
- Neue Device No. unter Linux anmelden:  
# `adconfig add <deviceno> [OPTIONS] TYPE net`
- ADwin-System mit Ethernet-Schnittstelle konfigurieren:  
# `adconfig config <deviceno> [OPTIONS]`  
Verwenden Sie die gleichen Daten, die Sie mit `add` angemeldet haben!
- Eine angemeldete Device No. löschen:  
# `adconfig del <deviceno>`
- Hilfe zum Programm `adconfig` anzeigen:  
\$ `adconfig --help`
- Versionsnummer des Programms anzeigen:  
\$ `adconfig --version`

Der Parameter `<deviceno>` kann in dezimaler oder hexadezimaler Schreibweise angegeben werden, z.B. 336 oder 0x150.

Die Details zu den Funktionen des Programms sind nachfolgend beschrieben oder können mit `man 8 adconfig` angezeigt werden.

#### Liste aller Device No. anzeigen

```
$ adconfig
```

Es wird eine Liste aller unter Linux angemeldeten Device No. mit deren Parametern angezeigt. Die Device No. wird hexadezimal angezeigt, z.B. 0x150 (dezimal: 336).

#### Neue Ethernet-Device No. unter Linux anmelden

```
# adconfig add <deviceno> [UID <uid|username>]
[GID <gid|groupname>] [MODE <mode>] [DESC <" ">]
<TYPE> net IP <addr> [PW <" ">] [PORT <#>] [TIMEOUT <#>]>
```

<code>&lt;deviceno&gt;</code>	Die anzumeldende Device No.
<code>UID &lt;uid username&gt;</code>	Name des Nutzers, der das ADwin-System nutzen darf (Voreinstellung: root)
<code>GID &lt;gid groupname&gt;</code>	Name der Gruppe, die das ADwin-System nutzen darf (Voreinstellung: root)
<code>MODE &lt;mode&gt;</code>	Zugriffsrechte der Nutzer auf das ADwin-System (Voreinstellung: 0666)
<code>DESC &lt;" "&gt;</code>	Beschreibungstext (bis 32 Zeichen, Voreinstellung: "")
<code>TYPE net IP &lt;addr&gt;</code>	ADwin-System mit Ethernet-Schnittstelle. <code>&lt;addr&gt;</code> : IP-Adresse im Ethernet-Netzwerk
<code>PW &lt;" "&gt;</code>	Passwort (Voreinstellung: ""), maximal 10 Zeichen
<code>PORT &lt;#&gt;</code>	Netzwerk Port Nummer (Voreinstellung: 6543)
<code>TIMEOUT &lt;#&gt;</code>	Timeout des Netzwerk-Protokolls (Voreinstellung: 1000ms)

Geben Sie nur die Parameter an, die keine Voreinstellung haben oder deren Voreinstellung sie ändern möchten.

Die mit `UID`, `GID` und `MODE` eingestellten Zugriffsrechte stellen bei Ethernet kaum mehr als einen Schutz vor versehentlicher Fehlbedienung dar, weil die Einstellungen umgangen werden können.

#### Device No. anzeigen

#### Device No. anmelden: Ethernet





**Device No. anmelden:  
ADwin-System am Host-  
Rechner**

**Ethernet-Gerät ohne  
DHCP konfigurieren**

Die Einrichtung eines *ADwin*-Systems in einem Ethernet-Netzwerk ist im Installations-Handbuch beschrieben. Dort finden Sie auch eine genauere Beschreibung der Netzwerk-spezifischen Parameter.

Achten Sie darauf, dass eine IP-Adresse in einem Ethernet-Netzwerk nicht doppelt vergeben werden darf. Anderenfalls können erhebliche Kommunikations-Probleme auftreten.

#### Beispiel

```
# adconfig add 0x3A TYPE net IP 10.100.100.83
```

Lassen Sie sich anschließend den definierten Eintrag anzeigen:

```
$ adconfig
```

```
[0x3A] UID=0 GID=0 MODE=0666 TYPE=net DESC=""
HOSTLINK=0x0 PW="" COUNT=5 TIMEOUT=1000 PORT=6543
IP=10.100.100.83
```

#### Device No. anmelden – *ADwin*-System am Host-Rechner

In einem IP-Netzwerk (z. B. Ethernet) kann ein Client-PC ein *ADwin*-System an einem anderen Rechner (Host-PC) auch dann ansprechen, wenn es in einem anderen Netzwerk betrieben wird.

Voraussetzungen hierfür:

- Der Host-PC hat Zugriff auf das *ADwin*-System.
- Der Host-PC kann vom Client-PC über das Netzwerk angesprochen werden.
- Das Programm *ADwinTcpiServer* läuft auf dem Host-PC (derzeit nur unter Windows).

Wenn die Voraussetzungen erfüllt sind, müssen die Anmeldeungs-Parameter wie folgt verwendet werden:

```
# adconfig add <deviceno> ...
TYPE net IP <addr> [PW <" ">] [PORT <#>]
[TIMEOUT <#>] HOSTLINK <#>
```

net IP <addr>	IP-Adresse des Host-Rechners
PW <" ">	Die Parameter müssen hier identisch sein mit den Einstellungen, die am Host-PC im Programm <i>ADwinTcpiServer</i> vorgenommen wurden (siehe Online-Hilfe des Programms).
PORT <#>	
TIMEOUT <#>	
HOSTLINK <#>	Zusätzlich: Device No. des <i>ADwin</i> -Systems, die am Host-PC mit <i>adconfig</i> eingestellt ist.

#### *ADwin*-System mit Ethernet-Schnittstelle konfigurieren (ohne DHCP)

```
# adconfig config <hwaddr> IP <addr> MASK <addr>
[GATEWAY <addr>] [PORT <#>] [PW <" ">]
```

<hwaddr>	Die MAC-Adresse des <i>ADwin</i> -Systems im Format 00:50:C2:0A:nn:nn.
IP <addr>	IP-Adresse des <i>ADwin</i> -Systems im Ethernet-Netzwerk im Format nnn.nnn.nnn.nnn.
MASK <addr>	Subnet mask im Format nnn.nnn.nnn.nnn.
GATEWAY <addr>	IP-Adresse für Default gateway (Voreinstellung: 0.0.0.0, d.h. kein gateway)
PORT <#>	Netzwerk Port-Nummer (Voreinstellung: 6543)
PW <" ">	Neues Passwort, bis zu 10 alphanumerische Zeichen (Voreinstellung: "", d.h. das existierende Passwort wird entfernt!)
DESC <" ">	Beschreibungstext, bis zu 16 Zeichen (Voreinstellung: "")

Geben Sie nur die Parameter an, die keine Voreinstellung haben oder deren Voreinstellung sie ändern möchten. Verwenden Sie die gleichen Werte, die Sie mit `add` angemeldet haben!

Beachten Sie bitte, dass beim Konfigurieren immer ein Passwort übermittelt wird. Wenn Sie ein vorher eingestelltes Passwort beibehalten möchten, müssen Sie das Passwort also angeben, sonst wird es gelöscht.

### Beispiel

```
# adconfig config 00:50:C2:0A:22:DD IP 10.100.100.83
  MASK 255.255.255.00
```

### ADwin-System mit Ethernet-Schnittstelle für DHCP konfigurieren

Es ist möglich, aber keineswegs empfohlen, dass ein ADwin-System seine IP-Adresse von einem DHCP-Server dynamisch zugewiesen bekommt. Hierzu aktivieren Sie die Option DHCP.

```
# adconfig config <hwaddr> [GATEWAY <addr>] [PORT <#>]
  DHCP yes [PW <" ">]
```

<hwaddr>	Die MAC-Adresse des ADwin-Systems im Format 00:50:C2:0A:nn:nn.
GATEWAY <addr>	IP-Adresse für Default gateway (Voreinstellung: 0.0.0.0, d.h. kein gateway)
PORT <#>	Netzwerk Port-Nummer (Voreinstellung: 6543)
PW <" ">	Neues Passwort, bis zu alphanumerische 10 Zeichen (Voreinstellung: "", d.h. das existierende Passwort wird entfernt!)
DHCP <no   yes>	Festlegung, ob ein DHCP-Server verwendet wird (Voreinstellung: no); siehe unten.
DESC <" ">	Beschreibungstext, bis zu 16 Zeichen (Voreinstellung: "")

Wir empfehlen, die Option DHCP nicht zu aktivieren!  
Bei aktiver Option können oft Kommunikationsprobleme auftreten.

Geben Sie nur die Parameter an, die keine Voreinstellung haben oder deren Voreinstellung sie ändern möchten. Verwenden Sie die gleichen Werte, die Sie mit `add` angemeldet haben!

### Angemeldete Device No. löschen

```
# adconfig del <deviceno>
```

Die Device No. wird unter Linux abgemeldet und kann nicht mehr angesprochen werden.



