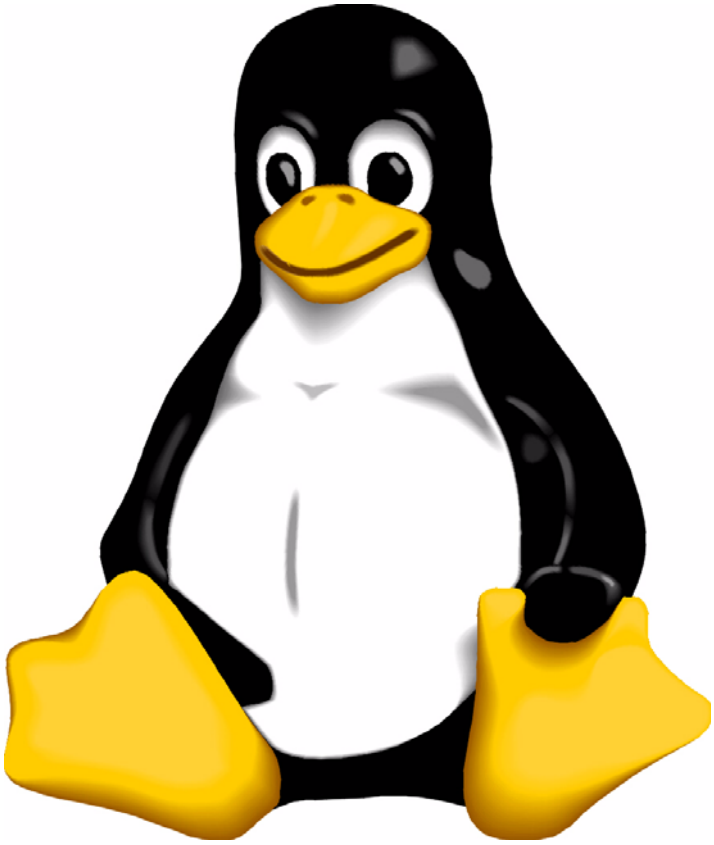
ADwin for Linux

Manual



ADwin driver Linux, Manual Jan. 2020

For any questions, please don't hesitate to contact us:

Hotline:	+49 6251 96320
Fax:	+49 6251 5 68 19
E-Mail:	info@ADwin.de
Internet	www.ADwin.de

JÄGER
Computergesteuerte
Messtechnik GmbH

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

Table of contents

Typographical Conventions

"Warning" stands for information, which indicate damages of hardware or software, test setup or injury to persons caused by incorrect handling.

You find a "note" next to

– information, which absolutely have to be considered in order to guarantee an error free operation.
– advice for efficient operation.

"Information" refers to further information in this documentation or to other sources such as manuals, data sheets, literature, etc.

<C:\ADwin\ …>

File names and paths are placed in <angle brackets> and characterized in the font `Courier New`.

Program text

Program commands and user inputs are characterized by the font `Courier New`.

Var_1

Source code elements such as commands, variables, comments and other text are characterized by the font `Courier New` and are printed in color.

Bits in data (here: 16 bit) are referred to as follows:

Bit No.	15	14	13	…	01	00
Bit value	2^{15}	2^{14}	2^{13}	…	2^1=2	2^0=1
Synonym	MSB	-	-	-	-	LSB

1 Information about this Manual

This manual contains information about using the interface library *ADwin* Linux as well as the *ADbasic* and the *TiCoBasic* compiler.

Additional information is available in

– the manual "*ADwin* Installation", which describes all interface installations for the *ADwin* systems.

– the manuals "*ADbasic*" and "*TiCoBasic*", which contain all instructions for the compilers. You will also find information about
  • the structure of *ADbasic* / *TiCoBasic* programs
  • the optimization of *ADbasic* / *TiCoBasic* programs
  • the run-time behaviour of processes

  A comfortable user interface as real-time development tool is available for your programming tasks – up to this time only for Windows. You may also use an editor of your choice.

– *ADwin-Pro* systems only: Software manual with the instructions for programming the Pro modules.

– the hardware manual for the *ADwin* system you are using.

– Manuals of *ADwin* drivers for program packages and programming languages.

Knowledge in handling the development environment or programming language is assumed.

Please note:

For *ADwin* systems to function correctly, follow strictly the information provided in this documentation and in other mentioned manuals.

Qualified personnel

Programming, start-up and operation, as well as the modification of program parameters must be performed only by appropriately qualified personnel.

> *Qualified personnel are persons who, due to their education, experience and training as well as their knowledge of applicable technical standards, guidelines, accident prevention regulations and operating conditions, have been authorized by a quality assurance representative at the site to perform the necessary acivities, while recognizing and avoiding any possible dangers.*
> *(Definition of qualified personnel as per VDE 105 and ICE 364).*

Availability of the documents

This product documentation and all documents referred to, have always to be available and to be strictly observed. For damages caused by disregarding the information in this documentation or in all other additional documentations, no liability is assumed by the company *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, Germany.

Legal information

OEM products are mentioned without referring to possible patent rights, the existence of which may not be excluded.

Subject to change.

Hotline address: see inner side of cover page.

2 *ADwin* Linux

ADwin Linux is a library. It provides interface functions, so that development environments, such as C, C++, Matlab, or LabView can communicate with the *ADwin* system.

The library *ADwin* Linux defines methods, functions and system variables for process control and data transfer from and to the *ADwin* system. The definition includes the name of a function as well as number, data type and order of the parameters to be transferred.

Communication with the *ADwin* system

With the development environment, you can control processes in the *ADwin* system, get data from there or send data to the system. The processes are programmed in *ADbasic* (see *ADbasic* manual) and compiled with the compiler, which is provided as standard delivery item (see chapter 4).

Real-time operating system

The library functions communicate with the real-time operating system of the *ADwin* system. Therefore, the operating system must be loaded after each power-up (e.g. the file `<adwin9.btl>`). After a successful loading the system will be able to receive and execute processes, receive instructions from the PC and exchange data with it. The processes programmed in *ADbasic*, include the program code for measurement, open-loop or closed-loop control of your application.

The real-time operating system performs the following tasks:

10 processes

– Management of up to 10 real-time processes with low or high priority (selectable). Processes with low priority can be interrupted by processes with high priority, the latter cannot be interrupted by other processes.

Data memory

– Providing global variables:
  - 80 integer variables (`PAR_1` ... `PAR_80`), already specified
  - 80 float variables (`FPAR_1` ... `FPAR_80`), already specified
  - 200 data arrays of arbitrary length (`DATA_1`...`DATA_200`), user specified

  The values of these variables or data arrays can be changed or read from the development environment at any time. Local variables and arrays can be defined.

