ADwin Driver

Driver for Scilab

ADwin driver Scilab, Manual version 1.0, January 2007

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 have absolutely to be considered in order to guarantee an operation without any errors
– 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 instructions and user inputs are characterized by the font Courier New.

Var_1

ADbasic source code elements such as `INSTRUCTIONS`, *`variables`*, `comments` and `other text` are characterized by the font `Courier New` and are printed in color (see also the editor of the ***ADbasic*** development environment).

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 gives detailed information about the ***ADwin*** driver for Scilab®, versions 3 and 4.

The following documents are also important for the driver description:

– The "***ADwin*** Installation Manual" describes the hardware and software installation for all ***ADwin*** systems

– The manual "***ADbasic"*** describes the development environment and the instructions of the ***ADbasic*** compiler. The ***ADwin*** system is programmed with the easy-to-use real-time development tool ***ADbasic***.

– The hardware manuals for your ***ADwin*** systems.

It is assumed that that the user has a good command of the Scilab® environment.

Please note:

For ***ADwin*** systems to function correctly, adhere strictly to 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.

2 *ADwin* Driver for Scilab®

This section introduces into the capabilities of the ***ADwin*** driver for Scilab and describes how to communicate with an ***ADwin*** system from Scilab.

2.1 Interface to the Development Environment

The ***ADwin*** driver for Scilab is the interface to communicate with the ***ADwin*** systems.

The combination of the environment Scilab with an ***ADwin*** system provides totally new possibilities. On the one hand you use the intelligence and performance of the ***ADwin*** system for measurements, open and closed-loop controls. On the other hand you have many Scilab-functions for administration, analysis, and documentation of the measurement data and a comfortable user interface.

Applications:

– Open-loop control of fast test stands
– Signal generation
– Intelligent measurements, acquiring data under complex trigger conditions
– Open and closed-loop control
– Online processing, data reduction
– Hardware-in-the-Loop, simulation of sensor data

2.2 Communication with the *ADwin* System

With the development environment you control processes in the ***ADwin*** system, read data from there or send data to it. With the real-time development tool ***ADbasic*** you program processes. (see manual or online help ***ADbasic***).

Data and instructions between Scilab and the ***ADwin*** system are processed as shown below.

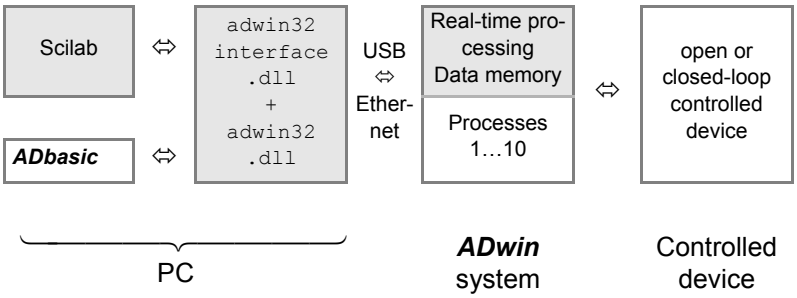

Fig. 1 – ***ADwin***-Scilab interface

adwin32.dll

The `adwin32.dll` is the central interface to the ***ADwin*** system for Windows applications and is therefore also used by the ***ADwin*** driver for Scilab. With this interface several Window programs can communicate with the ***ADwin*** system at the same time: Development environments, ***ADbasic*** and ***ADtools*** are working simultaneously with the ***ADwin*** system.

Real-Time Processing

The `adwin32.dll` interface communicates with the real-time processor of the ***ADwin*** device - the operating system. Therefore you have to load the operating system first (e.g. the file `<ADwin9.btl>`) before powering up the system. Only after it has been successfully loaded, the system is able to accept commands coming from the PC or to exchange data with it. The processes being programmed in ***ADbasic*** contain the program code for measurement, open or closed-loop control of your application.

The operating system executes the following tasks:

10 processes

– Managing up to 10 real-time processes with low or high priority (individually selectable). Low-priority processes can be interrupted by high-priority processes; high-priority cannot be interrupted by other processes.

Data memory

– Availability of global variables:
  - 80 integer variables (*PAR_1* … *PAR_80*), predefined.
  - 80 float variables (*FPAR_1* … *FPAR_80*), predefined.
  - 200 data arrays (*DATA_1* … *DATA_200*), length and data type can be set individually.

  The values of these variables or data arrays can be read and changed at any time.

Communication

– Communication between ***ADwin*** system and PC (`adwin32.dll`).

  The communication process is running at medium priority on the ***ADwin*** system and can interrupt low-priority processes for a short time. The communication process interprets and processes all instructions you are sending to the ***ADwin*** system: Control commands and commands for data exchange.

  The following table shows examples of each group.

control commands, e.g.	
`Load_Process`	loads an ***ADbasic*** process to the ***ADwin*** system
`Start_Process`	starts a process.
commands for data exchange, e.g.	
`Get_Par`	returns the current value of a global variable
`Set_Par`	changes the value of a parameter.
`GetData_Long`	returns the values from a *DATA* array of type Long.

The communication process never sends data to the PC without being told to do so. Thus, it is assured that only then data are transferred to the PC, when they have been explicitly requested before.

3 Install the *ADwin* Driver for Scilab®

Please follow the installation steps described below in order to get easy access to your ***ADwin*** device from Scilab.

3.1 Do the "*ADwin* driver installation"

For installation you need an up-to-date ***ADwin*** CDROM.

