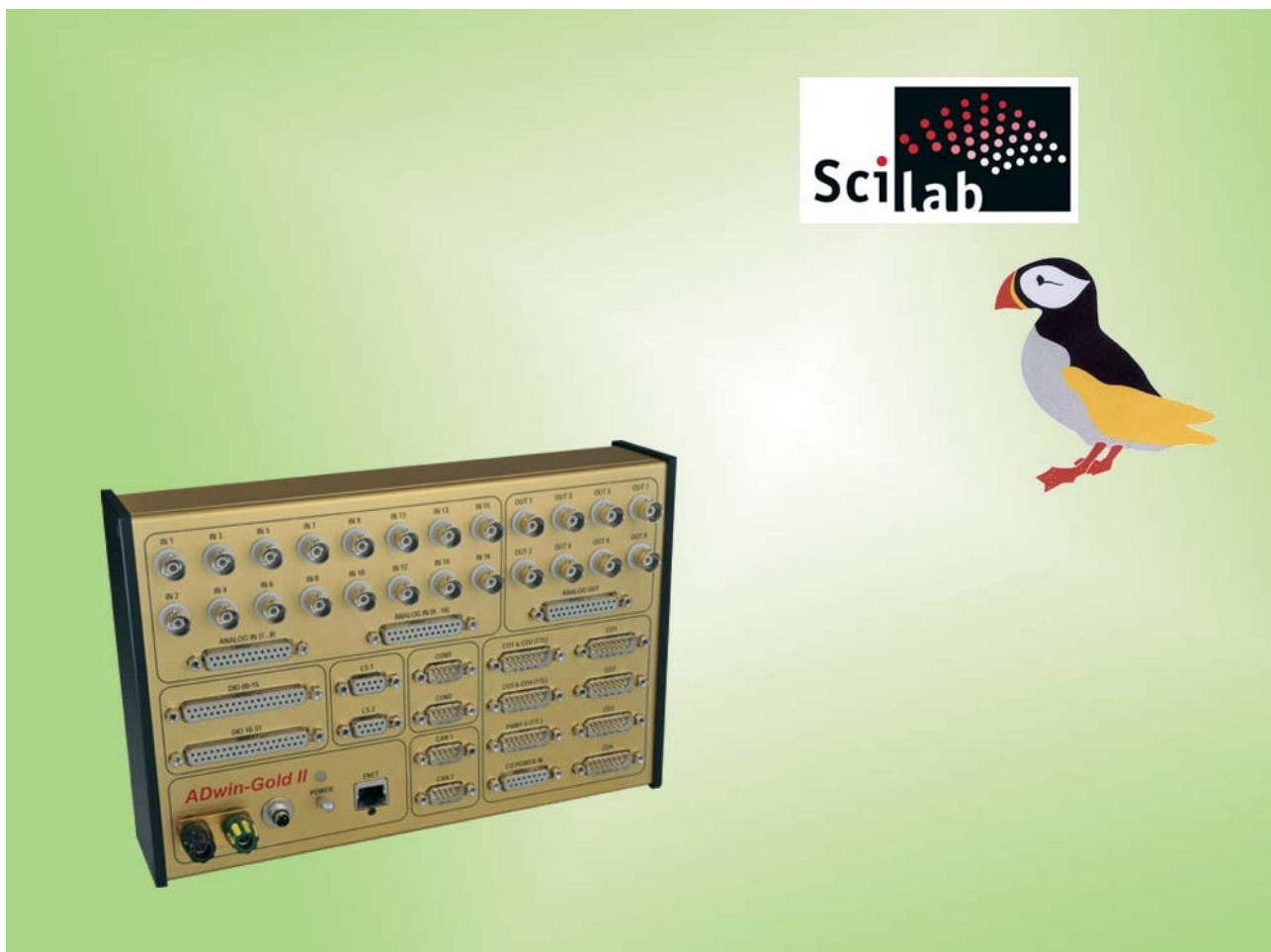


# ***ADwin Driver***

## **Driver for Scilab**



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

Hotline:	+49 6251 96320
Fax:	+49 6251 56819
E-Mail:	<a href="mailto:info@ADwin.de">info@ADwin.de</a>
Internet	<a href="http://www.ADwin.de">www.ADwin.de</a>



Jäger Com-  
putergesteuerte  
Messtechnik GmbH  
Rheinstraße 2-4  
D-64653 Lorsch  
Germany

## Table of contents

Typographical Conventions .....	IV
1 Information about this Manual .....	1
2 <b>ADwin</b> Driver for Scilab® .....	2
2.1 Interface to the Development Environment .....	2
2.2 Communication with the <b>ADwin</b> System .....	2
3 Install the <b>ADwin</b> Driver for Scilab® .....	4
3.1 Installing Hardware and Software .....	4
3.2 Accessing the <b>ADwin</b> System .....	5
3.3 Accessing an <b>ADwin</b> System via other PCs .....	5
4 General Information about <b>ADwin</b> Functions .....	6
4.1 Locating Errors .....	6
4.2 The "DeviceNo." .....	6
4.3 Data Types .....	7
4.4 Exchange Data of Two-Dimensional Arrays .....	7
5 Description of the <b>ADwin</b> Driver Functions .....	8
5.1 System control .....	8
5.2 Process control .....	12
5.3 Transfer of Global Variables .....	16
5.4 Transfer of Data Arrays .....	20
5.5 Error handling .....	30
Annex .....	A-1
A.1 Program Examples .....	A-1
A.2 List of error messages .....	A-3
A.3 Index of functions .....	A-4

## 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

ADbasic source code elements such as commands, 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 commands 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 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.