Communication

– Communication between *ADwin* system and PC.

  The communication process is running with medium priority on the *ADwin* system and can interrupt low-priority processes for a short time. It interprets and processes all instructions, which are sent from the PC to the *ADwin* system: Control instructions and instructions for data exchange. The following table shows examples for each group.

Control instructions, e.g.	
`LOAD_PROCESS`	transfers a process to the system
`START_PROCESS`	starts a process
Instructions for data exchange, e.g.	
`GET_PAR`	returns the current value of a parameter
`SET_PAR`	changes the value of a parameter
`GETDATA_LONG`	returns the values of a `DATA` array

The communication process never sends data to the PC without being requested to do so. This assures that data is transferred to the PC only if you have requested this data.

3 Installing and Initializing *ADwin* Linux

For installation of the library the following is required:

– Kernel since 2.6 for *ADwin* systems with Ethernet connection.

– Build environment according to the Linux distribution
  • Debian/Ubuntu: `build-essential`
  • Red Hat (Fedora, CentOS): `@development-tools`
  • openSUSE: `devel_C_C++`
  • solus: `system.devel`

– The following Linux packages: `cmake`, `bison`, `flex`, `gengetopt`

– Wine since version 1.4 for *ADbasic* and *TiCoBasic*.

When using an older *ADwin* system with Link data connection or with MAC OS, this version of the *ADwin* Linux library is unsuitable. Instead, use the previous version 5.00 in connection with Linux kernel 2.4.

The installation contains the following steps:

– Installing wine
– Software Configuration and Installation
– Configuring the ADwin System
– Including the Library in C

For parts of the installation "root" privileges are necessary. If root privileges are required, the # sign is indicated at the beginning of an edit line, otherwise the `$` sign.

With some Linux distributions, the root account is disabled for reasons of safety. In this case, call the instruction via the program `sudo`, e.g. `sudo make install`.

3.1 Installing wine

For installation of Wine, please refer to the help for installation of additional packages for your distribution. Some distributions allow the separate installation of Wine for 32-bit and 64-bit Windows emulation. Since *ADbasic* and *TiCoBasic* are plain 32-bit applications, do select the package `wine32` anyway not regarding whether your Linux system is 32-bit or 64-bit.

Under Ubuntu normally this will do

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

Under Debian, the 32-bit support must be explicitely enabled:

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

3.2 Installing ADwin software

The `./LINUX` directory with the following zip archives can be found in the *ADwin* software package:

ADwin software package

– `adwin-lib-x.y.zip`: The archive (version `x.y`) contains:
  • *ADwin* Linux library (shared library).
  • Configuration program `adconfig`.

– `adwin-compiler-x.y.zip`: The *ADbasic* compiler.

– `adwin-labview.zip:` The driver for the LabView development environment (complete installation see manual "LabView Driver").

– `adwin-matlab.zip:` The driver for the MATLAB environment (complete installation see manual "Matlab Driver").

– `adwin-doc-x.y.zip`: All documentations for software and hardware as pdf format.

Installing the archives

Install archives in the same order as indicated above. Start with `adwin-lib-x.y.zip` (see notes below).

To install an archive, proceed as follows:

1. Extract the archive `<filename>.zip`.
2. Go on with the following steps:
```
$ cd adwin-lib-x.y
$ mkdir build -p
$ cd build
$ cmake .. -DCMAKE_INSTALL_PREFIX=/opt/adwin
$ cmake --build .
# make install
```
   With the prefix `DCMAKE_INSTALL_PREFIX`, the files are installed into the directory `/opt/adwin/` We assume this configuration in the following text. The directory is written to the file `/etc/adwin/ADWINDIR`.
3. Install the other archives in the same way.

   The prefix can be omitted here, the directory path is read from the file `/etc/adwin/ADWINDIR`.

   In the end, there is a directory structure as shown below (see page 5).

Environment variable

In earlier versions of the Linux library, the installation directory was stored in the environment variable `ADWINDIR`. This is possible further on, but not required.

To set the variable, you may either
  • indicate the directory (see `configure`) directly:
```
$ export ADWINDIR=/opt/adwin/
```
  • or use a file (automatically generated from `make` / `make install`):
```
$ export ADWINDIR=´cat /etc/adwin/ADWINDIR/´
```

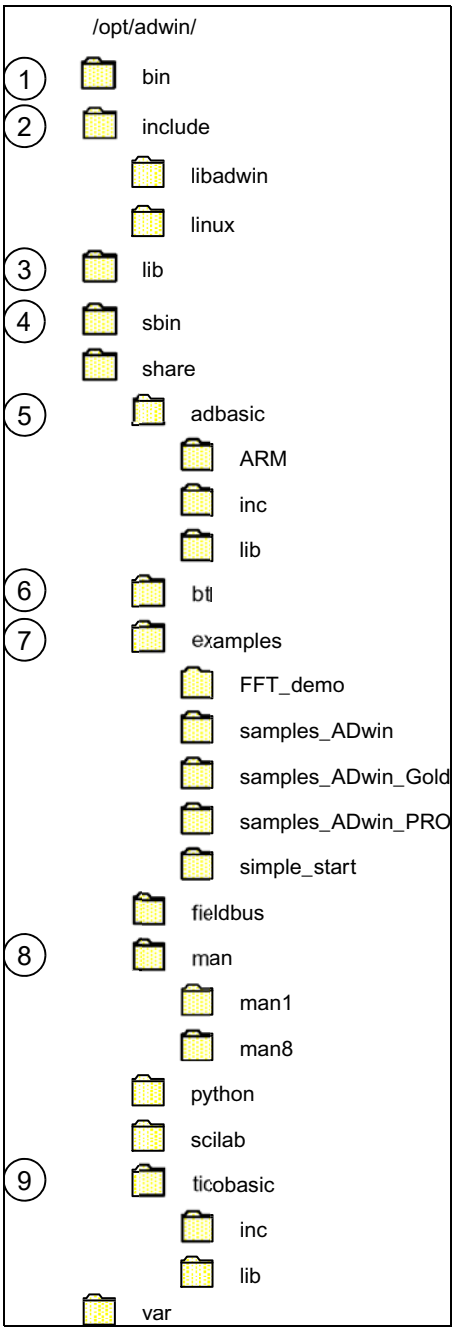
If you use `bash`, the following lines may be useful to add:
```
$ export PATH=$PATH:$ADWINDIR/bin:$ADWINDIR/sbin
$ export MANPATH=$MANPATH:$ADWINDIR/man
```

Alternatively, you can set the `MANPATH` in the file `/etc/manpath.config` for the whole system.

After installation the structure of directories shown to the right is given.