### Ethernet-Gerät mit DHCP konfigurieren



### Device No. löschen

### 3.5 Bibliothek in C einbinden

Um die Funktionen der Bibliothek *ADwin* Linux in C oder C++ zu verwenden, gehen Sie vor wie folgt:

- Fügen Sie folgende Zeilen in Ihr Programm ein:
  - Funktionen zum Zugriff auf das *ADwin*-System:  
`#include <libadwin.h>`
  - Fehlernummern, die beim Zugriff auf das *ADwin*-System auftreten können:  
`#include <libadwin/errno.h>`

- Fügen Sie für den C-Compiler einige Flags in Ihr Makefile ein, um Ihr Programm mit der Bibliothek *ADwin* Linux zu kompilieren:

```
$ CFLAGS += -I$(ADWINDIR)/include -L$(ADWINDIR)/lib  
-ladwin  
$ CFLAGS += -Wl,--rpath -Wl,$(ADWINDIR)/lib
```

Beachten Sie, dass hier die Umgebungsvariable `ADWINDIR` verwendet wird (Setzen der Variablen siehe [Seite 4](#)).

Anstatt den Pfad zur Bibliothek an den Linker zu übergeben, können Sie den Pfad auch in der Datei `/etc/ld.so.conf` eintragen und `ldconfig` aufrufen.

Wenn Sie Threads benutzen, müssen Sie zusätzlich hinzufügen:

```
$ CFLAGS += -D_REENTRANT
```

#### Portierung von Windows

Die Funktionen der Bibliothek *ADwin* Linux sind kompatibel zu den Funktionen der Schnittstelle unter Windows (`adwin32/64.dll`). Sie können daher Programm-Quelltexte, die Sie in einer Programmiersprache (hier: C / C++) unter Windows erstellt haben, in der Regel ohne Änderung unter Linux verwenden (natürlich nur, soweit es um den Zugriff auf *ADwin*-Systeme geht).

Änderungen im Programm sind nur dort erforderlich, wo Pfadangaben oder Dateinamen verwendet werden. Hier machen sich die typischen Unterschiede zwischen Windows und Linux bemerkbar: Syntax und Groß- und Kleinschreibung der Pfadnamen.

Für *ADbasic*-Quelltexte ist eine Portierung von (und nach) Windows in gleicher Weise möglich.

### 4 ADbasic-Quelltexte kompilieren

In einem *ADbasic*-Quelltext programmieren Sie die Funktion eines Prozesses, der auf dem *ADwin*-System abläuft. Sie erzeugen einen *ADbasic*-Quelltext mit einem Editor Ihrer Wahl. Unter Windows steht Ihnen eine komfortable Bedienoberfläche als Echtzeit-Entwicklungstool zur Verfügung.

Mit dem *ADbasic*-Compiler erzeugen Sie aus dem Quelltext Binärcode, der auf der passenden *ADwin*-Hardware lauffähig ist.

#### 4.1 ADbasic- und TiCoBasic-Compiler

Mit Version 6 wurde der *ADbasic*- und *TiCoBasic*-Compiler für Windows so angepasst, dass beide ohne Änderungen mit Wine (Version ab 1.4) unter Linux ausgeführt werden können. Sämtliche Include- und Library-Dateien sind unter Windows und Linux identisch.

Vor dem ersten Compiler-Aufruf müssen Sie mit dem Hilfsprogramm *adwin-license* Ihren License key eingeben. Erst danach können Sie Binär- oder Library-Dateien erzeugen.

Sie finden den License key auf dem Deckblatt Ihres *ADbasic*-Handbuchs.

Beim ersten Start von *adwin-license*, *adbasic* oder *ticobasic* legt Wine eine virtuelle Windows-Registry an, in der z.B. der Lizenzschlüssel abgelegt wird. Daher kann der erste Start verhältnismäßig lange dauern.

Beide Compiler werden in Linux aus der Kommandozeile (Shell) aufgerufen. Der Compiler übersetzt Quelltext und erzeugt – je nach Schalter und Parameter – entweder eine Binär-Datei oder eine Library-Datei. Die Funktion von Binär- und Library-Dateien ist im *ADbasic-/TiCoBasic*-Handbuch erläutert.

Ein oder mehrere Kommandozeilen-Aufrufe können in einem Shell-Skript oder Makefile zusammengefasst werden, um mehrere Quelltexte eines Projekts mit nur einem Aufruf zu kompilieren.



#### 4.2 Aufruf ADbasic-Compiler

Es gibt 5 Hauptschalter beim Aufruf des *ADbasic*-Compilers:

- |   |  |
|---|--|
| 1. <code>adbasic /VER</code>                            | Versionsnummer des Compilers anzeigen.   |
| 2. <code>adbasic /H</code>                              | Hilfe aufrufen.  |
| 3. <code>adbasic /MAKE"make-file"</code>                | Hauptschalter, Dateiname und weitere Schalter für einen Aufruf aus dem <code>make-file</code> entnehmen.<br><br>Der Text im <code>makefile</code> kann über mehrere Zeilen verteilt geschrieben werden. Schalter außerhalb des <code>makefile</code> sind nicht erlaubt. |
| 4. <code>adbasic /M [OPTIONS] &lt;infile.bas&gt;</code> | Einen <i>ADbasic</i> -Quelltext kompilieren und eine Binär-Datei erzeugen.   |
| 5. <code>adbasic /L [OPTIONS] &lt;infile.bas&gt;</code> | Einen <i>ADbasic</i> -Quelltext kompilieren und eine Library-Datei erzeugen.   |

Die Schalter `[OPTIONS]` für das Kompilieren eines Quelltexts sind nachfolgend beschrieben.

Wenn Dateinamen ohne oder mit relativem Pfadnamen angegeben werden, wird als Basis das Arbeitsverzeichnis verwendet, aus dem heraus der Compiler aufgerufen wird.

**Syntax**

```
[path/]adbasic /M [path/]infile.bas
    [/A<fout>] [/IP<path>] [/LP<path>] [/S<system>]
    [/P<ptype>] [/E<event>] [/PN<#>] [/P<prio>]
    [/PD<cycle>] [/PS<stack>] [/O<level>]
    [/L<lang>] [/V<#>]

[path/]adbasic /L [path/]infile.bas
    [/A<fout>] [/IP<path>] /LP<path>] [/S<system>]
    [/P<ptype>] [/O<level>] [/L<lang>]
```

**Schalter**

[path/]	Verzeichnis, in dem sich das Programm adbasic befindet, bei Standardinstallation: /opt/adwin/bin/
infile.bas	Dateiname des zu kompilierenden Quelltextes. Der Pfad ist optional.
/A<fout>	Optional: Name der zu erzeugenden Binär- oder Library-Datei, ohne Dateiendung angeben.  Eine Binärdatei wird mit der Endung .Txn erzeugt, eine Library-Datei (Bibliothek) mit der Endung .Lix. x      Prozessortyp; siehe <ptype> beim Schalter /P. n      Prozessnummer; siehe <#> beim Schalter /PN.
/IP<path>	Setzt den Suchpfad für Include/Dateien; Default ist der Eintrag in der Registry.
/LP<path>	Setzt den Suchpfad für Library/Dateien; Default ist der Eintrag in der Registry.
/S<system>	Hardware, für die die Datei kompiliert wird: /SC    Karten (ISA/Bus) /SL    Light/16 /SG    Gold; Default /SGII   Gold II /SP    Pro /SPII   Pro II /SX    X-A20
/P<ptype>	Prozessortyp, für den die Datei kompiliert wird: /P2    Prozessor T2 /P4    Prozessor T4 /P5    Prozessor T5 /P8    Prozessor T8 /P9    Prozessor T9; Default /P10   Prozessor T10 /P11   Prozessor T11 /P12   Prozessor T12 /P121   Prozessor T12.1
/E<event>	Ereignisquelle für den Prozess wählen: /ET    Timer; Default /EE    Extern
/PN<#>	Setzt die Prozessnummer (1...10) der zu kompilierenden Datei; Default = 1.
/P<prio>	Setzt die Priorität des Prozesses auf:



- /PH      hoch (high); Default
- /PL      niedrig (low), Priorität 1.
- /PLn     niedrig, Level n (-10...10).  
          Level 10 hat die höchste Priorität.
- /PD<cycle>    Zykluszeit (Processdelay) des Prozesses auf `cycle`  
                  einstellen. Ohne den Schalter ist die Voreinstellung  
                  1000, für T11 3000.
- /PS<stack>    nur mit Option /P12 oder /P121: Größe des Programm-  
                  Stapelspeichers (stack size) auf x einstellen, Default:  
                  1000.
- /O<level>     Legt die Optimierungsstufe fest. Die Optimierung kann  
                  die Ausführungszeit verkürzen, aber in Ausnahmefällen  
                  auch Laufzeitfehler verursachen.
- /O0      Optimierungsstufe 0 (=keine Optimierung)
- /O1      Optimierungsstufe 1 (Default)
- /O2      Optimierungsstufe 2
- /O3      Optimierungsstufe 3 (nur mit Option /P12)
- /Os      Geringe Speichergröße (nur mit Option /P12)
- /Of      Hohe Geschwindigkeit (nur mit Option /P12)
- /L<lang>      Sprache, in der Fehlermeldungen und Warnungen aus-  
                  gegeben werden.
- /LE      Englisch. Default.
- /LG      Deutsch
- /V<#>        Setzt die Versionsnummer des Prozesses; Default = 1.