3.1.1 Installation under Linux

First, please follow the installation guide in the "ADwin Linux" manual. The ***ADwin*** driver for Scilab is contained in the Linux package.

After successful installation the driver `ADwin-scilab-driver.sce` is stored in the installation folder `</opt/adwin/share/scilab/>`.

This is how to use the Scilab driver:

– Run Scilab

– Type in the command line
`exec /opt/adwin/share/scilab/ADwin-scilab-driver.sce`.

All driver functions will now be loaded being ready for use.

3.1.2 Installation under Windows

If *ADwin* is installed yet

If you have already installed ***ADwin*** hardware and software, you have to copy the Scilab drivers only: continue with "Install Scilab driver".

Or else: New installation

Else, if ***ADwin*** hardware is to be installed, then start the installation with the manual "***ADwin*** installation" which is delivered with the ***ADwin*** hardware. It describes how to

– to install the software "`ADwin Driver and ADbasic`" from the ***ADwin*** CDROM.

– to install the hardware in the PC (if neccessary) and
set up the hardware connections between PC and ***ADwin*** system.

After a successful installation you will find the files in the folders of `<C:\ADwin\>` (standard installation):

Examples for ***ADbasic***	`.\ADbasic\samples_ADwin`
Test program for ***ADwin-Gold***, ***ADwin-light-16*** and plug-in boards.	`.\Tools\Test\ADtest.exe`
Test program for ***ADwin-Pro***	`.\Tools\Test\ADPro.exe`

Install Scilab driver

This is how to install the ***ADwin*** driver for Scilab under Windows:

– Create a directory `<C:\ADwin\Developer\Scilab\>`

– Extract the file `<ADwinScilab.zip>` into the new directory. It will then contain the following files:
- `ADwin.sce`
- `ADwin32interface.dll`
- `ADwin32interface.lib`

– Run Scilab

– Select the menu entry `File` > `Exec ...`, then select the file `ADwin.sce`.

All driver functions will now be loaded being ready for use.

3.2 Accessing the *ADwin* System

With the installation of hardware and software you have successfully checked the access to the ***ADwin*** system.

Type the following lines in the command line:

```
--> Boot('C:\ADwin\ADwin9.btl');
--> Get_Last_Error()
ans =
       0
```

This is what the lines do:

– You load the operating system of the processor T9 to the ***ADwin*** system (= booting).

  Windows only: The driver uses device number 336 (= 150 hex) as default target. If needed, set the device number in `ADwin.sce` to the one you have set in *ADconfig* during installation.

  Linux only: There is no default device number. If needed, set the device number using Set_DeviceNo (page 8).

  The filenames for other processors than T9 are given on page 9.

– You query the error code, which was created after booting. The value 0 confirms the ***ADwin*** system to be ready.

– An error code > 0 denotes an error during booting. A list of all error messages is given in chapter A.2 in the annex.

You may now use all driver functions to get access to the ***ADwin*** system.

As an introcduction we recommend to work with the program examples in the annex, section A.1.

3.3 Accessing an *ADwin* System via other PCs

If an ***ADwin*** system is connected to a host PC, but is not accessible within an Ethernet network directly, you can nevertheless get a connection using the program `ADwinTcpipServer`.

Detailed information about the use of `ADwinTcpipServer` is given on the program's online help.

4 General Information about *ADwin* Functions

4.1 Locating Errors

The function `Get_Last_Error` (see page 29) is recommended to locate errors upon execution of an ***ADwin*** function.

To handle each error, call `Get_Last_Error` after each access to the ***ADwin*** system. The function returns the number of the error that occured last.

4.2 The "DeviceNo."

A "Device No." is the number of a specified ***ADwin*** system connected to a PC. An ***ADwin*** system is always accessed via the "Device No.".

The "Device No." for the ***ADwin*** system is generated with the program *ADconfig*. You will find more information about the program's usage in the online help of *ADconfig*.

Windows only: All functions of the ***ADwin*** driver for Scilab use an internal variable `DeviceNo` to access an ***ADwin*** system. The variable is programmed in the file `<ADwin.sce>`. The default number is 336 (150 hex).

4.3 Data Types

The functions and parameters of the ***ADwin*** driver for Scilab use the following data types:

Data Type	Definition
char	unsigned integer 8-bit
int	signed integer 32- bit
float	signed float 32- bit

Variables (1×1 matrix) and row vectors can be used as function parameters. No other data types are allowed.

4.4 Exchange Data of Two-Dimensional Arrays

In ***ADbasic*** global *DATA* arrays can be declared as 2-dimensional arrays (2D). But the functions of the ***ADwin*** driver use only row vectors in Scilab.
A row vector may be easily changed into a 2-dimensional array using the Scilab function `matrix`.

In general the following table shows how an element in a 2D array in ***ADbasic*** is related to an element in a row vector in Scilab:

ADbasic	Scilab
DATA_n[i][j]	Vector[s·(i-1)+j]

Here *s* is the second dimension of *DATA_n* when you declare the array in ***ADbasic***.

Please see the notes on 2-dimensional arrays in the ***ADbasic*** manual, too.

Example: A 2D array in ***ADbasic*** is declared as

```
DIM DATA_8[7][3] AS FLOAT 'that is s=3
```

The 7×3 elements of the array are read in Scilab with `GetData_Float`:

```
--> vector = GetData_Float(8,1,21);
// read directly into a 3x7 matrix:
--> mat_3x7 = matrix(GetData_Float(8,1,21),3,7);
--> mat_7x3 = mat_3x7';
```

The data are transferred in the following order:

Index of *DATA_8*	[1][1]	[1][2]	[1][3]	[2][1]	…	[7][1]	[7][2]	[7][3]
Index of vector	[1]	[2]	[3]	[4]	…	[19]	[20]	[21]

Thus, the function `GetData_Float` transfers the element *DATA_8*[7][2] into vector[20].