The most important files and programs are:

1. *ADbasic* and *TiCoBasic* compiler.
2. Include files for program development.
3. ADwin program library.
4. The program *ADconfig*, which applies a device no. to an *ADwin* system.
5. Include and library files for the *ADbasic* compiler.
6. Operating system files `*.btl` for *ADwin* processors.
7. *ADbasic* examples, sorted by *ADwin* systems.
8. Manuals
   - `man1`: ADbasic and TiCoBasic compiler
   - `man8`: ADconfig
9. Include and library files for the *TiCoBasic* compiler.



Installed structure of directories

3.3 Installing Cross Compiler for T12 / T12.1

With *ADwin* processors T12 and T12.1, the *ADbasic* compiler creates C code which subsequently is automatically translated into *ADbasic* processes by a einem cross compiler. Under Windows, this task is done with the Linaro GCC compiler `arm-linux-gnueabihf-4.9-2014.09`, which you can also install under Linux manually. However we have gained positive experience to simply use the most current cross compiler of the distribution.

You install the C compiler this way:

– Debian and Ubuntu

```
# apt-get install gcc-arm-linux-gnueabihf
```

– RedHat based distributions like Fedora, CentOS

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

– openSUSE

  Get the package `linar-toolchain` gfrom the address https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/arm-linux-gnueabihf/

  Then create links to all binaries in `/usr/bin`:
  ```
  # ln -s [BINARY-DOWNLOAD]/bin/arm-linux-gnueabihf* /usr/bin/
  ```

  Set the environment variable
  ```
  CROSS_COMPILE=arm-linux-gnueabihf-
  ```

3.4 Configuring the *ADwin* System

Each *ADwin* system is accessed under Linux via a "device no." To configure the device number use the program `adconfig` in the directory `/opt/adwin/sbin/`:

– Add the device no. of the *ADwin* system under Linux (there is no default device no. after a fresh installation). The PC uses the log-on data to access the *ADwin* system.

– If the *ADwin* system has an Ethernet interface, you must configure the system, too (see page 8).

  The configuration data are not stored on the PC, but are transferred to the *ADwin* system. The data must correspond to the log-on data under Linux .

  In the "ADwin Installation" manual, you will find a description about the basics of Ethernet operation.

The `adconfig` program requires root privileges (`su`).

3.4.1 Example: Standard configuration

The following lines explain how to configure the device no. 0x150 for an *ADwin* system with Ethernet interface:

1. Add the device no. 0x150 (decimal: 336) with a fixed IP address (here: 10.100.100.83):
   ```
   # adconfig add 0x150 TYPE net IP 10.100.100.83
   ```
2. Display the results of adding the device no.
   ```
   # 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. Configure the *ADwin* system with MAC address, IP address and subnet mask:
   ```
   # adconfig config 00:50:C2:0A:22:DD IP 10.100.100.83
    MASK 255.255.255.0
   ```

The device no. 0x150 is now configured and you can start working with the *ADwin* system.

3.4.2 Configuration in detail

With the `adconfig` program, you can execute the following functions:

– Display the list of all device numbers under Linux :
  ```
  $ adconfig
  ```

– Add a new device no. under Linux :
  ```
  # adconfig add <deviceno> [OPTIONS] TYPE net
  ```

– Configure an *ADwin* system with Ethernet interface
  ```
  # adconfig config <deviceno> [OPTIONS]
  ```

Use the same data you have used when adding the device number!

– Delete a device no.:
  `# adconfig del <deviceno>`
– Display the help for the adconfig program:
  `$ adconfig --help`
– Display the version number of the program:
  `$ adconfig --version`

The parameter <deviceno> can be indicated in decimal or hexadecimal notation, e.g. 336 or 0x150.

You will find more details about the functions of the program in the following text or you can display them with man 8 adconfig.

Displaying the list of all device numbers

Displaying the device no.

```
$ adconfig
```

A list of all configured device numbers with their parameters is displayed. The device numbers are displayed in hexadecimal notation i.e. 0x150 for decimal 336.

Add a new Ethernet device no. under Linux

Adding the device no.: Ethernet

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

<deviceno>	The device no. to add.
UID <uid|username>	Name of the user (default: root)
GID <gid|groupname>	Name of the group (default: root).
MODE <mode>	User access rights (default: 0666)
DESC <"">	Description (up to 32 chars, default "")
TYPE net IP <addr>	*ADwin* system with Ethernet interface. <addr>: IP address in the Ethernet network
PW <"">	Password (default: "")
PORT <#>	Network port number (default: 6543)
TIMEOUT <#>	Timeout of the network protocol (default: 1000 ms)

Enter only parameters, which have no default settings or whose default settings you want to change.

The access rights set by UID, GID and MODE do merely protect against accidentally faulty access, because they can be bypassed.

The installation of an *ADwin* system in an Ethernet network is described in the installation manual. Here you will also find a more detailed description of the network-specific parameters.

Pay attention that you do not have the same IP address in an Ethernet network twice. Otherwise massive communication problems may occur.

Example

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

Display the results of the configuration:

```
# 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
```

Adding the Device no: *ADwin* system and host

Adding the Device no. - *ADwin* system and host

In an IP network (e.g. Ethernet), a client PC can communicate with an *ADwin* system conected to another PC (host PC) even if it is running in a different network.

It is required that
- the host PC has access to the *ADwin* system,
- the host PC can be accessed from the client PC via network
- the program `ADwinTcpipServer` is running on the host PC (currently available for Windows only).

If the requirements are met, the following log-on parameters must be set:

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

`net IP <addr>`	IP address of the host PC

`PW <"">`
`PORT <#>`
`TIMEOUT <#>`
	The parameters must be identical to the settings you made in the host PC program `ADwinTcpip-Server` (see online help of the program).

`HOSTLINK <#>`	Additionally: Device no. of the *ADwin* system, set with `adconfig` at the host PC.

Configuring the Ethernet device without DHCP

Configuring an *ADwin* system with Ethernet interface (without DHCP)

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

`<hwaddr>`	MAC address of the *ADwin* system in the format `00:50:C2:0A:nn:nn`.
`IP <addr>`	IP address of the *ADwin* system in the Ethernet network in the format `nnn.nnn.nnn.nnn`.
`MASK <addr>`	Subnet mask in the format `nnn.nnn.nnn.nnn`.
`GATEWAY <addr>`	IP address for the default gateway (default: 0.0.0.0, i.e. no gateway)
`PORT <#>`	Network port number (default: 6543)
`PW <"">`	New password (default: "", i.e. the existing password will be deleted!)
`DESC <"">`	Description (up to 16 chars, default: "")

Enter only parameters, which have no default settings or whose default settings you want to change. Do use the same values you have used with `add`!

Please note that the password is always transferred during configuration. If you want to keep an already existing password, you must therefore indicate the password, otherwise it will be deleted.

Example

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

Configuring an *ADwin* system with Ethernet interface for DHCP

Configuring the Ethernet device with DHCP

It is possible but not recommended that an *ADwin* system gets its IP address from the DHCP server (see option DHCP).

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

<hwaddr>	The MAC address of the *ADwin* system in the format 00:50:C2:0A:nn:nn.
GATEWAY <addr>	IP address for the default gateway (default: 0.0.0.0, i.e. no gateway)
PORT <#>	Network port number (default: 6543)
PW <"">	New password (default: "", i.e.the existing password will be deleted!)
DHCP <no | yes>	Here you determine if a DHCP server is used (default: no); see below.
DESC <"">	Description (up to 16 chars, default: "")

We recommend not activating the option DHCP!
If this option is active, communication problems may occur frequently.

Enter only parameters, which have no default settings or whose default settings you want to change. Do use the same values you have used with add!

Deleting a device no.

Deleting a device no.

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

The device no. is deleted under Linux and cannot be accessed any more.

3.5 Including the Library in C

To use the functions of the library *ADwin* Linux in C or C++, go on as follows:

– Include the following lines into your program:
  - Functions for accessing the *ADwin* system:
    `#include <libadwin.h>`
  - Error numbers, you may get when accessing the *ADwin* system:
    `#include <libadwin/errno.h>`

– Include the following flags into your Makefile for the C compiler, to compile your program with the library *ADwin* Linux:

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

Note that here the environment variable `ADWINDIR` is used. (For setting the variables see page 4).
Instead of passing the library path to the linker you can enter the path to the file `/etc/ld.so.conf` and call `ldconfig`.

If you use threads you have to add:

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

Porting from Windows

The functions of the *ADwin* Linux library are compatible to the functions of the interface under Windows (`adwin32/64.dll`). Therefore, program source codes written in a programming language (here: C/C++) under Windows can normally be used under Linux without making any modifications (of course only with regards to accessing the *ADwin* systems).

Modifications in the program are only necessary where path or file names are used. Here the typical differences between Windows and Linux become apparent: Syntax and case sensitivity of the path names.

For *ADbasic* source codes porting from (and to) Windows is likewise possible.

4 Compiling *ADbasic* Source Code

In an *ADbasic* source code, you program the functions of a process that is running on the *ADwin* system. You generate an *ADbasic* source code with an editor of your choice. A comfortable Windows user interface as real-time development tool is available.

From the source code, you use the *ADbasic* compiler to create binary code, which can be run run on the appropraite *ADwin* hardware.

4.1 ADbasic and TiCoBasic Compiler

For Apple MacIntosh, the *ADbasic* and *TiCoBasic* compilers are not available. Use the Linux or Windows compilers instead.

With version 6, the *ADbasic* and *TiCoBasic* compilers for Windows have been adapted such as both can be processed without changes using Wine (since version 1.4) under Linux. All include and library files under Windows and Linux are identical.

The license key must be entered with the tool `adwin-license` before the compiler can be used. Only then will you be able to generate binary or library files.
The license key be found on the cover sheet of the *ADbasic* manual.

Upon first start of `adwin-license`, `adbasic`, or `ticobasic` Wine creates a virtual Windows registry, where e.g. the license key is stored. Therefore the first start can take a comparatively long time.

Both compilers are called under Linux in the command line (shell). The compiler compiles a source code file and generates–depending on the options and parameters–either a binary file or a library file. The function of the binary and library files is described in the *ADbasic TiCoBasic* manual.

One or more command line calls may be combined in a shell script or Makefile, to compile several source codes of a project with only one call.

4.2 Syntax ADbasic Compiler

There are 5 main options when calling the *ADbasic* compiler:

1. `adbasic /VER`	Display the compiler version number
2. `adbasic /H`	Call the help
3. `adbasic /MAKE"make-file"`	Read main option, file name and other options of a single call from the `make-file`. The text in the `makefile` may be written using several lines. Options outside the `makefile` are not permitted.
4. `adbasic /M [OPTIONS] <infile.bas>`	Compile an *ADbasic* source code and generate a binary file.
5. `adbasic /L [OPTIONS] <infile.bas>`	Compile an *ADbasic* source code and generate a library file.

The `[OPTIONS]` for compiling source code are described below.

If file names are given without or with relative path names, the base directory for a file name is the working directory, from which the compiler is called.

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>]
```

Switches

`[path/]` Directory where you find the program `adbasic`, at standard installation: `/opt/adwin/bin/`

`infile.bas` File name of the source code to be compiled. Path is optional.

`/A<fout>` Optional: Name of the binary or library file to be generated, without file extension.

A binary file is generated with the extension `.Txn`, a library file with the extension `.LIx`.

- `x` Processor type; see `<ptype>`, option `/P`.
- `n` Process number; see `<#>`, option `/PN`.

`/IP<path>` Sets the search path for include files; default `/opt/adwin/share/inc/`

`/LP<path>` Sets the search path for library files; default `/opt/adwin/share/lib/`

`/S<system>` *ADwin* system, for which the file is to be compiled:

- `/SC` Boards (ISA bus)
- `/SL` Light/16
- `/SG` Gold; default
- `/SGII` Gold II
- `/SP` Pro
- `/SPII` Pro II
- `/SX` X-A20

`/P<ptype>` Processor type, for which the file is to be compiled:

- `/P2` Processor T2
- `/P4` Processor T4
- `/P5` Processor T5
- `/P8` Processor T8
- `/P9` Processor T9; default
- `/P10` Processor T10
- `/P11` Processor T11
- `/P12` Prozessor T12
- `/P121` Prozessor T12.1

`/E<event>` Select the event source for the process:

- `/ET` Timer; default
- `/EE` External

`/PN<#>` Sets the process number (1…10) of the file to be compiled; default = 1.

`/P<prio>` Sets the priority of the process to:

- `/PH` High; default
- `/PL` Low, priority 1.
- `/PLn` Low, Level n (-10…10). Level 10 has the highest priority.

/PD<cycle>	Set cycle time (Processdelay) of the process to cycle. Default: 1000, T11: 3000.
/PS<stack>	only with option /P12 or /P121: Set size of program stack to x, Default: 1000.
/O<level>	Sets the optimize level. Optimization may shorten processing time, but may exceptionally cause run/time errors.
	/O0 optimize level 0 (off)
	/O1 optimize level 1; default
	/O2 optimize level 2
	/O3 optimize level 3 (with option /P12 only)
	/Os small memory size (with option /P12 only)
	/Of high speed size (with option /P12 only)
/L	Language for warnings and error messages.
	/LE English. Default.
	/LG German
/V<#>	Sets the version number of the process; default = 1.