### Bemerkungen

Optionale Angaben sind in eckige Klammern gesetzt. Die Reihenfolge der Schalter ist beliebig. In Kommandozeilen wird zwischen Groß- und Kleinschreibung unterschieden.

Wenn der Schalter /A nicht verwendet wird, wird die erzeugte Binär- oder Library-Datei in dem Verzeichnis gespeichert, in dem sich der Quelltext befindet.

Die Hardware-Schalter können mit folgenden Prozessor-Schaltern kombiniert werden:

Hardware	Prozessor-Schalter
/SC	/P2, /P4, /P5, /P8
/SG, /SL	/P9
/SP	/P4, /P5, /P8, /P9, /P10
/SGII	/P11
/SPII	/P11, /P12
/SX	/P121

Wenn während des Kompilierens ein Fehler auftritt, bricht der Compiler seine Arbeit ab und erzeugt keine Binär- oder Library/Datei.

### 4.3 Aufruf TiCoBasic-Compiler

Es gibt 5 Hauptschalter beim Aufruf des *TiCoBasic*-Compilers:

1. `ticobasic /VER` Versionsnummer des Compilers anzeigen.
2. `ticobasic /H` Hilfe aufrufen.
3. `ticobasic`  
`/MAKE"makefile"` 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.
4. `ticobasic /M [OPTIONS] /PROCESS`  
`<infile#.bas>`  
`[PROCESS OPTIONS]` Einen oder mehrere *TiCoBasic*-Quelltexte kompilieren und eine Binär-Datei erzeugen.  
  
Für jeden Quelltext ist jeweils separat anzugeben: `/PROCESS <infile#.bas> [PROCESS OPTIONS]`.
5. `ticobasic /L [OPTIONS] <infile.bas>` Einen *TiCoBasic*-Quelltext kompilieren und eine Library-Datei erzeugen.

Die Schalter `[OPTIONS]` für das Kompilieren eines Quelltexts sind nachfolgend beschrieben.

Wenn Dateinamen ohne oder mit relativem Pfadnamen angegeben werden, wird als Basis das Arbeitsverzeichnis verwendet, aus dem heraus der Compiler aufgerufen wird.

#### Syntax

```
[path/]ticobasic /M [path/] [/A<fout>] [/IP<path>]
[/LP<path>] [/S<system>] [/P<ptype>] [/L<lang>]
/PROCESS infile.bas [/E<event>] [/EE<ext>]
[/PN<#>] [/P<prio>] [/PD<cycle>] [/O<level>]
[/V<#>]

[path/]ticobasic /L [path/]infile.bas
[/A<fout>] [/IP<path>] /LP<path> [/S<system>]
[/P<ptype>] [/L<lang>] [/O<level>]
```

#### Schalter

<code>[path/]</code>	Verzeichnis, in dem sich das Programm <code>ticobasic</code> befindet, bei Standardinstallation: <code>/opt/adwin/bin/</code>
<code>/A&lt;fout&gt;</code>	Optional: Name der zu erzeugenden Binär- oder Library-Datei, ohne Dateiendung angeben.  Eine Binärdatei wird mit der Endung <code>.TIn</code> erzeugt, eine Library-Datei (Bibliothek) mit der Endung <code>.TLx</code> .
<code>x</code>	Prozessortyp; siehe <code>&lt;ptype&gt;</code> beim Schalter <code>/P</code> .
<code>n</code>	Prozessnummer; siehe <code>&lt;#&gt;</code> beim Schalter <code>/PN</code> .
<code>/IP&lt;path&gt;</code>	Setzt den Suchpfad für Include/Dateien; Default ist der Eintrag in der Registry.

`/LP<path>` Setzt den Suchpfad für Library/Dateien;  
Default ist der Eintrag in der Registry.

`/S<system>` Hardware, für die die Datei kompiliert wird:  
`/SGII` Gold II  
`/SPII` Pro II; Default  
`/SX` X-A20

`/P<ptype>` Prozessortyp, für den die Datei kompiliert wird:  
`/P1` Prozessor TiCo1  
`/P2` Prozessor TiCo1

`/L<lang>` Sprache, in der Fehlermeldungen und Warnungen ausgegeben werden.  
`/LE` Englisch. Default.  
`/LG` Deutsch

`/PROCESS` Schlüsselwort für die Optionen des nächsten Quelltexts  
`infile.bas`. Muss für jeden Quelltext wiederholt werden.  
  
Nur gemeinsam mit dem Hauptschalter `/M`.

`infile.bas` Dateiname eines zu kompilierenden Quelltextes. Der Pfad ist optional.

`/E<event>` Ereignisquelle für den Prozess wählen:  
`/ET` Timer; Default  
`/EE` Extern; erfordert die Optionen `/EEA`, `/EEM`,  
`/EEV`, `/EEO`.  
`/EN` Ungesteuert.

`/PN<#>` Setzt die Prozessnummer (1...4) der zu kompilierenden Datei; Default = 1.

`/P<prio>` Setzt die Priorität des Prozesses auf:  
`/PH` hoch (high); Default  
`/PL` niedrig (low).

`/PD<cycle>` Zykluszeit (Processdelay) des Prozesses auf cycle einstellen. Default: 3000.

`/O<level>` Legt die Optimierungsstufe fest. Die Optimierung kann die Ausführungszeit verkürzen, aber in Ausnahmefällen auch Laufzeitfehler verursachen.  
`/O0` Optimierungsstufe 0 (=keine Optimierung)  
`/O1` Optimierungsstufe 1 (Default)  
`/O2` Optimierungsstufe 2

`/V<#>` Setzt die Versionsnummer des Prozesses; Default = 1.

#### 4.4 Beispiele für ADbasic



```
/opt/adwin/bin/adbasic /L /home/user/test.bas
```

Der Aufruf kompiliert die Quelltextdatei `test.bas` und erzeugt im Verzeichnis `/home/user/` die Library-Datei `test.li9`.

Da außer `/L` keine weiteren Schalter angegeben sind, werden die Standardeinstellungen verwendet:

- Prozessor: T9
- System: Gold
- Prozessnummer: 1
- Event: Timer
- Initial Processdelay: 1000
- Priorität: Hoch (high)
- Include/Pfad: `/opt/adwin/share/inc/`
- Library/Pfad: `/opt/adwin/share/lib/`
- Optimierungsstufe: 1
- Sprache: Englisch

Wenn Sie das Verzeichnis, in dem sich der Compiler `adbasic` befindet, in Ihren Suchpfad aufnehmen (siehe Installation, [Seite 4](#)), können Sie obige Zeile kürzer schreiben als:

```
$ adbasic /L /home/user/test.bas
```

Wir empfehlen in jedem Fall – speziell für eine Automatisierung des Aufrufs – die vollständige Variante:

```
adbasic /L test.bas /A"test" /LE /SG /P9 /O1
```

In den folgenden Beispielen nehmen wir an, dass sich der Pfad des Compilers `adbasic` im Suchpfad befindet.



```
adbasic /L string.bas /SP /O1
```

Der Kommandozeilenaufruf kompiliert die Quelltextdatei `string.bas` mit der Optimierungsstufe 1 in eine Library-Datei für ein Pro-System mit Prozessor T9.

Der gleiche Aufruf, nur für den Prozessor T10, sieht so aus:

```
adbasic /L string.bas /P10 /S1 /O1
```



```
adbasic /home/user/test.bas  
/opt/adwin/share/example/bas_dmo6f /SG /P9
```

Kompiliert die Demo-Datei `bas_dmo6f.bas` in eine Binär-Datei für ein *ADwin-Gold*-System mit Prozessor T9.



```
adbasic /opt/adwin/share/example/bas_dmo6 /SC /P8
```

Kompiliert die Demo-Datei `bas_dmo6.bas` in eine Binär-Datei für eine *ADwin-Karte* mit dem Prozessor T8.



```
adbasic /home/user/my_file.bas /SPII /P11 /Ayour_file
```

Die Anweisung kompiliert die Datei `my_file.bas` für ein *ADwin-Pro II*-System mit Prozessor T11. Die erzeugte Binärdatei hat den Namen `your_file.tb1` und befindet sich im aktuellen Verzeichnis.



```
adbasic /home/user/my_file.bas /A/somewhere/your_file
```

Die Binärdatei heißt nun `your_file.t91` und befindet sich im Verzeichnis `/somewhere`.