The general formula `s=3` results in:

ADbasic	Scilab	
DATA_n[1][1]	Array[3·(1-1)+1]	= vector[1]
DATA_n[1][2]	Array[3·(1-1)+2]	= vector[2]
…	…	…
DATA_n[7][2]	Array[3·(7-1)+2]	= vector[20]
DATA_n[7][3]	Array[3·(7-1)+3]	= vector[21]

5 Description of the *ADwin* Driver Functions

The description of the functions is divided into the following sections:

In the appendix A.3 you find an overview of all functions.

Please pay attention to chapter 4, where general aspects for the use of ***ADwin*** functions are described.

Instructions for accessing analog and digital inputs / outputs are not part of the ***ADwin*** driver for Scilab. These actions are programmed in ***ADbasic***.

5.1 System control

Set_DeviceNo

`Set_DeviceNo` sets the device number.

```
Set_DeviceNo (DeviceNo)
```

Parameters

DeviceNo	board address or DeviceNo in decimal notation. Typical DeviceNo's are 336 (= `150h` hexadecimal), 400 (=`190h`), etc. The default setting is `336` or `hex2dec("150")`.

Notes

The PC distinguishes and accesses the ***ADwin*** systems by the device number. Systems with link adapter are already configured in factory (default setting: 336).

Further information can be found in the online help of the program *ADconfig* or in the manual "***ADwin*** Installation".

Example

```
// Set the device number 3
Set_DeviceNo(3);
```

Get_DeviceNo

Windows only: `Get_DeviceNo` returns the current device number.

```
ret_val = Get_DeviceNo ()
```

Notes

The PC distinguishes and accesses the ***ADwin*** systems by the device number. Systems with link adapter are already configured in factory (default setting: 336).

Further information can be found in the online help of the program *ADconfig* or in the manual "***ADwin*** Installation".

Example

```
// Query the current device number
num = Get_DeviceNo();
```

Boot

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

`Boot` (*`Filename`*)

Parameters

`Filename` Path and filename of the operating system file (see below).

Notes

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

The operating system file to be loaded depends on the processor type of the system you want to communicate with. The following table shows the file names for the different processors.

The files are located in the directory `<C:\ADwin\>` (Windows) or `/opt/adwin/share/btl/` (Linux).

ADwin-Type	Processor	Operating System File
ADwin-9	T9	`ADwin9.btl`
		`ADwin9s.btl`[1]
ADwin-10	T10	`ADwin10.btl`
ADwin-11	T11	`ADwin11.btl`

The computer will only be able to communicate with the ***ADwin*** system after the operating system has been loaded. Load the operating system again after each power up of the ***ADwin*** system.

Loading the operating system with `Boot` takes about one second. As an alternative you can also load the operating system via ***ADbasic*** development environment. (icon B).

Example

```
// Load the operating system for the T10 processor
Boot('C:\ADwin\ADwin10.btl'); // Windows path
```

Test_Version

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

`ret_val` = **`Test_Version`** ()

Parameters

`ret_val` 0: correct operating system
≠0: wrong operating system or no access to processor.

Example

```
// Test, if the processor system is loaded
ret_val = Test_Version();
```

1. Optimized operating system with smaller memory needs.

Processor_Type

`Processor_Type` returns the processor type of the system.

```
ret_val = Processor_Type ()
```

Parameters

ret_val	Parameter for the processor type of the system. 0: Error 9: T9 1010: T10 1011: T11

Example

```
// Query the processor type
ret_val = Processor_Type();
```

Workload

`Workload` returns the average processor workload since the last call of `Workload`.

```
ret_val = Workload (Priority)
```

Parameters

Priority	0: Current total workload of the processor. ≠0: not supported at the moment.
ret_val	Processor workload (in percent).

Notes

The processor workload is evaluated for the period between the last and the current call of `Workload`. If you need the current processor workload, you must call the function twice and in a short time interval (approx. 1 ms).

Example

```
// Query the current processor workload
Workload(0);
ret_val = Workload(0);
```

Free_Mem

`Free_Mem` returns the free memory of the system for the different memory types.

```
ret_val = Free_Mem (MemSpec)
```

Parameters

Mem_Spec	Memory type: 0 : all memory types; T2, T4, T5, T8 only 1 : internal program memory (PM_LOCAL) 2: internal data memory (EM_LOCAL); up from T11 3 : internal data memory (DM_LOCAL) 4 : external DRAM memory (DRAM_EXTERN)
ret_val	Usable free memory (in bytes)

Example

```
// Query the free memory in the external DRAM
ret_val = Free_Mem(4);
```

5.2 Process control

Instructions for the control of single processes on the ***ADwin*** system.

There are the processes 1…10 and 15:

– 1…10: You write the process in ***ADbasic*** yourself.

– 15: Control of the flash LED on ***ADwin-Gold*** and ***ADwin-Pro***.

Process 15 is part of the operating system and is started automatically after booting. For detailed information see manual ***ADbasic***, chapter "Process Management".

Load_Process

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

`Load_Process` (*`Filename`*)

Parameters

`Filename` Path and filename of the binary file to be loaded

Notes

You generate binary files in ***ADbasic*** with "`Make` > `Make Bin file`".