Notes

Optional switches are set in square brackets. The order of the options is arbitrary. In command lines, options are case sensitive.

If the option /a is not used the generated binary or library file is saved in the directory where the source code is located.

The hardware options can be combined with the following processor options:

Hardware	Processor options
/SC	/P2, /P4, /P5, /P8
/SG, /SL	/P9
/SP	/P4, /P5, /P8, /P9, /P10
/SGII	/P11
/SPII	/P11, /P12
/SX	/P121

If an error occurs during compilation, the compiler stops and does not generate any binary or library files.

4.3 Syntax TiCoBasic Compiler

There are 5 main options when calling the *TiCoBasic* compiler:

1. `ticobasic /VER`	Display the compiler version number
2. `ticobasic /H`	Call the help
3. `ticobasic /MAKE"make-file"`	Read main option, file name and other options of a single call from the `make-file`. The text in the `makefile` may be written using several lines. Options outside the `makefile` are not permitted.
4. `ticobasic /M [OPTI-ONS] /PROCESS <infile#.bas> [PROCESS OPTIONS]`	Compile one or more *TiCoBasic* source codes and generate a binary file. Enter for each source code separately: `/PROCESS <infile#.bas> [PROCESS OPTIONS]`.
5. `ticobasic /L [OPTIONS] <infile.bas>`	Compile an *TiCoBasic* source code and generate a library file.

The `[OPTIONS]` for compiling source code are described below.

If file names are given without or with relative path names, the base directory for a file name is the working directory, from which the compiler is called.

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>]
```

Switches

`[path/]`	Directory where you find the program `ticobasic`, at standard installation: `/opt/adwin/bin/`
`/A<fout>`	Optional: Name of the binary or library file to be generated, without file extension. A binary file is generated with the extension `.TIn`, a library file with the extension `.TLx`. `x` Processor type; see `<ptype>`, option `/P`. `n` Process number; see `<#>`, option `/PN`.
`/IP<path>`	Sets the search path for include files; default `/opt/adwin/share/inc/`

/LP<path>	Sets the search path for library files; default /opt/adwin/share/lib/
/S<system>	*ADwin* system, for which the file is to be compiled: /SC Boards (ISA bus) /SL Light/16 /SG Gold; default /SGII Gold II /SP Pro /SPII Pro II /SX X-A20
/P<ptype>	Processor type, for which the file is to be compiled: /P2 Processor T2 /P4 Processor T4 /P5 Processor T5 /P8 Processor T8 /P9 Processor T9; default /P10 Processor T10 /P11 Processor T11 /P12 Prozessor T12 /P121 Prozessor T12.1
/PROCESS	Key word for options of the next source code infile.bas. Must be repeated for every source code file.
infile.bas	File name of the source code to be compiled. Path is optional.
/E<event>	Select the event source for the process: /ET Timer; default /EE External; required options /EEA, /EEM, /EEV, /EEO. /EE Without source, uncontrolled.
/PN<#>	Sets the process number (1…4) of the file to be compiled; default = 1.
/P<prio>	Sets the priority of the process to: /PH High; default /PL Low.
/PD<cycle>	Set cycle time (Processdelay) of the process to cycle. Default: 3000.
/O<level>	Sets the optimize level. Optimization may shorten processing time, but may exceptionally cause run/time errors. /O0 optimize level 0 (off) /O1 optimize level 1; default /O2 optimize level 2
/L	Language for warnings and error messages. /LE English. Default. /LG German
/V<#>	Sets the version number of the process; default = 1.

4.4 Examples for ADbasic

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

The source code file test.bas is compiled and generates the library file test.li9 in the directory /home/user/.
Since no other options than /L are indicated, the default settings are used:

- Processor: T9
- System: Gold
- Process number: 1
- Event: Timer
- Initial Processdelay: 1000
- Priority: High
- Include/path: /opt/adwin/share/inc/
- Library path: /opt/adwin/share/lib/

If you include the directory where the compiler adbasic is located, into your search path (see "Installation" page 3), you can write the line above in a shorter way:

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

In the following examples, we assume that the adbasic path is part of your search path.

```
$ adbasic /L /string.bas /sp
```

The command line call compiles the source code file string.bas to a library file for a Pro system with a T9 processor.
The same call, but for a T10 processor, is as follows:

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

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

Compiles the demo file bas_dmo6f.bas into a binary file for an *ADwin/Gold* system with T9 processor.

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

Compiles the demo file bas_dmo6.bas into a binary file for an *ADwin* board with a T8 processor.

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

Compiles the file my_file.bas for an *ADwin-Pro II* with a T11 processor. The generated binary file has the name your_file.tb1 and can be found in the current directory.

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

The name of the binary file is your_file.t91 and is located in the directory /somewhere.

5 Methods and Functions of *ADwin* Linux

The syntax of *ADwin* Linux methods and functions depends on the development environment.
This is the reason why we have used a general syntax in the description below. Use the relevant syntax for your development environment.

The syntax of the instructions is illustrated below:

```
datatype instr (datatype Var1, datatype Var2, …)
```

Meaning

`datatype`	With an instruction: Data type of the return value. With a parameter: Data type of the parameter. Data types are: `void`, `int32`, `float`, `double`, `char`.
instr	In some development environments (C, C++), instruction names are case sensitive.
Var1, *Var2*	Variables and return values are described under "Parameters", behind the instruction syntax.

Instructions for accessing analog and digital inputs and outputs (and other hardware functionalities) are not included in the *ADwin* Linux library. Such applications are programmed in *ADbasic*.

5.1 System Control and System Information

Initialization of the *ADwin* system and information about the operating status.

Set_DeviceNo

The function `Set_DeviceNo` sets the device number of the *ADwin* system for all following instructions.

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

Parameters

DeviceNo	Device number of the system. Typical device numbers are 336 (= 0x150 hexadecimal), 400 (= 0x190 hexadecimal).

Notes

The PC distinguishes and accesses the *ADwin* systems using this device number. Systems with link adapter are already delivered with the default setting: 0x150h.

The function `Set_DeviceNo` does not communicate with the *ADwin* system and therefore does not affect error handling.

Example

```
Set_DeviceNo(3)
```

The device number 0x3 is set. All further instructions refer to this device (until a new device number is set with `Set_DeviceNo`).

Booting

`Boot` initializes the *ADwin* system and loads the operating system file.

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

Parameters

Filename pointer to the file name of the operating system file

Notes

The initialization deletes all processes on the system and sets all global variables to the value 0.

The operating system, which is to be loaded depends on the processor. The following table shows the file names for the different processors.

Processor	Operating system file
T2	ADwin2.btl
T4	ADwin4.btl
T5	ADwin5.btl
T8	ADwin8.btl
T9	ADwin9.btl
T10	ADwin10.btl
T11	ADwin11.btl
T12	ADwin12.btl
T12.1	ADwin121.btl

The PC will only be able to communicate with the *ADwin* system after the operating system has been loaded. Reload the operating system after each power-up of the *ADwin* system.

Loading the operating system successfully with `Boot` takes about 1 second.

Test_Version

`Test_Version` checks, if the correct operating system for the processor has been loaded and if the processor can be accessed.