### 5 Methoden und Funktionen von ADwin Linux

Die Syntax der Methoden und Funktionen aus der Bibliothek ADwin Linux ist abhängig von der jeweiligen Programmiersprache.

Aus diesem Grund haben wir im folgenden eine allgemeine Syntax zur Beschreibung verwendet. Verwenden Sie die für Ihre Programmiersprache jeweils richtige Syntax.

Die Befehlssyntax ist in folgender Weise dargestellt:

```
Datentyp Befehl (Datentyp Var1, Datentyp Var2, ...)
```

#### Bedeutung

<b>Datentyp</b>	Bei einem Befehl: Datentyp des Rückgabewerts. Bei einem Parameter: Datentyp des Parameters. Mögliche Datentypen sind: void, int32, float, double, char
<b>Befehl</b>	In manchen Programmiersprachen (C, C++) müssen die Befehlsnamen zwingend in der angegebenen Groß- und Kleinschreibung verwendet werden.
<b>Var1, Var2</b>	Variablen und Rückgabewerte werden unter „Parameter“ nach der Befehlssyntax beschrieben.

Befehle zum Ansprechen analoger und digitaler Ein- und Ausgänge (und anderer Hardware-Funktionalitäten) sind in der Bibliothek ADwin Linux nicht enthalten. Sie können solche Anwendungen in ADbasic programmieren.



#### 5.1 Systemsteuerung und -information

Initialisierung des ADwin-Systems und Informationen über den Betriebszustand.

Die Funktion `Set_DeviceNo` setzt die „Device No.“ des ADwin-Systems für alle folgenden Funktionen.

```
void Set_DeviceNo (int32_t DeviceNo)
```

#### Parameter

<i>DeviceNo</i>	Device No. des Systems. Typische Device No. sind 336 (= 0x150 hexadezimal) oder 400 (= 0x190 hexadezimal).
-----------------	--

#### Bemerkungen

ADwin-Systeme werden vom PC über die sogenannte „Device No.“ unterschieden und angesprochen.

Die Funktion `Set_DeviceNo` kommuniziert nicht mit dem ADwin-System und hat daher keinen Einfluss auf die Fehlerbehandlung.

#### Beispiel

```
Set_DeviceNo(3)
```

Die Device No. ist auf 0x3 gesetzt. Alle weiteren Befehle (bis zu einem neuen `Set_DeviceNo`) beziehen sich auf dieses Gerät.

**Set\_DeviceNo**



**Boot**

Boot initialisiert das ADwin-System und lädt die Betriebssystem-Datei dorthin.

```
void Boot(const char *Filename)
```

**Parameter**

*Filename*      Zeiger auf den Dateinamen der Betriebssystem-Datei

**Bemerkungen**

Die Initialisierung löscht alle Prozesse auf dem System und setzt alle globalen Variablen auf den Wert 0.

Das zu ladende Betriebssystem ist prozessorabhängig. Die folgende Tabelle zeigt die Dateinamen für die verschiedenen Prozessoren.

Kurzname	Betriebssystem-Datei
T2	ADwin2.btl
T4	ADwin4.btl
T5	ADwin5.btl
T8	ADwin8.btl
T9	ADwin9.btl
T10	ADwin10.btl
T11	ADwin11.btl
T12	ADwin12.btl
T121	ADwin121.btl

Der PC kann erst mit dem ADwin-System kommunizieren, nachdem Sie das Betriebssystem geladen haben. Laden Sie das Betriebssystem nach jedem Aus- und Einschalten des ADwin-Systems neu.

Das erfolgreiche Laden des Betriebssystems mit `Boot` nimmt ca. 1 Sekunde in Anspruch.

**Test\_Version**

`Test_Version` prüft, ob Treiberversion und Betriebssystem des Prozessors zueinander passen, und ob der Prozessor ansprechbar ist.

```
int32_t Test_Version()
```

**Parameter**

Rückgabewert    0 : OK  
                   1 : Falsche Treiberversion, Prozessor läuft weiter  
                   2 : Falsche Treiberversion, Prozessor wird gestoppt  
                   3 : Keine Antwort vom ADwin-System

`Processor_Type` gibt den Prozessortyp des Systems zurück.

```
int32_t Processor_Type()
```

### Parameter

Rückgabewert    Prozessor-Kennziffer des Systems

### Bemerkungen

Die Funktion liefert für Prüfzwecke den verwendeten Prozessortyp entsprechend der nachstehenden Tabelle.

Rückgabewert	Prozessortyp	Rückgabewert	Prozessortyp
0	Fehler	9	T9
2	T2	1010	T10
4	T4	1011	T11
5	T5	1012	T12
8	T8	10121	T121

`Workload` gibt die Prozessor-Auslastung zurück.

```
int32_t Workload(int Priority)
```

### Parameter

*Priority*        0 (Null): aktuelle Gesamtauslastung des Prozessors  
                   ≠0 : wird derzeit noch nicht unterstützt

Rückgabewert    Prozessor-Auslastung (in Prozent)

`Free_Mem` ermittelt den auf dem System verfügbaren freien Speicher für verschiedene Speicherarten.

```
int32_t Free_Mem(int32_t Mem_Spec)
```

### Parameter

*Mem\_Spec*        Speicherart  
                   `MEM_ALL` (0): alle Speicherarten gemeinsam; nur T2, T4, T5, T8  
                   `MEM_PM_LOCAL` (1): interner Programmspeicher; T9...T11  
                   `MEM_EM_LOCAL` (2): zusätzlicher interner Speicher; T11  
                   `MEM_DM_LOCAL` (3): interner Datenspeicher; T9...T11  
                   `MEM_DRAM_EXTERN` (4): externer DRAM-Speicher; T9...T11  
                   `MEM_CM` (5): Speicher, der Daten an den Cache liefern kann; nur T12/T12.1.  
                   `MEM_UM` (6): Speicher, der keine Daten an den Cache liefern kann; nur T12/T12.1.

Rückgabewert    Momentaner freier Speicher (in Byte)

### Bemerkungen

Die verschiedenen Speicherbereiche sind im *ADbasic*-Handbuch näher erläutert.

## Processor\_Type

## Workload

## Free\_Mem

## 5.2 Prozess-Steuerung

Befehle zur Steuerung einzelner Prozesse auf dem ADwin-System.

### Load\_Process

Load\_Process lädt die Binärdatei eines Prozesses ins ADwin-System.

```
void Load_Process(const char *Filename)
```

#### Parameter

*Filename*      Zeiger auf den Dateinamen der zu ladenden Binärdatei

#### Bemerkungen

Sie erzeugen eine Binärdatei unter Linux, indem Sie den Compiler `ad-basic` aufrufen (ohne den Schalter `-l`, siehe [Kapitel 4.2 auf Seite 11](#)).

Das Aus- und Einschalten eines Systems löscht geladene Prozesse. Sie müssen deshalb nach dem Einschalten die von Ihnen benötigten Prozesse erneut laden.

Bevor Sie den Prozess ins ADwin-System laden, müssen Sie sicherstellen, dass dort nicht bereits ein Prozess mit der gleichen Prozessnummer läuft. Wenn das doch der Fall ist, müssen Sie den laufenden Prozess zuerst mit `Stop_Process` stoppen.

Wenn Sie mehrmals Prozesse laden, kann es zu einer Speicherfragmentierung kommen. Beachten Sie die entsprechenden Hinweise im Handbuch *ADbasic*.



### Start\_Process

Start\_Process startet einen Prozess.

```
void Start_Process(int32_t ProcessNo)
```

#### Parameter

*ProcessNo*      Nummer des Prozesses (1...10)

### Stop\_Process

Stop\_Process stoppt einen Prozess.

```
void Stop_Process(int32_t ProcessNo)
```

#### Parameter

*ProcessNo*      Nummer des Prozesses (1...12, 15)

#### Bemerkungen

Es kann vorkommen, dass ein Prozess nicht sofort nach Durchführen des Befehls gestoppt wird, sondern etwas später. Sie können den Status des Prozesses mit `Process_Status` abfragen.



---

`Clear_Process` löscht einen Prozess aus dem Speicher.

```
void Clear_Process(int32_t ProcessNo)
```

### Parameter

*ProcessNo*     Nummer des Prozesses (1...12, 15)

### Bemerkungen

Geladene Prozesse belegen Speicherplatz im System. Sie können mit `Clear_Process` Prozesse aus dem Speicher entfernen, um für andere Prozesse mehr Platz zu erhalten.

Ein Prozess, der entfernt werden soll, darf nicht laufen. Stoppen Sie einen laufenden Prozess zuerst mit `Stop_Process` (und stellen Sie ggf. auch das Beenden mit `Process_Status` fest), bevor Sie ihn mit `Clear_Process` aus dem Speicher entfernen.