If you switch off your ***ADwin*** system all processes are deleted: Load the necessary processes again after power-up.

You can load up to 10 processes to an ***ADwin*** system. Running processes are not influenced by loading additional processes (with different process numbers).

Example

```
// Load binary file Testprog.T91
// T91 = Processor type T9, process no. 1
Load_Process('C:\MyADbasic\Testprog.T91');
```

Start_Process

`Start_Process` starts a process.

`Start_Process` (*`ProcessNo`*)

Parameters

`ProcessNo` Number of the process (1...10, 15).

Notes

The function has no effect, if you indicate the number of a process, which

- is already running or
- has the same number as the calling process or
- has not been loaded to the ***ADwin*** system yet.

Example

```
// Start Process 2
Start_Process(2);
```

Stop_Process

`Stop_Process` stops a process.

`Stop_Process` (*`ProcessNo`*)

Parameters

`ProcessNo` Process number (1...10, 15).

Notes

The function has no effect, if you indicate the number of a process, which

- has already been stopped or
- has not been loaded to the ADwin system yet.

Example

```
// Stop process 2
Stop_Process(2);
```

Clear_Process

`Clear_Process` deletes a process from memory.

`Clear_Process` (*`ProcessNo`*)

Parameters

`ProcessNo` Process number (1...10, 15).

Notes

Loaded processes need memory space in the system. With `Clear_Process` you can delete processes from the program memory to get more space for other processes.

If you want to delete a process, proceed as follows:

- Stop the running process with `Stop_Process`. A running process cannot be deleted.
- Check with `Process_Status`, if the process has really stopped.
- Delete the process from the memory with `Clear_Process`.

Process 15 in Gold and Pro systems is responsible for flashing the LED; after deleting (or stopping) this process the LED does not flash any more.

Example

```
// Delete process 2 from memory.
// Declared DATA and FIFO arrays remain.
Clear_Process(2);
```

Process_Status

`Process_Status` returns the status of a process.

```
ret_val = Process_Status (ProcessNo)
```

Parameters

ProcessNo	Process number (1...10, 15).
return value	Status of the process: 1 : Process is running. 0 : Process is not running, that means, it has not been loaded, not been started or has been stopped. -1: Process has been stopped, that means, it has received `Stop_Process`, but still waits for the last event.

Example

```
// Return the status of process 2
ret_val = Process_Status(2);
```

Set_Processdelay

`Set_Processdelay` sets the parameter Processdelay for a process

```
Set_Processdelay (ProcessNo, Processdelay)
```

Parameters

ProcessNo	Process number (1...10).
Processdelay	Value (1…2^{31}-1) to be set for the parameter *Processdelay* of the process (see table below).

Notes

The parameter *Processdelay* controls the time interval between two events of a time-controlled process (see manual ***ADbasic*** or online help). The parameter *Processdelay* replaces the former parameter *Globaldelay*.

For each process there is a minimum time interval: If you fall below the minimum time interval you will get an overload of the ***ADwin*** processor and communication will fail.

The time interval is specified in a time unit that depends on processor type and process priority:

Processor type	Process priority	
	high	low
T9	25ns	100µs
T10	25ns	50µs
T11	3.3ns	0.003µs = 3.3ns

Example

```
// Set Processdelay 2000 of process 1
Set_Processdelay(1,2000);
```

If process 1 is time-controlled, has high priority and runs on a T9 processor, process cycles are called every 50 µs (=2000∗25ns).

Get_Processdelay

`Get_Processdelay` returns the parameter Processdelay for a process.

```
ret_val = Get_Processdelay (ProcessNo)
```

Parameters

ProcessNo	Process number (1...10).
ret_val	The current value (1…2^{31}-1) of the parameter *Process-delay*.

Notes

The parameter *Processdelay* controls the time interval between two events of a time-controlled process (see `Set_Processdelay` as well as the manual or online help of ***ADbasic***).

Example

```
// Get Processdelay of process 1
x = Get_Processdelay(1);
```

5.3 Transfer of Global Variables

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

5.3.1 Global long variables (PAR_1...PAR_80)

The global LONG variables have the following range of values:

PAR_1 ... *PAR_80*: -2147483648 ... +2147483647
= -2^{31} ... $+2^{31}-1$

Set_Par

Set_Par sets a global LONG variable to the specified value.

Set_Par (*Index*, *Value*)

Parameters

Index Number (1 ... 80) of the global LONG variable *PAR_1* ... *PAR_80*.

Value Value to be set for the LONG variable.

Example

```
// Set LONG variable PAR_1 to 2000
Set_Par(1,2000);
```

Get_Par

Get_Par returns the value of a global LONG variable.

ret_val = **Get_Par** (*Index*)

Parameters

Index Number (1 ... 80) of the global LONG variable *PAR_1* ... *PAR_80*.

ret_val Current value of the variable.

Example

```
// Read the value of the LONG variable PAR_1
x = Get_Par(1);
```

Get_Par_Block

Get_Par_Block transfers a specified number of global LONG variables into a row vector.

ret_val = **Get_Par_Block** (*StartIndex*, *Count*)

Parameters

StartIndex Number (1 ... 80) of the first global LONG variable *PAR_1* ... *PAR_80* to be transferred.

Count Number (≥1) of the LONG variables to be transferred.

ret_val Row vector with transferred values

Example

Read the parameters PAR_10...PAR_39 and write the values to the row vector v:

```
v = Get_Par_Block(10, 30);
```

Get_Par_All

`Get_Par_All` transfers all global long variables (`PAR_1`…`PAR_80`) into a row vector.

```
ret_val = Get_Par_All ()
```

Parameters

ret_val Row vector with transferred values.