```
int32_t Test_Version()
```

Parameters

return value
- 0 : OK
- 1 : wrong driver version, processor continues working
- 2 : wrong driver version, processor stops
- 3 : no response from the *ADwin* system

Processor_Type

`Processor_Type` returns the processor type of the system.

```
int32_t Processor_Type()
```

Parameters

return value processor code number of the system

Notes

The function returns the used processor type for testing purposes, according to the table below.

return value	processor type
0	Error
2	T2
4	T4
5	T5
8	T8

return value	processor type
9	T9
1010	T10
1011	T11
1012	T12
10121	T12.1

Workload

`Workload` returns the processor workload.

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

Parameters

Priority 0 (zero): current total workload of the processor
≠0 : not supported at the moment

return value processor workload (in percent)

Free_Mem

`Free_Mem` determines and returns the free memory for different memory types.

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

Parameters

Mem_Spec type of memory
MEM_ALL (0):all types of memory; processors T2, T4, T5, T8 only
MEM_PM_LOCAL (1):local program memory; T9…T11
MEM_EM_LOCAL (2):local extra memory; T11 only
MEM_DM_LOCAL (3):local data memory; T9…T11
MEM_DRAM_EXTERN (4):external DRAM memory; T9…T11
MEM_CM (5): Memory, which can provide data to the cache; T12/T12.1 only.
MEM_UM (6): Memory, which cannot provide data to the cache; T12/T12.1 only.

return value currently free memory (in bytes)

Note

The different memory areas are described in more detail in the *ADbasic* manual.

5.2 Process Control

Instructions for controlling processes on the *ADwin* system.

Load_Process

`Load_Process` loads the binary file of a process into the *ADwin* system.

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

Parameters

Filename	pointer to the file name of the binary file

Notes

You generate a binary file under Linux by calling the compiler `adbasic` (without the option `-l`, see chapter 4.2).

Switching a system on and off of deletes loaded processes. You must therefore load the necessary processes again after power-up.

Before loading the process into the *ADwin* system, you have to ensure that no process using the same process number is already running. If there is such a process yet, you first have to stop the running process using `Stop_Process`.

If you load processes more than once, memory fragmentation can happen. Please note the appropriate hints in the *ADbasic* manual.

Start_Process

`Start_Process` starts a process.

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

Parameters

ProcessNo	number of the process (1…10)

Stop_Process

`Stop_Process` stops a process.

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

Parameters

ProcessNo	number of the process (1…12, 15)

Note

It may happen that a process cannot be stopped immediately after the instruction has been executed, but somewhat later. The status of the process is queried using `Process_Status`.

Clear_Process

`Clear_Process` deletes a process from memory.

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

Parameters

ProcessNo	number of the process (1…12, 15)

Notes

Loaded processes allocate memory in the system. You can delete processes from memory with `Clear_Process`, in order to get more memory space for other processes.

A process, which is to be deleted, must not be running. Stop a running process first with `Stop_Process`, (and determine with `Process_Status` that the process has stopped), before you delete it from memory with `Clear_Process`.

After power-up the internal processes 11, 12 and 15 are running on the *ADwin* system. If you want to delete them do so in the following order: 15, 12, 11.
Process 15 flashes the LED in the Gold and Pro systems. After deleting it, the LED does not blink any more.

Process_Status

`Process_Status` returns the status of a process.

```
int32_t Process_Status (ProcessNo)
```

Parameters

ProcessNo	number of the process (1…12, 15)
return value	status of the process 1 : process is running. 0 : process is not running, that is, it is not loaded, it has not been started or stopped. -1: process is being stopped, that is, it has received a `Stop_Process`, but still waits for the last event.

Set_Processdelay

`Set_Processdelay` sets the parameter Processdelay for a process

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

Parameters

ProcessNo	number of the process (1…10)
Process-delay	value to be set for the parameter `Processdelay`.

Notes

The parameter *Processdelay* controls the time interval between two events of a time-controlled process (see *ADbasic* manual).

The time interval is indicated in units of time, which depend on the processor type and the priority of a process.

Processor type	Process priority	
	high	low
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

Example (C)

```
Set_Processdelay(1,2000);
// the Processdelay of process is set to 2000.
```

This time-controlled process is called every 50 µs (=2000 · 25ns) in a high priority process and when using an T9 processor.

Get_Processdelay

`Get_Processdelay` returns the parameter Processdelay of a process.

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

Parameters

ProcessNo	number of the process (1…10)
return value	currently set value for *Processdelay*; in case of error: 255

Notes

The parameter *Processdelay* controls the time interval between two events of a time-controlled process (see *Set_Processdelay and ADbasic* manual).

Example (C)