Auf einem *ADwin*-System sind nach dem Booten des Betriebssystems die internen Prozesse 11, 12 und 15 verfügbar. Wenn Sie diese entfernen möchten, tun Sie dies in folgender Reihenfolge: 15, 12, 11.

Auf Gold- und Pro-Systemen sorgt der Prozess 15 für das Blinken der LED; nach dem Entfernen blinkt die LED nicht mehr.

### Clear\_Process



---

`Process_Status` liefert den Status eines Prozesses.

```
int32_t Process_Status(int32_t ProcessNo)
```

### Parameter

*ProcessNo*     Nummer des Prozesses (1...12, 15)

Rückgabewert    Status des Prozesses

1 : Prozess läuft.

0 : Prozess läuft nicht, d.h. er ist nicht geladen, nicht gestartet oder gestoppt.

-1: Prozess wird gestoppt, d.h. er hat ein `Stop_Process` erhalten, wartet aber noch auf den letzten Event.

### Process\_Status

**Set\_Processdelay**

Set\_Processdelay stellt den Parameter Processdelay für einen Prozess ein.

```
void Set_Processdelay(int32_t ProcessNo,
int32_t Processdelay)
```

**Parameter**

*ProcessNo* Nummer des Prozesses (1...10).  
*Processdelay* Einzustellender Wert für den Parameter Processdelay.

**Bemerkungen**

Der Parameter *Processdelay* steuert die Zeitspanne zwischen zwei Event-Aufrufen eines zeitgesteuerten Prozesses (siehe *ADbasic*-Handbuch).

Die Zeitspanne wird in Zeiteinheiten angegeben, die abhängig sind vom Prozessortyp und der Priorität eines Prozesses.

Prozessortyp	Prozess-Priorität	
	hoch	niedrig
T2, T4, T5, T8	1µs	64µs
T9	0,025µs (=25ns)	100µs
T10	0,025µs (=25ns)	50µs
T11	3,3ns	3,3ns (= 0,003µs)
T12	1ns	1ns
T12.1	1,5ns	1,5ns

**Beispiel (C)**

```
Set_Processdelay(1,2000);
// Bei Prozess 1 wird das Processdelay auf 2000 gesetzt
```

Bei einem hochpriorisierten, zeitgesteuerten Prozess und einem T10-Prozessor wird der Prozess alle 50µs (= 2000 · 25ns) aufgerufen.

**Get\_Processdelay**

Get\_Processdelay gibt den Parameter Processdelay eines Prozesses zurück.

```
int32_t Get_Processdelay(int32_t ProcessNo)
```

**Parameter**

*ProcessNo* Nummer des Prozesses (1...10)  
 Rückgabewert Aktuell eingestellter Wert für *Processdelay*,  
 im Fehlerfall: 255

**Bemerkungen**

Der Parameter *Processdelay* steuert die Zeitspanne zwischen zwei Event-Aufrufen eines zeitgesteuerten Prozesses (siehe *Set\_Processdelay* und *ADbasic*-Handbuch).

**Beispiel (C)**

```
int32_t erg;
erg = Get_Processdelay(1);
// Processdelay des ADbasic-Prozesses 1 abfragen
```

### 5.3 Übertragung von globalen Variablen

Befehle zur Datenübertragung zwischen PC und ADwin-System mit den vordefinierten globalen Variablen PAR\_1 ... PAR\_80 und FPAR\_1 ... FPAR\_80.

#### 5.3.1 Globale Long-Variablen (PAR\_1 ... PAR\_80)

Set\_Par setzt eine globale Long-Variable auf den gewünschten Wert.

```
void Set_Par(int32_t Index, int32_t Value)
```

##### Parameter

<i>Index</i>	Nummer der globalen Long-Variablen (1 ... 80)
<i>Value</i>	Zu setzender Wert für die Long-Variable

Get\_Par gibt den Wert einer globalen Long-Variablen zurück.

```
int32_t Get_Par(int32_t Index)
```

##### Parameter

<i>Index</i>	Nummer der globalen Long-Variablen (1 ... 80)
Rückgabewert	Aktueller Wert der Variablen

Get\_Par\_Block überträgt eine anzugebende Anzahl an globalen Long-Variablen in ein Feld.

```
void Get_Par_Block(int32_t Array[],  
int32_t StartIndex, int32_t Count)
```

##### Parameter

<i>Array</i>	Zeiger auf ein Feld (Feldlänge $\geq$ Count)
<i>StartIndex</i>	Nummer der ersten zu übertragenden Variablen (1...80)
<i>Count</i>	Anzahl der zu übertragenden Variablen (max. 80)

Get\_Par\_All überträgt alle globalen Long-Variablen in ein Feld.

```
void Get_Par_All(int32_t Array[])
```

##### Parameter

<i>Array</i>	Zeiger auf ein Feld (Feldlänge $\geq$ 80)
--------------	---

**Set\_Par**

**Get\_Par**

**Get\_Par\_Block**

**Get\_Par\_All**

### 5.3.2 Globale Float-Variablen (FPar\_1...FPar\_80)

#### Set\_FPar

Set\_FPar setzt eine globale Float-Variable auf den gewünschten Wert.

```
void Set_FPar(int32_t Index, float Value)
```

#### Parameter

<i>Index</i>	Nummer der globalen Float-Variablen (1...80)
<i>Value</i>	Zu setzender Wert für die Float-Variable

#### Get\_FPar

Get\_FPar gibt den Wert einer globalen Float-Variablen zurück.

```
float Get_FPar(int32_t Index)
```

#### Parameter

<i>Index</i>	Nummer der globalen Float-Variablen (1...80)
Rückgabewert	Aktueller Wert der Variablen

#### Get\_FPar\_Block

Get\_FPar\_Block überträgt eine anzugebende Anzahl an globalen Float-Variablen in ein Feld.

```
void Get_FPar_Block(float Array[],  
int32_t StartIndex, int32_t Count)
```

#### Parameter

<i>Array</i>	Zeiger auf ein Feld (Feldlänge $\geq$ Count)
<i>StartIndex</i>	Nummer der ersten zu übertragenden Variablen (1...80)
<i>Count</i>	Anzahl der zu übertragenden Variablen (max. 80)

#### Get\_FPar\_All

Get\_FPar\_All überträgt alle globalen Float-Variablen in ein Feld.

```
void Get_FPar_All(float Array[])
```

#### Parameter

<i>Array</i>	Zeiger auf ein Feld (Feldlänge $\geq$ 80)
--------------	---

`Get_FPar_Block_Double` überträgt eine anzugebende Anzahl an globalen Float-Variablen in ein Double-Feld.

```
void Get_FPar_Block_Double(double Array[],
    int32_t StartIndex, int32_t Count)
```

### Parameter

<code>Array</code>	Zeiger auf ein Feld (Feldlänge $\geq$ <code>Count</code> )
<code>StartIndex</code>	Nummer der ersten zu übertragenden Variablen (1...80)
<code>Count</code>	Anzahl der zu übertragenden Variablen (max. 80)

### Bemerkungen

Diese Funktion wurde für Applikationen entwickelt, die Fließkomma-Datensätze nur mit doppelter Genauigkeit (double precision arrays) verarbeiten können.

Bitte beachten Sie, dass die Daten auf dem ADwin-System immer mit einfacher Genauigkeit (single precision) verarbeitet werden. Sie sollten daher Daten aus dem Feld `Array[]` nur mit einfacher Genauigkeit anzeigen lassen, um Missverständnisse bezüglich der Genauigkeit zu vermeiden.

### Get\_FPar\_Block\_Double



`Get_FPar_All_Double` überträgt alle globalen Float-Variablen in ein Double-Feld.

```
void Get_FPar_All_Double(double Array[])
```

### Parameter

<code>Array</code>	Zeiger auf ein Feld (Feldlänge $\geq$ 80)
--------------------	---

### Bemerkungen

Diese Methode wurde für Applikationen entwickelt, die Fließkomma-Datensätze nur mit doppelter Genauigkeit (double precision arrays) verarbeiten können.

Bitte beachten Sie, dass die Genauigkeit der übertragenen Daten einfach (single precision) ist. Sie sollten daher Daten aus dem Feld `Array[]` nur mit einfacher Genauigkeit anzeigen lassen, um Missverständnisse bezüglich der Genauigkeit zu vermeiden.

### Get\_FPar\_All\_Double



## 5.4 Übertragung von Datenfeldern (Arrays)

Befehle zur Datenübertragung zwischen PC und ADwin-System mit globalen DATA-Feldern (DATA\_1...DATA\_200):

- [Einfache Datenfelder](#)
- [FIFO-Felder](#)
- [Datenfelder mit String-Daten](#)



Achten Sie darauf, dass Sie jedes Feld vor seiner Verwendung im *ADbasic*-Programm mit `DIM` deklarieren müssen (vgl. Handbuch „*ADbasic*“).