Example

Read the parameters `PAR_1`…`PAR_80` and write the values to the row vector `v`:

```
v = Get_Par_All;
```

5.3.2 Global float variables (FPAR_1...FPAR_80)

The global FLOAT variables have the following range of values:

FPAR_1 ... *FPAR_80*: negative: $-3.402823 \cdot 10^{+38}$ ... $-1.175494 \cdot 10^{-38}$
positive: $+1.175494 \cdot 10^{-38}$ ... $+3.402823 \cdot 10^{+38}$

Set_FPar

Set_FPar sets a global FLOAT variable to a specified value.

Set_FPar (*Index*, *Value*)

Parameters

Index	Number (1 ... 80) of the global FLOAT variable *FPAR_1* ... *FPAR_80*.
Value	Value to be set for the FLOAT variable.

Example

```
// Set Float-Variable FPAR_6 to 34.7
Set_FPar(6, 34.7);
```

Get_FPar

Get_Par returns the value of a global FLOAT variable.

ret_val = **Get_FPar** (*Index*)

Parameters

Index	Number (1 ... 80) of the global FLOAT variable *FPAR_1* ... *FPAR_80*.
ret_val	Current value of the variables.

Example

```
// Read the value of the FLOAT variable FPAR_56
ret_val = Get_FPar(56);
```

Get_FPar_Block

Get_FPar_Block transfers a number of global FLOAT variables, which is to be indicated, into a row vector.

ret_val = **Get_FPar_Block** (*StartIndex*, *Count*)

Parameters

StartIndex	Number (1 ... 80) of the first global FLOAT variable *FPAR_1*... *FPAR_80* to be transferred.
Count	Number (≥1) of the FLOAT variables to be transferred.
Return value	Row vector with transferred values.

Example

Read the values of the variables *FPAR_10* ... *FPAR_34* and store in a row vector *v*:

```
v = Get_FPar_Block(10,25);
```

Get_FPar_All

`Get_FPar_All` transfers all global float variables (`FPAR_1`…`FPAR_80`) into a row vector.

```
ret_val = Get_FPar_All ()
```

Parameters

Return value Row vector with transferred values.

Example

Read the values of the variables *FPAR_1* … *FPAR_80* and store in a row vector *v*:

```
v = Get_FPar_All();
```

5.4 Transfer of Data Arrays

Instructions for data transfer between PC and ***ADwin*** system with global *DATA* arrays (*DATA_1*...*DATA_200*):

- Data arrays
- FIFO Arrays
- Data Arrays with String Data

You have to declare each array in ***ADbasic*** before using it in Scilab (see "***ADbasic***" manual).

5.4.1 Data arrays

Before using it in Scilab, you have to declare each array in ***ADbasic*** with
`DIM DATA_x AS LONG / FLOAT`

The data type must fit to the used function.

The value range of an array element depends on the data type:

- LONG: -2147483648 ... +2147483647 = -2^{31} ... $+2^{31}-1$
- FLOAT: negative: $-3.402823 \cdot 10^{+38}$ ... $-1.175494 \cdot 10^{-38}$
  positive: $+1.175494 \cdot 10^{-38}$ ... $+3.402823 \cdot 10^{+38}$

Data_Length

`Data_Length` returns the length of an ***ADbasic*** array, that is the declared number of elements.

ret_val = **Data_Length** (*DataNo*)

Parameters

DataNo	Array number (1...200).
ret_val	Declared length of the array (= number of elements).

Notes

The data type of the array (LONG or FLOAT) does not matter.

Example

In ***ADbasic*** *DATA_2* is dimensioned as:

```
DIM DATA_2[2000] AS LONG
```

In Scilab you will have the length of the array *DATA_2*:

```
--> Data_Length(2)
ans =
    2000
```

SetData_Long

SetData_Long transfers data from a row vector into a DATA array of the ***ADwin*** system.

SetData_Long (*DataNo*, *Vector*, *StartIndex*)

Parameters

DataNo	Number (1...200) of destination array *DATA_1* ... *DATA_200* of type LONG.
Vector	Row vector from which data are transferred.
StartIndex	Number (≥1) of the first element in the destination array, into which data is transferred.

Notes

The DATA array must be greater than or equal to the number of values in the Scilab vector plus *StartIndex*.

If Scilab data from multi-dimensional matrices is to be transferred the data has to be copied into a row vector first. In a column vector the first data element will be transferred only.

Example

Write the complete row vector x into DATA_1, beginning at the array element DATA_1[100]:

```
SetData_Long(1,x,100);
```

SetData_Float

SetData_Float transfers data from a row vector into a DATA array of the ***ADwin*** system.

SetData_Float (*DataNo*, *Vector*, *StartIndex*)

Parameters

DataNo	Number (1...200) of destination array *DATA_1* ... *DATA_200* of type FLOAT.
Vector	Row vector from which data are transferred.
StartIndex	Number (≥1) of the first element in the destination array, into which data is transferred.

Notes

The DATA array must be greater than or equal to the number of values in the Scilab vector plus *StartIndex*.

If Scilab data from multi-dimensional matrices is to be transferred the data has to be copied into a row vector first. In a column vector the first data element will be transferred only.