```
int32_t erg;
erg = Get_Processdelay(1);
// query Processdelay of ADbasic process 1
```

5.3 Transfer of Global Variables

Instructions for data transfer between PC and *ADwin* system with the standard global variables PAR_1 ... PAR_80 and FPAR_1 ... FPAR_80.

5.3.1 Global long variables (PAR_1 ... PAR_80)

Set_Par sets a global long variable to the specified value.

Set_Par

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

Parameters

Index	number of the global long variable (1 … 80)
Value	value to be set for the long variable

Get_Par returns the value of a global long variable.

Get_Par

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

Parameters

Index	number of the global long variable (1 … 80).
return value	current value of a variable

Get_Par_Block transfers the indicated amount of global long variables into an array.

Get_Par_Block

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

Parameters

Array	pointer to an array (Array length ≥ *Count*)
StartIndex	number of the first variable to be transferred (1…80)
Count	number of variables to be transferred (max. 80)

Get_Par_All transfers all global long variables into an array.

Get_Par_All

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

Parameters

Array	pointer to an array (array length ≥ 80)

5.3.2 Global float variables (FPAR_1…FPAR_80)

Set_FPar

Set_FPar sets a global float variable to a specified value.

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

Parameters

Index	number of the global float variable (1…80)
Value	value to be set for a float variable

Get_FPar

Get_Par returns the value of a global float variable.

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

Parameters

Index	number of the global float variable (1…80)
return value	current status of the variables

Get_FPar_Block

Get_FPar_Block transfers the indicated amount of global float variables into an array.

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

Parameters

Array	pointer to an array (array length ≥ *Count*)
StartIndex	number of the first variable to be transferred (1…80)
Count	number of variables to be transferred (max. 80)

Get_FPar_All

Get_FPar_All transfers all global float variables into an array.

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

Parameters

Array	pointer to an array (array length ≥ 80)

Get_FPar_Block_Double

`Get_FPar_Block_Double` transfers the indicated amount of global float variables into a double array.

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

Parameters

Array	pointer to an array (array length ≥ *Count*)
StartIndex	number of the first variable to be transferred (1…80)
Count	number of variables to be transferred (max. 80)

Notes

This method has been developed for applications, which process floating-point data sets only with double precision.

Please consider that *ADwin* systems use single precision data. To avoid misunderstandings, you should display the array *Array[]* only with single precision.

Get_FPar_All_Double

`Get_FPar_All_Double` transfers all global float variables into a double array.

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

Parameters

Array	pointer to an array (array length ≥ 80)

Notes

This method has been developed for applications, which process floating-point data sets only with double precision.

Please consider that *ADwin* systems use single precision data. To avoid misunderstandings, you should display the array *Array[]* only with single precision.

5.4 Transfer of Data Arrays

Instructions for data transfer between PC and *ADwin* system with global DATA arrays (DATA_1…DATA_200), which may be declared as FIFO arrays.

You must declare each array in *ADbasic* with the instruction DIM before usage (see "*ADbasic*" manual).

5.4.1 Simple data arrays

Data_Length

Data_Length returns the length of an array (i.e. number of elements), declared under *ADbasic*.

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

Parameters

DataNo	number of the array (1…200)
return value	declared data length (=number of elements)

Notes

To determine the length of a string in a DATA array of the type **STRING** you use the instruction String_Length.

SetData_Long

SetData_Long transfers long data from the PC into a DATA array of the *ADwin* system.

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

Parameters

DataNo	Number (1...200) of destination array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the source array, from which data are transferred.
StartIndex	Number (≥1) of the first element in the destination array, into which data is transferred.
Count	Number (≥1) of the LONG data to be transferred.

GetData_Long

GetData_Long transfers long data from a DATA array of an *ADwin* system into an array.

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

Parameters

DataNo	Number (1...200) of the source array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the destination array, into which data are transferred.
StartIndex	Number (≥1) of the first element in the source array to be transferred.
Count	Number (≥1) of the LONG data to be transferred.

SetData_Float

SetData_Float transfers float data from the PC into a DATA array of the *ADwin* system.

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

Parameters

DataNo	Number (1...200) of destination array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the source array, from which data are transferred.
StartIndex	Number (≥1) of the first element in the array, into which data is transferred.
Count	Number (≥1) of the FLOAT data to be transferred.

GetData_Float

GetData_Float transfers float data from a DATA array of the *ADwin* system into an array.

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

Parameters

DataNo	Number (1...200) of the source array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the destination array, into which the data are transferred.
StartIndex	Number (≥1) of the first element in the source array that is transferred.
Count	Number (≥1) of the FLOAT data to be transferred.

SetData_Double

SetData_Double transfers double data from the PC with single precision to a DATA array of the *ADwin* system.

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

Parameters

DataNo	Number (1...200) of destination array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the source array, from which data are transferred.
StartIndex	Number (≥1) of the first element in the destination array, into which data is transferred.
Count	Number (≥1) of the double data to be transferred.

Notes

This method has been developed for applications, which process floating-point data sets only with double precision.

Please consider that *ADwin* systems use single precision data. To avoid misunderstandings, you should display the array `Data[]` only with single precision.

GetData_Double

GetData_Double transfers float data from a DATA array of the *ADwin* system into a double array.

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

Parameters

DataNo	Number (1...200) of the source array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the destination array, into which the values are transferred.
StartIndex	Number (≥1) of the first element in the source array to be transferred.
Count	Number (≥1) of the double data to be transferred.

Notes

This method has been developed for applications, which process floating-point data sets only with double precision.

Please consider that *ADwin* systems use single precision data. To avoid misunderstandings, you should display the array *Data[]* only with single precision.

Data2File

Data2File saves long or float data from a DATA array of the *ADwin* system in a file (on the hard disk).

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

Parameters

Filename	pointer to the file name
DataNo	Number (1...200) of the source array DATA_1 ... DATA_200.
Startindex	Number (≥1) of the first element in the source array to be transferred.
Count	Number (≥1) of the first data to be transferred.
Mode	Write mode: 0: File will be overwritten. 1: Data is appended to an existing file.

Notes

The DATA array must not be defined as **FIFO**.

The data are saved as binary file. If not existing, the file will be created.

Example (C)

```
Data2File("Test.dat", 1, 1, 1000, 0);
// Saves the elements 1…1000 from the ADbasic-
// array DATA_1 in the file Test.dat
```

5.4.2 FIFO Arrays

Instructions for data transfer between the PC and the *ADwin* system with global DATA arrays (DATA_1...DATA_200), which are declared as FIFOs.

Before using it you have to declare each FIFO array under *ADbasic* (see *ADbasic* manual): `DIM DATA_x[n] AS TYPE as FIFO`

When working with FIFO arrays you should not write more elements into the FIFO than you have declared. You should definitely not read out more elements than you have written into the FIFO. To avoid malfunction check the FIFO status first:

– Check with the function `Fifo_Full`, if a FIFO array contains at least as many elements as you wish to read out with `GetFifo_X`.

– Check with the function `Fifo_Empty`, if a FIFO array contains at least as many free elements as you wish to write into the FIFO with `SetFifo_X`.

Fifo_Empty

`Fifo_Empty` returns the number of free elements of a FIFO array.

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

Parameters

FifoNo	Number (1...200) of the FIFO array DATA_1 ... DATA_200.
return value	Number of empty elements in the FIFO array.

Fifo_Full

`Fifo_Full` returns the number of used elements in a FIFO array.

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

Parameters

FifoNo	Number (1...200) of the FIFO array DATA_1 ... DATA_200.
return value	Number of the used elements in the FIFO array.

Fifo_Clear

`Fifo_Clear` initializes the write and read pointers of a FIFO array.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.

Notes

While declaring a FIFO array (in *ADbasic*) the FIFO pointers are not automatically initialized. Therefore, call `Fifo_Clear` just at the beginning of your program, either in *ADbasic* or with this function.

Initializing the FIFO pointers during the program is useful, when all written elements are to be cancelled (for instance because of an error).

SetFifo_Long

`SetFifo_Long` transfers long data from the PC to a FIFO array of the *ADwin* system.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the source array, from which data are transferred.
Count	Number (≥1) of elements to be transferred.

GetFifo_Long

`GetFifo_Long` transfers long FIFO data from the *ADwin* system to the PC.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the destination array, into which data are transferred.
Count	Number (≥1) of elements to be transferred.

SetFifo_Float

`SetFifo_Float` transfers float data from the PC into a FIFO array of the *ADwin* system.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the source array, from which data are transferred.
Count	Number (≥1) of elements to be transferred.

GetFifo_Float

`GetFifo_Float` transfers float FIFO data from the *ADwin* system to the PC.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the destination array, into which data are transferred.
Count	Number (≥1) of elements to be transferred.

SetFifo_Double

`SetFifo_Double` transfers double data from the PC with single precision to a FIFO array of the *ADwin* system.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the source array, from which data are transferred.
Count	Number (≥1) of elements to be transferred.

Notes

This method has been developed for applications, which process floating-point data sets only with double precision.

Please consider that *ADwin* systems use single precision data. To avoid misunderstandings, you should display the array *Data[]* only with single precision.

GetFifo_Double

`GetFifo_Double` transfers float FIFO data from the *ADwin* system into a double array.

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

Parameters

FifoNo	number of the FIFO Number (1...200) of the FIFO array DATA_1 ... DATA_200.
pc_Array[]	Pointer to the destination array, into which data are transferred.
Count	Number (≥1) of elements to be transferred.

Notes

This method has been developed for applications, which process floating-point data sets only with double precision (double precision arrays).

Please consider that the data have single precision. Therefore, you should display data from the array *Data[]* only with single precision, in order to avoid misunderstandings regarding the precision.

5.4.3 Data arrays with string data

Instructions for the data transfer between PC and *ADwin* system with global DATA arrays (DATA_1…DATA_200), which contain string data. The DATA array must be declared with DIM … AS STRING in *ADbasic*.

String_Length

String_Length returns the length of a data string in a DATA array.

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

Parameters

DataNo	Number (1...200) of the array DATA_1 … DATA_200.
return value	length of the string

Notes

String_Length counts the characters in a DATA array until the first end identifier occurs (ASCII number 0). The end identifier is not counted as a character.

Example (C)