### 5.4.1 Einfache Datenfelder

#### Data\_Length

`Data_Length` gibt die in *ADbasic* deklarierte Länge eines Felds vom Typ `LONG` oder `FLOAT` zurück, d.h. die Anzahl der Elemente.

```
int32_t Data_Length(int DataNo)
```

#### Parameter

*DataNo* Nummer des Felds (1...200)

Rückgabewert >0: Deklarierte Länge des Felds (=Anzahl der Elemente)  
 0: Fehler - Feld ist nicht deklariert.  
 -1: Sonstiger Fehler.

#### Bemerkungen

Bei einem *DATA*-Feld vom Typ `STRING` stellen Sie die Länge der Zeichenfolge mit dem Befehl `String_Length` fest.

#### SetData\_Long

`SetData_Long` überträgt Long-Daten vom PC in ein *DATA*-Feld des ADwin-Systems.

```
void SetData_Long(int DataNo, int32_t pc_Array[],  
int32_t Startindex, int32_t Count)
```

#### Parameter

*DataNo* Nummer (1...200) des Zielfelds *DATA\_1* ... *DATA\_200*.

*pc\_Array[]* Zeiger auf das Quellfeld, aus dem Daten übertragen werden.

*StartIndex* Nummer ( $\geq 1$ ) des ersten Elements im Zielfeld, das beschrieben wird.

*Count* Anzahl ( $\geq 1$ ) der zu übertragenden Long-Daten.

`GetData_Long` überträgt Long-Daten aus einem DATA-Feld vom ADwin-System in ein Feld.

```
void GetData_Long(int DataNo, int32_t pc_Array[],  
int32_t Startindex, int32_t Count)
```

### Parameter

<i>DataNo</i>	Nummer (1...200) des Quellfelds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Zielfeld, in das die Daten übertragen werden.
<i>StartIndex</i>	Nummer ( $\geq 1$ ) des ersten Elements im Quellfeld, das übertragen wird.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Long-Daten.

`SetData_Float` überträgt Float-Daten vom PC in ein DATA-Feld des ADwin-Systems.

```
void SetData_Float(int DataNo, float pc_Array[],  
int32_t Startindex, int32_t Count)
```

### Parameter

<i>DataNo</i>	Nummer (1...200) des Zielfelds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Quellfeld, aus dem Daten übertragen werden.
<i>StartIndex</i>	Nummer ( $\geq 1$ ) des ersten Elements im Zielfeld, das beschrieben wird.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Float-Daten.

`GetData_Float` überträgt Float-Daten aus einem DATA-Feld vom ADwin-System in ein Feld.

```
void GetData_Float(int DataNo, float pc_Array[],  
int32_t Startindex, int32_t Count)
```

### Parameter

<i>DataNo</i>	Nummer (1...200) des Quellfelds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Zielfeld, in das die Werte übertragen werden.
<i>StartIndex</i>	Nummer ( $\geq 1$ ) des ersten Elements im Quellfeld, das übertragen wird.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Float-Daten

### GetData\_Long

### SetData\_Float

### GetData\_Float

**SetData\_Double**

SetData\_Double überträgt Double-Daten vom PC mit einfacher Genauigkeit in ein DATA-Feld des ADwin-Systems.

```
void SetData_Double(int DataNo, double pc_Array[],
int32_t Startindex, int32_t Count)
```

**Parameter**

<i>DataNo</i>	Nummer (1...200) des Zielfelds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Quellfeld, aus dem Daten übertragen werden.
<i>StartIndex</i>	Nummer ( $\geq 1$ ) des ersten Elements im Zielfeld, das beschrieben wird.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Double-Daten.

**Bemerkungen**

Diese Methode wurde für Applikationen entwickelt, die Fließkomma-Datensätze nur mit doppelter Genauigkeit (double precision arrays) verarbeiten können.



Bitte beachten Sie, dass die Daten auf dem ADwin-System immer mit einfacher Genauigkeit (single precision) verarbeitet werden. Sie sollten daher Daten aus dem Feld *Data[]* nur mit einfacher Genauigkeit anzeigen lassen, um Missverständnisse bezüglich der Genauigkeit zu vermeiden.

**GetData\_Double**

GetData\_Double überträgt Float-Daten aus einem DATA-Feld vom ADwin-System in ein Double-Feld.

```
void GetData_Double(int DataNo, double pc_Array[],
int32_t Startindex, int32_t Count)
```

**Parameter**

<i>DataNo</i>	Nummer (1...200) des Quellfelds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Zielfeld, in das die Werte übertragen werden.
<i>StartIndex</i>	Nummer ( $\geq 1$ ) des ersten Elements im Quellfeld, das übertragen wird.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Double-Daten

**Bemerkungen**

Diese Methode wurde für Applikationen entwickelt, die Fließkomma-Datensätze nur mit doppelter Genauigkeit (double precision arrays) verarbeiten können.



Bitte beachten Sie, dass die Genauigkeit der übertragenen Daten einfach (single precision) ist. Sie sollten daher Daten aus dem Feld *Data[]* nur mit einfacher Genauigkeit anzeigen lassen, um Missverständnisse bezüglich der Genauigkeit zu vermeiden.



`Data2File` speichert Long- oder Float-Daten aus einem DATA-Feld des ADwin-Systems in einer Datei (auf der Festplatte).

```
void Data2File(const char *Filename, int DataNo,  
int32_t Startindex, int32_t Count, int32_t Mode)
```

### Parameter

<i>Filename</i>	Zeiger auf den Dateinamen
<i>DataNo</i>	Nummer (1...200) des Quellfelds <code>DATA_1</code> ... <code>DATA_200</code> .
<i>Startindex</i>	Nummer ( $\geq 1$ ) des ersten Elements im Quellfeld, das übertragen wird.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten
<i>Mode</i>	0 : Datei wird überschrieben (falls vorhanden) 1 : Daten werden an eine vorhandene Datei angehängt

### Bemerkungen

Das DATA-Feld darf nicht als `FIFO` definiert sein. Die Daten werden binär gespeichert.

Die Daten werden in der Datei binär gespeichert. Wenn die Datei nicht vorhanden ist, wird sie neu angelegt.

### Beispiel (C)

```
Data2File("Test.dat", 1, 1, 1000, 0);  
// Speichert die Elemente 1...1000 aus dem ADbasic-  
// Feld DATA_1 in der Datei Test.dat
```

### Data2File



### 5.4.2 FIFO-Felder

Befehle zur Datenübertragung zwischen PC und ADwin-System mit globalen DATA-Feldern (DATA\_1...DATA\_200), die als FIFO deklariert sind.

Sie müssen jedes FIFO-Feld vor seiner Verwendung unter *ADbasic* deklarieren (vgl. Handbuch „*ADbasic*“): `DIM DATA_x[n] as TYPE as FIFO`

Beim Arbeiten mit FIFO-Feldern sollten Sie als Faustregel nicht mehr Elemente beschreiben als deklariert sind. Auf keinen Fall sollten Sie mehr Elemente auslesen als vorher beschrieben wurden. Vermeiden Sie dies wie folgt:

- Prüfen Sie mit der Funktion `Fifo_Full`, ob ein FIFO-Feld mindestens so viele Elemente enthält, wie Sie mit `GetFifo_X` auslesen möchten.
- Prüfen Sie mit der Funktion `Fifo_Empty`, ob ein FIFO-Feld mindestens so viele freie Elemente enthält, wie mit `SetFifo_X` beschreiben möchten.

#### Fifo\_Empty

`Fifo_Empty` liefert die Anzahl der freien Elemente eines Fifo-Felds.

```
int32_t Fifo_Empty(int FifoNo)
```

#### Parameter

*FifoNo* Nummer (1...200) des FIFO-Felds `DATA_1 ... DATA_200`.  
Rückgabewert Anzahl der freien Elemente im FIFO-Feld.

#### Fifo\_Full

`Fifo_Full` liefert die Anzahl der belegten Elemente eines Fifo-Felds.

```
int32_t Fifo_Full(int FifoNo)
```

#### Parameter

*FifoNo* Nummer (1...200) des FIFO-Felds `DATA_1 ... DATA_200`.  
Rückgabewert Anzahl der belegten Elemente im FIFO-Feld.

#### Fifo\_Clear

`Fifo_Clear` initialisiert den Schreib- und den Lesezeiger eines Fifo-Felds.

```
void Fifo_Clear(int FifoNo)
```

#### Parameter

*FifoNo* Nummer (1...200) des FIFO-Felds `DATA_1 ... DATA_200`.

#### Bemerkungen

Beim Deklarieren eines FIFO-Felds (im *ADbasic*-Programm) werden die FIFO-Zeiger nicht automatisch initialisiert. Rufen Sie deshalb `Fifo_Clear` gleich zu Beginn Ihres Programms auf, entweder in *ADbasic* oder mit dieser Funktion.