Example

Write the complete row vector x into DATA_1, beginning at the array element DATA_1[100]:

```
SetData_Float(1,x,100);
```

GetData_Long

`GetData_Long` transfers parts of a `DATA` array from an ***ADwin*** system into a row vector.

ret_val = **GetData_Long** (*DataNo*, *StartIndex*, *Count*)

Parameters

DataNo	Number (1...200) of the source array *DATA_1* ... *DATA_200* of type LONG.
StartIndex	Number (≥1) of the first element in the source array to be transferred.
Count	Number (≥1) of the data to be transferred.
ret_val	Row vector with transferred values.

Notes

Even though an ***ADbasic*** array may be dimensioned 2-dimensional, the return value is always a row vector. If needed, the vector may be transformed into a matrix in Scilab, e.g. using `matrix`.

There is more information about 2-dimensional arrays in chapter 4.4 on page 7.

Example

Transfer 1000 values from `DATA_1` starting from index 100 into row vector `x`:

```
x = GetData_Long(1, 100, 1000);
```

GetData_Float

`GetData_Float` transfers parts of a `DATA` array from an ***ADwin*** system into a row vector.

ret_val = **GetData_Float** (*DataNo*, *StartIndex*, *Count*)

Parameters

DataNo	Number (1...200) of the source array *DATA_1* ... *DATA_200* of type FLOAT.
StartIndex	Number (≥1) of the first element in the source array to be transferred.
Count	Number (≥1) of the data to be transferred.
ret_val	Row vector with transferred values.

Notes

Even though an ***ADbasic*** array may be dimensioned 2-dimensional, the return value is always a row vector. If needed, the vector may be transformed into a matrix in Scilab, e.g. using `matrix`.

There is more information about 2-dimensional arrays in chapter 4.4 on page 7.

Example

Transfer 1000 values from `DATA_1` starting from index 100 into row vector `x`:

```
x = GetData_Float(1, 100, 1000);
```

Data2File

`Data2File` saves data from a `DATA` array of the ***ADwin*** system into a file (on the hard disk).

`Data2File` (*`Filename`*, *`DataNo`*, *`StartIndex`*, *`Count`*, *`Mode`*)

Parameters

`Filename`	Path and file name. If no path is indicated, the file is saved in the project directory.
`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

Save elements 1...1000 from the ***ADbasic*** array *`DATA_1`* into the file <C:\Test.dat>:

```
Data2File('C:\Test.dat', 1, 1, 1000, 0);
```

5.4.2 FIFO Arrays

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

You must declare each FIFO array before using it in ADbasic (see "***ADbasic***" manual): DIM *DATA_x*[*n*] AS *LONG* / *FLOAT* AS FIFO

The value range of an FIFO array element depends on the data type:

- LONG: -2147483648 ... +2147483647
- FLOAT: negative: $-3.402823 \cdot 10^{+38}$ ... $-1.175494 \cdot 10^{-38}$
  positive: $+1.175494 \cdot 10^{-38}$ ... $+3.402823 \cdot 10^{+38}$

To ensure that the FIFO is not full, the FIFO_EMPTY function should be used before writing into it. Similarly, the FIFO_FULL function should always be used to check if there are values which have not yet been read, before reading from the FIFO.

Fifo_Empty

Fifo_Empty returns the number of empty elements in a FIFO array.

ret_val = **Fifo_Empty** (*FifoNo*)

Parameters

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

Example

In ***ADbasic*** *DATA_5* is dimensioned as:

```
DIM DATA_5[100] AS LONG AS FIFO
```

In Scilab you will get the number of empty elements in *DATA_5*:

```
--> Fifo_Empty(5)
ans =
    68
```

Fifo_Full

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

ret_val = **Fifo_Full** (*FifoNo*)

Parameters

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

Example

In ***ADbasic*** *DATA_12* is dimensioned as:

```
DIM DATA_12[2500] AS FLOAT AS FIFO
```

In Scilab you will get the number of used elements in *DATA_12*:

```
--> Fifo_Full(12)
ans =
    2105
```

Fifo_Clear

`Fifo_Clear` initializes the write and read pointers of a FIFO array. Now the data in the FIFO array are no longer available.

`Fifo_Clear` (*`FifoNo`*)

Parameters

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

Notes

During start-up of an ***ADbasic*** program the FIFO pointers of an array are not initialized automatically. We therefore recommend to call `Fifo_Clear` at the beginning of your ***ADbasic*** program.

Initializing the FIFO pointers during program run is useful, if you want to clear all data of the array (because of a measurement error for instance).

Example

```
// Clear data in the FIFO array DATA_45
Fifo_Clear(45);
```

SetFifo_Long

`SetFifo_Long` transfers data from a row vector into a FIFO array.

`SetFifo_Long` (*`FifoNo`*, *`Vector`*)

Parameters

`FifoNo`	Number (1...200) of the FIFO array *`DATA_1`* ... *`DATA_200`* of type LONG.
`Data`	Row vector with values to be transferred.

Notes

You should first use the function `Fifo_Empty` to check, if the FIFO array has enough empty elements to hold all data of the row vector. If more data are transferred into the FIFO array than empty elements are given, the surplus data are overwritten and are definitively lost.

Example

Check FIFO array *`DATA_12`* for empty elements and transfer all elements of the row vector *`vector`* into the FIFO array:

```
num_fifo = Fifo_Empty(12);
num_vector = size(vector);
if num_fifo >= num_vector(2) then
  SetFifo_Long(12, vector);
end
```

SetFifo_Float

`SetFifo_Float` transfers data from a row vector into a FIFO array.