```
erg = String_Length (2);
// determines the length of the string in DATA_2.
```

SetData_String

SetData_String transfers a string into a DATA array.

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

Parameters

DataNo	number of the DATA array (1…200)
String	pointer to the string to be transferred

Notes

Each transferred string is automatically NULL-terminated(i.e. ASCII code 0 is appended).

Please consider the different usage of the development environments with this end identifier, when it is part of the string.

For instance, *ADbasic* and C transfer the string "Hello\0 world" only up to the end identifier, i.e. "Hello". Delphi (Kylix) however, transfers the complete string.

Example (C)

```
SetData_String (2, "this is a text");
// the string "this is a text" is written into the array
// DATA_2 and the end identifier is added.
```

GetData_String transfers a string from a DATA array to a buffer.

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

Parameters

DataNo	number of the DATA array (1…200)
String	pointer to the array, into which the string is to be written (array length ≥ *MaxCount*+1)
MaxCount	maximum amount of characMax. number (≥1) of the transferred characters without termination char.
	ters to be transferred
return value	amount of the read characters (without end identifier)

Notes

After writing the string into the array *pData* an additional end identifier is added as last string (ASCII number 0), so that the string contains *MaxCount*+1 elements.

If the string, which is to be transferred has an end identifier, then the transfer stops exactly there. The number of the characters read up to this point without end identifier is the return value.

If *MaxCount* is greater than the number of string chars defined in ***ADbasic***, you will receive the error "Data too small" via Get_Last_Error().

If you set *MaxCount* to a high value, the function will have an appropriately long execution time, even if the transferred string is short.
For time-critical applications with large strings, it may be faster to proceed as follows:

- You determine the actual number of chars in the string using String_Length().
- You read the string with Getdata_String() and pass the actual number of chars as *MaxCount*.

Example (C)

```
char ArrayString[100];
GetData_String (2, ArrayString, 100);
// get a string with 100 elements from DATA_2
```

If the DATA array in the *ADwin* system has an end identifier for instance at position 9, then 8 characters are read.

5.5 Error Handling

Get_Last_Error

`Get_Last_Error` returns the error number, which refers to the last instruction (of the program) sent to the *ADwin* system.

```
int32_t Get_Last_Error()
```

Parameters

return value	≠0 : number of error, which occurred last (see Annex A.1). 0 : no error occurred

Notes

The returned error number always refers to

- the last instruction, which was sent to the *ADwin* system (most but not all instructions are members of this group)
- the program, which sent that instruction

After the function is called, the error number is reset to 0 (zero).

Example (C)

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

Get_Last_Error_Text

`Get_Last_Error_Text` returns an error text related to an error number.

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

Parameters

Errno	error number
return value	pointer to the error text

Notes

This function can be called in combination with the function `Get_Last_Error`.

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_Known_DeviceNo is a two-level function: It determines either the number of the configured ***ADwin*** systems or it returns the device numbers of these systems. The result of the first calling should be used to set the array length of the second calling.

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

1st Call: determining the number of the ***ADwin*** systems

Devices	= ZERO
Count_Devices	Call: Pointer to long with value 0 (zero). return value: address where the number of ***ADwin*** systems, configured in *ADconfig*, is entered
Return value	no function

2nd Call: determining the device numbers of the systems

Devices	call: pointer to an array return value: device numbers of the systems (configured in *ADconfig*)
Count_Devices	number of the configured systems return value: number of the systems, currently configured in *ADconfig*
Return value	0 : no error; *Count_Devices* and *Devices* contain information. 1 : error, that means, *Count_Devices* was indicated lower than the number of the configured systems. *Count_Devices* and *Devices* are not changed.

Notes

Use the function in steps:

1. Use Get_Known_DeviceNo to get the number of systems configured in *ADconfig*.

2. Dimension the array Devices with the value of Count_Devices.

3. Call Get_Known_DeviceNo again, but now you pass the array and the number of the systems. You will get the device numbers of the *ADwin* systems.

Note that the passed parameters must contain the correct values for the corresponding functions phase.

Example (C)

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

// 1. call where ZERO is transferred
// Determines the number of devices configured in
// ADconfig and returns them to count_devices.
count_devices = 0;
Get_Known_DeviceNo(NULL, &count_devices);
// Set the array devices for entering the device number
// (= allocating the necessary memory)
devices = new int32_t[count_devices];

// 2. call where an array is transferred
```

```
// Query the device number and put it into devices
error = Get_Known_DeviceNo(devices, &count_devices);
```

Annex

A.1 Error messages

No.	Error message
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 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 (TcpIp).
2001	Network timeout.
2002	Wrong password.
3001	Device is unknown.

A.2 Index of methods and properties