Das Initialisieren der FIFO-Zeiger im Programmablauf ist sinnvoll, wenn alle beschriebenen Elemente (z.B. wegen eines Fehlers) verworfen werden sollen.

SetFifo\_Long überträgt Long-Daten aus dem PC in ein Fifo-Feld des ADwin-Systems.

```
void SetFifo_Long(int FifoNo, int32_t pc_Array[],
int32_t Count)
```

### Parameter

<i>FifoNo</i>	Nummer (1...200) des FIFO-Felds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Quellfeld, aus dem Daten übertragen werden.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten.

GetFifo\_Long überträgt Long-Daten aus einem Fifo-Feld vom ADwin-System in den PC.

```
void GetFifo_Long(int FifoNo, int32_t pc_Array[],
int32_t Count)
```

### Parameter

<i>FifoNo</i>	Nummer (1...200) des FIFO-Felds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Zielfeld, in das Daten übertragen werden (Feldlänge $\geq$ <i>Count</i> ).
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten.

SetFifo\_Float überträgt Float-Daten aus dem PC in ein Fifo-Feld des ADwin-Systems.

```
void SetFifo_Float(int FifoNo, float pc_Array[],
int32_t Count)
```

### Parameter

<i>FifoNo</i>	Nummer (1...200) des FIFO-Felds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Quellfeld, aus dem Daten übertragen werden.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten.

GetFifo\_Float überträgt Float-Daten aus einem Fifo-Feld vom ADwin-System in den PC.

```
void GetFifo_Float(int FifoNo, float pc_Array[],
int32_t Count)
```

### Parameter

<i>FifoNo</i>	Nummer (1...200) des FIFO-Felds <a href="#">DATA_1</a> ... <a href="#">DATA_200</a> .
<i>pc_Array[]</i>	Zeiger auf das Zielfeld, in das Daten übertragen werden (Feldlänge $\geq$ <i>Count</i> ).
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten.

## SetFifo\_Long

## GetFifo\_Long

## SetFifo\_Float

## GetFifo\_Float

**SetFifo\_Double**

SetFifo\_Double überträgt Double-Daten aus dem PC mit einfacher Genauigkeit in ein Fifo-Feld des ADwin-Systems.

```
void SetFifo_Double(int FifoNo, double pc_Array[],
int32_t Count)
```

**Parameter**

<i>FifoNo</i>	Nummer (1...200) des FIFO-Felds <i>DATA_1</i> ... <i>DATA_200</i> .
<i>pc_Array[]</i>	Zeiger auf das Quellfeld, aus dem Daten übertragen werden.
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten.

**Bemerkungen**

Diese Methode wurde für Applikationen entwickelt, die Fließkomma-Datensätze nur mit doppelter Genauigkeit (double precision arrays) verarbeiten können.

Bitte beachten Sie, dass die Daten auf dem ADwin-System immer mit einfacher Genauigkeit (single precision) verarbeitet werden. Sie sollten daher Daten aus dem Feld *Data[]* nur mit einfacher Genauigkeit anzeigen lassen, um Missverständnisse bezüglich der Genauigkeit zu vermeiden.

**GetFifo\_Double**

GetFifo\_Double überträgt Float-Daten aus einem Fifo-Feld vom ADwin-System in ein Double-Feld im PC.

```
void GetFifo_Double(int FifoNo, double pc_Array[],
int32_t Count)
```

**Parameter**

<i>FifoNo</i>	Nummer (1...200) des FIFO-Felds <i>DATA_1</i> ... <i>DATA_200</i> .
<i>pc_Array[]</i>	Zeiger auf das Zielfeld, in das Daten übertragen werden (Feldlänge $\geq$ <i>Count</i> ).
<i>Count</i>	Anzahl ( $\geq 1$ ) der zu übertragenden Daten.

**Bemerkungen**

Diese Methode wurde für Applikationen entwickelt, die Fließkomma-Datensätze nur mit doppelter Genauigkeit verarbeiten.

Bitte beachten Sie, dass die Genauigkeit der übertragenen Daten einfach ist. Sie sollten daher Daten aus dem Feld *Data[]* mit einfacher Genauigkeit anzeigen, um Missverständnisse bzgl. der Genauigkeit zu vermeiden.



### 5.4.3 Datenfelder mit String-Daten

Befehle zur Datenübertragung zwischen PC und ADwin-System mit globalen DATA-Feldern (DATA\_1...DATA\_200), die String-Daten enthalten. Das DATA-Feld muss in *ADbasic* mit DIM ... AS STRING deklariert werden.

String\_Length gibt die Länge eines Datenstrings in einem DATA-Feld zurück.

```
int32_t String_Length(int DataNo)
```

#### Parameter

*DataNo* Nummer (1...200) des Felds *DATA\_1* ... *DATA\_200*.

Rückgabewert Länge des Strings = Anzahl der Zeichen.

#### Bemerkungen

String\_Length zählt die Zeichen im DATA-Feld bis zum ersten Auftreten der Endekennung (ASCII-Nummer 0). Die Endekennung selbst wird nicht als Zeichen gezählt.

#### Beispiel (C)

```
int32_t erg;
erg = String_Length (2);
// Ermittelt die Länge des Strings in DATA_2.
```

SetData\_String überträgt einen String in ein DATA-Feld.

```
void SetData_String(int DataNo, char *String)
```

#### Parameter

*DataNo* Nummer (1...200) des Felds *DATA\_1* ... *DATA\_200*.

*String* Zeiger auf den zu übertragenden String oder Zeichenkette in einfachen Hochkommata.

#### Bemerkungen

Jedem übertragenen String wird zusätzlich als letztes Zeichen die Endekennung (ASCII-Nummer 0) angehängt.

Beachten Sie den unterschiedlichen Umgang der Programmiersprachen mit der Endekennung, wenn diese im zu übertragenden String enthalten ist.

Beispielsweise übertragen *ADbasic* und C den String „Hello\0 world“ nur bis zur Endekennung, also „Hello“. Dagegen überträgt beispielsweise Delphi (Kylix) die vollständige Zeichenfolge.

#### Beispiel (C)

```
SetData_String (2, "Dies ist ein Text" );
// Der String "Dies ist ein Text" wird in das Feld
// DATA_2 geschrieben und die Endekennung angehängt.
```

### String\_Length

### SetData\_String



**GetData\_String**

GetData\_String überträgt einen String aus einem DATA-Feld zum PC.

```
int GetData_String(int DataNo, char *String,
int32_t MaxCount)
```

**Parameter**

<i>DataNo</i>	Nummer des DATA-Felds (1...200)
<i>String</i>	Zeiger auf das Feld, in das der String geschrieben werden soll (Feldlänge $\geq$ <i>MaxCount</i> +1).
<i>MaxCount</i>	Max. Anzahl ( $\geq 1$ ) der übertragenen Zeichen ohne Endeckennung.
Rückgabewert	Anzahl der gelesenen Zeichen (ohne Endeckennung).

**Bemerkungen**

Nach dem Schreiben der Zeichen in das Feld *String* wird zusätzlich als letztes Zeichen die Endeckennung (ASCII-Nummer 0) angehängt, so dass der String insgesamt *MaxCount*+1 Zeichen enthält.

Wenn der String im *DATA*-Feld eine Endeckennung enthält, stoppt die Übertragung genau dort, d.h. die Endeckennung wird nicht übertragen. Die Anzahl der bis dahin gelesenen Zeichen ohne die Endeckennung ist der Rückgabewert.

Wenn *MaxCount* größer ist als die in *ADbasic* definierte Zeichenzahl des Strings, erhalten Sie über *Get\_Last\_Error()* den Fehler "Data too small".

Wenn Sie einen großen Wert für *MaxCount* angeben, hat die Funktion eine entsprechend lange Ausführungszeit, selbst wenn der übertragene String nur kurz ist.

Bei zeitkritischen Anwendungen mit großen Strings kann es günstiger sein, wie folgt vorzugehen:

- Sie stellen die tatsächliche Anzahl der Zeichen im String mit *String\_Length()* fest.
- Sie lesen den String mit *Getdata\_String()* und übergeben die tatsächliche Zeichnanzahl als *MaxCount*.

**Beispiel (C)**

```
char ArrayString[101];
GetData_String (2, ArrayString, 100);
// einen String mit 100 Elementen aus DATA_2 holen
```

Enthält das *DATA*-Feld im *ADwin*-System z. B. an Position 9 eine Endeckennung, so werden 8 Zeichen gelesen.