SetFifo_Float (*FifoNo*, *Vector*)

Parameters

FifoNo	Number (1...200) of the FIFO array *DATA_1* ... *DATA_200* of type FLOAT.
Data	Row vector with values to be transferred.

Notes

You should first use the function `Fifo_Empty` to check, if the FIFO array has enough empty elements to hold all data of the row vector. If more data are transferred into the FIFO array than empty elements are given, the surplus data are overwritten and are definitively lost.

Example

Check FIFO array *DATA_12* for empty elements and transfer all elements of the row vector *vector* into the FIFO array:

```
num_fifo = Fifo_Empty(12);
num_vector = size(vector);
if num_fifo >= num_vector(2) then
  SetFifo_Float(12, vector);
end
```

GetFifo_Long

`GetFifo_Long` transfers FIFO data from a FIFO array to a row vector.

ret_val = **GetFifo_Long** (*FifoNo*, *Count*)

Parameters

FifoNo	Number (1...200) of the FIFO array *DATA_1* ... *DATA_200* of type LONG.
Count	Number (≥1) of elements to be transferred.
ret_val	Row vector with transferred values.

Notes

You should first use the function `Fifo_Empty` to check, how much used elements the FIFO array has. If more data are read from the FIFO array than used elements are given, the surplus data is erroneous.

Example

Query the number of used elements in the FIFO array *DATA_12* and transfer 200 values into the row vector *v*:

```
num_fifo = Fifo_Full(12);
if num_fifo >= 200 then
  v = GetFifo_Long(12, 200);
end
```

GetFifo_Float

`GetFifo_Float` transfers FIFO data from a FIFO array to a row vector.

```
ret_val = GetFifo_Float (FifoNo, Count)
```

Parameters

FifoNo	Number (1...200) of the FIFO array *DATA_1* ... *DATA_200* of type FLOAT.
Count	Number (≥1) of elements to be transferred.
ret_val	Row vector with transferred values.

Notes

You should first use the function `Fifo_Empty` to check, how much used elements the FIFO array has. If more data are read from the FIFO array than used elements are given, the surplus data is erroneous.

Example

Query the number of used elements in the FIFO array *DATA_12* and transfer 200 values into the row vector *v*:

```
num_fifo = Fifo_Full(12);
if num_fifo >= 200 then
  v = GetFifo_Float(12, 200);
end
```

5.4.3 Data Arrays with String Data

Instructions for data transfer between PC and ***ADwin*** system with global *DATA* arrays (*DATA_1*...*DATA_200*) that contain string data.

You must declare each *DATA* array before using it in ***ADbasic*** (see manual "***ADbasic***"): DIM *DATA_x[n]* AS STRING.

An element in the *DATA* array of type STRING may contain a character with ASCII number 0 … 127. The termination (ASCII character 0) marks the end of a string in a *DATA* array.

String_Length

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

ret_val = **String_Length** (*DataNo*)

Parameters

DataNo	Number (1...200) of the array *DATA_1* … *DATA_200*.
ret_val	String length = number of characters.

Notes

String_Length counts the characters in a *DATA* array up to the termination char (ASCII character 0). The termination char is not counted as character.

Example

In ***ADbasic*** *DATA_2* is dimensioned as:

```
DIM DATA_2[2000] AS STRING
DATA_2 = "Hello World"
```

In MATLAB you will get the length of the array *DATA_2*:

```
--> String_Length(2)
ans =
    11
```

SetData_String

SetData_String transfers a string into *DATA* array.

SetData_String (*DataNo*, *String*)

Parameters

DataNo	Number (1...200) of the FIFO array *DATA_1* … *DATA_200*.
String	String variable or text in quotes which is to be transferred.

Notes

SetData_String appends the termination char (ASCII character 0) to each transferred string.

Example

```
SetData_String(2,'Hello World');
```

The string "Hello World" is written into the array *DATA_2* and the termination char is added.

GetData_String

`GetData_String` transfers a string from a *DATA* array into a string variable.

```
ret_val = GetData_String (DataNo, MaxCount)
```

Parameters

DataNo	Number (1...200) of the array *DATA_1* ... *DATA_200*.
MaxCount	Max. number (≥1) of the transferred characters without termination char.
ret_val	String variable with the transferred chars.

Notes

If the string in the *DATA* array contains a termination char, the transfer stops exactly there, that is the termination char will not be transferred.

String handling seems to be buggy in Scilab. In some cases Scilab adds extra spaces to transferred strings.

Example

```
// Get a string of max. 100 characters from DATA_2
string = GetData_String(2,100);
```

If the *DATA* array in the ***ADwin*** system has the termination char at position 9, then 8 characters are read.

5.5 Error handling

Show_Errors

`Show_Errors` enables or disables the display of error messages in a message box.

```
Show_Errors (OnOff)
```

Parameters

OnOff	0: Do not show any error messages. 1: Show error messages in a message box (default).

Notes

On Linux, this instruction has no effect, since message boxes are not available. `Get_Last_Error` may be used instead to create a message box "manually".

The function `Show_Errors` refers to all functions that may display error messages in a message box. These are:

- `Boot`
- `Test_Version`
- `Load_Process`

If message boxes are disabled with `Show_Errors`, the program keeps on running when an error occurs. The user cannot and does not have to confirm any error messages.

Example

```
// Show error messages
Show_Errors(1);
```

Get_Last_Error

`Get_Last_Error` returns the number of the ADwin error that was recognized last on the PC.

```
ret_val = Get_Last_Error ()
```

Parameters

ret_val	0: no error ≠0: error number

Notes

To each error number you will get the text with the function `Get_Last_Error_Text`. You will find a list of all error messages in chapter A.2 of the Appendix.