### 5.5 Steuerung und Fehlerbehandlung

`Get_Last_Error` gibt die Fehlernummer zurück, die sich auf den zuletzt an das ADwin-System gesendeten Befehl bezieht.

```
int32_t Get_Last_Error()
```

#### Parameter

Rückgabewert  $\neq 0$ : Nummer des zuletzt aufgetretenen Fehlers (siehe [Anhang A.1](#)).  
0 : kein Fehler aufgetreten

#### Bemerkungen

Die zurückgegebene Fehlernummer bezieht sich immer auf

- den letzten Befehl, der an das ADwin-System gesendet wurde (die meisten, aber nicht alle Befehle gehören zu dieser Gruppe).
- das Programm, das den Befehl gesendet hat.

Nachdem die Funktion ausgeführt wurde, wird die Fehlernummer auf 0 (Null) zurückgesetzt.

#### Beispiel (C)

```
int32_t Error;  
Error = Get_Last_Error();
```

`Get_Last_Error_Text` gibt einen Fehlertext zu einer vorhandenen Fehlernummer zurück.

```
char *Get_Last_Error_Text(int32_t Errno)
```

#### Parameter

*Errno*            Fehlernummer  
Rückgabewert    Zeiger auf Fehlertext

#### Bemerkungen

Diese Funktion wird üblicherweise mit dem Rückgabewert von `Get_Last_Error` aufgerufen.

#### Beispiel (C)

```
int32_t Error = Get_Last_Error();  
printf („Get_Last_Error:%d\n“, Error);  
printf („Last_Error_Text:%s\n“,  
      Get_Last_Error_Text(Error));
```

#### Get\_Last\_Error

#### Get\_Last\_Error\_Text

**Get\_Known\_DeviceNo**

Get\_Known\_DeviceNo ist eine 2stufige Funktion: Sie ermittelt entweder die Anzahl der konfigurierten **ADwin**-Systeme oder sie gibt die Gerätenummern dieser Systeme zurück. Das Ergebnis des 1. Aufrufs dient dazu, die Feldgröße für den 2. Aufruf einzurichten.

```
int16_t Get_Known_DeviceNo (int32_t Devices[],
                           int32_t *Count_Devices)
```

**1. Aufruf:** Anzahl der **ADwin**-Systeme ermitteln

*Devices* = NULL

*Count\_Devices* Aufruf : Zeiger auf eine Long-Variable mit Wert 0.  
Rückgabewert: Anzahl ( $\geq 1$ ) der in *ADconfig* konfigurierten **ADwin**-Systeme.

Rückgabewert ohne Funktion

**2. Aufruf:** Gerätenummern der Systeme ermitteln

*Devices* Aufruf: Zeiger auf ein Feld.  
Rückgabewert: Gerätenummern der (in *ADconfig* konfigurierten) Systeme.

*Count\_Devices* Aufruf: Zahl der tatsächlich konfigurierten Systeme.  
Rückgabewert: Anzahl ( $\geq 1$ ) der in *ADconfig* konfigurierten **ADwin**-Systeme (s.o.).

Rückgabewert 0 : Kein Fehler; *Count\_Devices* und *Devices* enthalten die gewünschten Informationen.  
1 : Fehler, d.h. *Count\_Devices* wurde kleiner angegeben als die tatsächliche Zahl der konfigurierten Systeme. *Count\_Devices* und *Devices* sind unverändert.

**Bemerkungen**

Verwenden Sie die Funktion wie vorgesehen in 2 Stufen:

1. Ermitteln Sie mit *Get\_Known\_DeviceNo* die Anzahl der in *ADconfig* konfigurierten Systeme (*Count\_Devices*).

2. Dimensionieren Sie das Feld *Devices* mit dem Wert aus *Count\_Devices*.

3. Rufen Sie *Get\_Known\_DeviceNo* erneut auf, jetzt aber übergeben Sie das Feld und die Anzahl der Systeme. Sie erhalten die Device No. der **ADwin**-Systeme.

Beachten Sie, dass die Aufrufparameter die korrekten Werte für die jeweilige Funktionsstufe enthalten müssen.

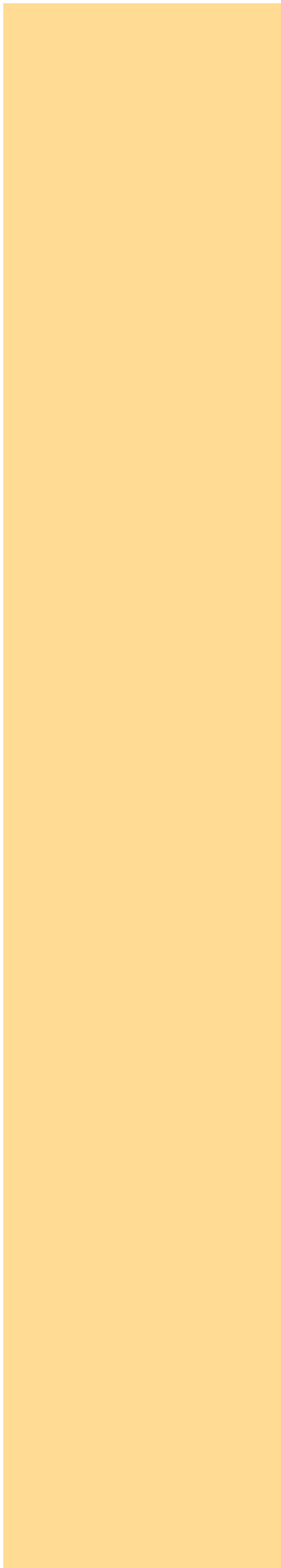
**Beispiel (C)**

```
int32_t *devices;
int32_t count_devices;
int32_t error;

// 1. Aufruf mit Übergabe von NULL
// Ermittelt die Anzahl der in ADconfig konfigurierten
// Geräte und gibt sie in count_devices zurück.
count_devices = 0;
Get_Known_DeviceNo(NULL, &count_devices);
// Feld devices zum Eintragen der Gerätenr. einrichten
// (=Zuweisen des benötigten Speichers)
devices = new int32_t[count_devices];
```



```
// 2. Aufruf mit Übergabe eines Felds
// Abfrage der Gerätenummern und Eintrag in devices
error = Get_Known_DeviceNo(devices, &count_devices);
```



## Anhang

### A.1 Fehlermeldungen

Fehler-Nr.	Fehlermeldung (nur englisch möglich)
0	No Error.
1	Timeout error on writing to the ADwin-system.
2	Timeout error on reading from the ADwin-system.
10	The device No. is not allowed.
11	The device No. is not known.
15	Function for this device not allowed.
20	Incompatible versions of ADwin operating system , driver (ADwin32.DLL) and/or ADbasic binary-file.
100	The Data is too small.
101	The Fifo is too small or not enough values.
102	The Fifo has not enough values.
150	Not enough memory or memory access error.
200	File not found.
201	A temporary file could not be created.
202	The file is not an ADBasic binary-file.
203	The file is not valid.
204	The file is not a BTL.
2000	Network error (Tcplp).
2001	Network timeout.
2002	Wrong password.
3001	Device is unknown.

**A.2 Index der Methoden und Properties****B**[Boot · 18](#)**C**[Clear\\_Process · 21](#)**D**[Data\\_Length · 26](#)[Data2File · 29](#)**F**[Fifo\\_Clear · 30](#)[Fifo\\_Empty · 30](#)[Fifo\\_Full · 30](#)[Free\\_Mem · 19](#)**G**[Get\\_FPar · 24](#)[Get\\_FPar\\_All · 24](#)[Get\\_FPar\\_All\\_Double · 25](#)[Get\\_FPar\\_Block · 24](#)[Get\\_FPar\\_Block\\_Double · 25](#)[Get\\_Known\\_DeviceNo · 36](#)[Get\\_Last\\_Error · 35](#)[Get\\_Last\\_Error\\_Text · 35](#)[Get\\_Par · 23](#)[Get\\_Par\\_All · 23](#)[Get\\_Par\\_Block · 23](#)[Get\\_Processdelay · 22](#)[GetData\\_Double · 28](#)[GetData\\_Float · 27](#)[GetData\\_Long · 27](#)[GetData\\_String · 34](#)[GetFifo\\_Double · 32](#)[GetFifo\\_Float · 31](#)[GetFifo\\_Long · 31](#)**L**[Load\\_Process · 20](#)**P**[Process\\_Status · 21](#)[Processor\\_Type · 19](#)**S**[Set\\_DeviceNo · 17](#)[Set\\_FPar · 24](#)[Set\\_Par · 23](#)[Set\\_Processdelay · 22](#)[SetData\\_Double · 28](#)[SetData\\_Float · 27](#)[SetData\\_Long · 26](#)[SetData\\_String · 33](#)[SetFifo\\_Double · 32](#)[SetFifo\\_Float · 31](#)[SetFifo\\_Long · 31](#)[Start\\_Process · 20](#)[Stop\\_Process · 20](#)[String\\_Length · 33](#)**T**[Test\\_Version · 18](#)**W**[Workload · 19](#)