After the function call the error number is automatically reset to 0.

Even if several errors occur, `Get_Last_Error` only will only return the number of the error that occured last.

Example

```
// Read last error number
Error = Get_Last_Error();
```

Get_Last_Error_Text

`Get_Last_Error_Text` returns the error text to a given error number.

```
ret_val = Get_Last_Error_Text (LastError)
```

Parameters

Last_Error	Error number
ret_val	Error text

Notes

Usually, the return value of the function `Get_Last_Error` is used as error number *Last_Error*.

Example

```
errnum = Get_Last_Error();
if errnum<>0 then
 pErrText = Get_Last_Error_Text(errnum);
end
```

Set_Language

`Set_Language` sets the language for the error messages.

```
Set_Language (Language)
```

Parameters

language	Languages for error messages: 0: Language set in Windows 1: English 2: German

Notes

On Linux, this instruction has no effect. Error messages will always be in english.

The instruction changes the language setting for the error messages of the `adwin32.dll` and for the function `Get_Last_Error_Text`.

If a different language than English or German is set under Windows, the error messages are displayed in English.

Example

```
// set english language for error messages
Set_Language(1);
```

Annex

A.1 Program Examples

The following examples are written for a ***ADwin-Gold*** system, which is accessed via the Device No. `336`. You find the corresponding source files (and the appropriate binary files for ***ADbasic***) in the following directories:

– ***ADbasic***: `C:\ADwin\ADbasic\Samples_ADwin`

– Scilab®: `C:\ADwin\Developer\Scilab\Samples`

We assume, that you load the process to your ***ADwin*** systen from ***ADbasic*** (source file). Alternatively, you can load the binary file from Scilab® to the system with the instruction `Load_Process`.

If you use an other system (than ***ADwin-Gold***), you have to adjust the instruction `ADC` in the ***ADbasic*** programs. If you use a different Device No. than `336`, you have to set it in Scilab® with the instruction `Set_DeviceNo`.

Online Evaluation of Measurement Data

BAS_DMO1

The ***ADbasic*** program described below writes the lowest and highest measurement values of the analog input channel 1 to the parameters `Par_1` und `Par_2`.

```
REM The program BAS_DMO1 searches the maximum and
REM minimum values out of 1000 measurements of ADC1
REM and writes the result to Par_1 and Par_2

DIM i1, iw, max, min AS LONG
INIT:
 i1 = 1
 max = 0
 min = 65535

EVENT:
 iw = adc(1)
 IF (iw>max) THEN max = iw
 IF (iw<min) THEN min = iw
 i1 = i1+1
 IF (i1>1000) THEN
  i1 = 1
  Par_1 = min : REM Write minimum value to Par_1
  Par_2 = max : REM Write maximum value to Par_2
  max = 0
  min = 65535
 ENDIF
```

From Scilab® these data can be read out with the function `Get_Par(1)`:

```
// sci_dmo1.sce
// Queries 5 times PAR_1 and PAR_2
Start_Process(1)
for i = 1:3,
 minimum = Get_Par(1) // query Par_1 (minimum value)
 maximum = Get_Par(2) // query Par_2 (maximum value)
 x_message(['To continue please press OK'],['OK'])
end;
Stop_Process(1)
```

BAS_DMO2 **Digital Proportional Controller**

The ***ADbasic*** program described below is a digital proportional controller, which reads the setpoint from `PAR_1` and the gain factor from `PAR_2`.

```
REM The program BAS_DMO2 is a digital proportional
REM controller. The setpoint is defined by Par_1,
REM the gain by Par_2.

DIM deviation, actval AS LONG

EVENT:
 deviation = PAR_1 - ADC(1)
 actval = deviation * PAR_2 + 32768
 DAC(1, actval)
```

From Scilab® the setpoint and the gain factor can be changed with the following instructions:

```
Set_Par(1, 17); // Change setpoint to 17
Set_Par(2, 3); // Change gain factor to 3
```

BAS_DMO3 **Data Transfer**

The ***ADbasic*** program described below writes measurement data into a DATA-array.

```
REM The program BAS_DMO3 measures the analog input 1
REM and writes the data to a DATA array
REM The data are transferred by using a DATA-array

DIM DATA_1[1000] AS LONG
DIM index AS LONG

INIT:
 Par_10 = 0
 index = 0              'reset array pointer
 Processdelay = 40000   'cycle-time of 1ms (T9)

EVENT:
 index = index + 1      'increment array pointer
 IF (index > 1000) THEN'1000 samples done?
  Par_10 = 1                'set End-Flag
  END                   'terminate process
 ENDIF
 DATA_1[index] = ADC(1)'acquire sample and save in array
```

From Scilab® the saved DATA array can be read:

```
// sci_dmo3.sce
// reads array DATA_1.
Start_Process(1)
x = 0;
while x <> 1 do
 x = Get_Par(10);
end
y1 = GetData_Long(1,1,1000); // Read DATA_1
plot(y1);
```

A.2 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 (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.[1]
204	The file is not a BTL.
205	ADbasic binary-file is for the wrong processor or damaged.
2000	Network error (TcpIp).
2001	Network timeout.
2002	Wrong password.
3000	USB-device is unknown.
3001	Device is unknown.

1. Possibly the file <ADwin5.btl> has no memory table, or another file was renamed to <ADwin5.btl> or the file is damaged.

A.3 Index of functions