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 activities, while recognizing and avoiding any possible dangers.*

*(Definition of qualified personnel as per VDE 105 and ICE 364).*

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.

This documentation, including all pictures is protected by copyright. Reproduction, translation as well as electronical and photographic archiving and modification require a written permission by the company *Jäger Computergesteuerte Messtechnik GmbH*, Lorsch, Germany.

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

Hotline address: see inner side of cover page.



### Qualified personnel

### Availability of the documents



### Legal information

### 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. You are programming processes with the real-time development tool *ADbasic*, create a binary file and transfer it to the *ADwin* system. (see manual or online help *ADbasic*).

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

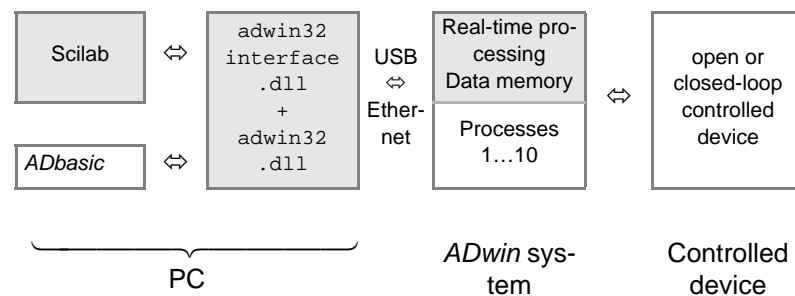


Fig. 1 – *ADwin*-Scilab interface

**adwin32.dll**

**Real-Time Processing**

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.

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 <ADwin11.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:

- 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.
- 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 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 commands 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 <i>ADbasic</i> process to the <i>ADwin</i> 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 <i>DATA</i> 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.

**10 processes**

**Data memory**

**Communication**



### 3 Install the ADwin Driver for Scilab®

For installation you need an up-to-date ADwin CDRom.

The driver functions are described in chapter 5. An alphabetical list of functions is shown on page A-7.

#### 3.1 Installing Hardware and Software

##### 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.

You can automatically run the ADwin driver upon startup using Scilab's autostart feature: Enter the above command line into the file `.scilab` (in the directory `.Scilab/scilab-[version]`).

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

##### 3.1.2 Installation under Windows

If you have already installed an ADwin system and software skip this section and continue with chapter 3.2.

Else, if an ADwin system is to be newly installed, please start the installation with the manual "ADwin installation" which is delivered with the ADwin hardware. It describes how to

- to install the software from the ADwin CDRom.
- to install the communications driver under Windows.
- to install the hardware in the PC (if necessary) and set up the hardware connections between PC and ADwin system.

After successful installation you find the files in the following folders under `C:\ADwin\` (standard installation):

Drivers and examples for Scilab	<code>.\Developer\Scilab\...</code>
Examples for ADbasic	<code>.\ADbasic\samples_ADwin</code>
Test program for ADwin-Gold, ADwin-light-16 and plug-in boards.	<code>.\Tools\Test\ADtest</code>
Test program for ADwin-Pro	<code>.\Tools\Test\ADpro</code>

Now you include the Scilab driver:

- 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.

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

If ADwin is installed

Else: New installation





## 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\ADwin11.btl');  
ans =  
      0
```

This is what the lines do:

- You load the operating system of the processor T11 to the *ADwin* system (= booting).

Windows only: The driver uses device number 1 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 T11 are given on page 9.

The error code created after booting is automatically returned. 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 introduction 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

There are 2 possibilities to locate errors upon execution of an *ADwin* function:

1. The return value `ErrorCode` of a function which indicates if an error has occurred.

We recommend to use this option.

The following functions do not return an `ErrorCode` value:

`Show_Errors`, `Set_DeviceNo`, `Get_DeviceNo`, `Test_Version`,  
`Set_Language`, `Get_Last_Error`, `Get_Last_Error_Text`

2. The function `Get_Last_Error` (see page 30) returns the number of the error that occurred last.

To handle each error, call `Get_Last_Error` after each access to the *ADwin* system.

To each error number you get the text with `Get_Last_Error_Text` (see page 31). You find a list of error messages in chapter A.2 of the Appendix.



For instance the detection of an error in the function `Get_Processdelay`.

First the recommended version:

```
// Processdelay of process 2
[gd_2, errcode] = Get_Processdelay(2);
if (gd_2 == 0) then
begin
... // no error
end;
```

Second version using `Get_Last_Error`:

```
gd_2 = Get_Processdelay(2); // Processdelay of process 2
If (Get_Last_Error() == 0) then
begin
... // no error
end;
```

### 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 1.

Return value of the  
function

`Get_Last_Error`

### 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 ( $1 \times 1$  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:

<i>ADbasic</i>	Scilab
<i>DATA_n</i> [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 <i>DATA_8</i>	[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:

<i>ADbasic</i>	Scilab
<i>DATA_n</i> [1][1]	Array[3·(1-1)+1] = vector[1]
<i>DATA_n</i> [1][2]	Array[3·(1-1)+2] = vector[2]
...	...
<i>DATA_n</i> [7][2]	Array[3·(7-1)+2] = vector[20]
<i>DATA_n</i> [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:

- System control, page 8
- Process control, page 12
- Transfer of Global Variables, page 16
- Transfer of Data Arrays, page 20
- Error handling, page 30

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.

Commands 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.  
The default setting is 1.

#### 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 = 150 hex).

Further information can be found in the online help of the program *AD-config* 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 = 150 hex).

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

#### Example

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

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

```
ErrorCode = Boot (Filename)
```

### Parameters

*Filename* Path and filename of the operating system file (see below).  
*ErrorCode* *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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 <sup>1</sup>
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 .

### Example

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

**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();
```

## Boot



## Test\_Version

1. Optimized operating system with smaller memory needs.

**Processor\_Type**

`Processor_Type` returns the processor type of the system.

```
[ret_val, ErrorCode] = Processor_Type (ErrorCode)
```

**Parameters**

<i>ret_val</i>	Parameter for the processor type of the system. 0: Error 9: T9 1010: T10 1011: T11
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

**Example**

```
// Query the processor type
[ret_val, ErrCode] = Processor_Type();
```

**Workload**

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

```
[ret_val, ErrorCode] = Workload (Priority)
```

**Parameters**

<i>Priority</i>	0: Current total workload of the processor. ≠0: not supported at the moment.
<i>ret_val</i>	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
[ret_val, ErrCode] = Workload(0);
```

---

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

```
[ret_val, ErrorCode] = Free_Mem (MemSpec)
```

### Parameters

<i>Mem_Spec</i>	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)
<i>ret_val</i>	Usable free memory (in bytes)
<i>ErrorCode</i>	<i>ADwin</i> error number which was recognized last on the PC. For more information, see chapter 4.1.

### Example

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

---

### Free\_Mem

## 5.2 Process control

Commands 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.

```
ErrorCode = Load_Process (Filename)
```

#### Parameters

<i>Filename</i>	Path and filename of the binary file to be loaded
<i>ErrorCode</i>	<i>ADwin</i> error number which was recognized last on the PC. For more information, see chapter 4.1.

#### 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.TB1
// TB1 = Processor type T11, process no. 1
ErrCode = Load_Process('C:\MyADbasic\Testprog.TB1');
```



### Start\_Process

`Start_Process` starts a process.

```
ErrorCode = Start_Process (ProcessNo)
```

#### Parameters

<i>ProcessNo</i>	Number of the process (1...10, 15).
<i>ErrorCode</i>	<i>ADwin</i> error number which was recognized last on the PC. For more information, see chapter 4.1.

#### 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
ErrCode = Start_Process(2);
```



`Stop_Process` stops a process.

```
ErrorCode = Stop_Process (ProcessNo)
```

### Parameters

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

`ErrorCode` ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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
ErrCode = Stop_Process(2);
```

`Clear_Process` deletes a process from memory.

```
ErrorCode = Clear_Process (ProcessNo)
```

### Parameters

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

`ErrorCode` ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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.
ErrCode = Clear_Process(2);
```

## Stop\_Process

## Clear\_Process



**Process\_Status**

`Process_Status` returns the status of a process.

```
[ret_val, ErrorCode] = Process_Status (ProcessNo)
```

**Parameters**

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

*ret\_val*       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.

*ErrorCode*    *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

**Example**

```
// Return the status of process 2
[ret_val, ErrCode] = Process_Status(2);
```

**Set\_Processdelay**

`Set_Processdelay` sets the parameter *Processdelay* for a process

```
ErrorCode = Set_Processdelay (ProcessNo,
                               Processdelay)
```

**Parameters**

*ProcessNo*    Process number (1...10).

*Process-delay*    Value ( $1 \dots 2^{31}-1$ ) to be set for the parameter *Processdelay* of the process (see table below).

*ErrorCode*    *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

**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 of process 1 to 50µs (T11, high priority)
ErrCode = Set_Processdelay(1, 50E-6/3.3E-9);
```

If process 1 is time-controlled, has high priority and runs on a T11 processor, process cycles are called every 50 µs.

---

Get\_Processdelay returns the parameter Processdelay for a process.

```
[ret_val, ErrorCode] = Get_Processdelay (ProcessNo)
```

### Parameters

<i>ProcessNo</i>	Process number (1...10).
<i>ret_val</i>	The current value ( $1 \dots 2^{31}-1$ ) of the parameter <i>Processdelay</i> .
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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,ErrCode] = Get_Processdelay(1);
```

---

### Get\_Processdelay

### 5.3 Transfer of Global Variables

Commands for data transfer between PC and ADwin device with the pre-defined 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:

$$\begin{aligned} PAR_1 \dots PAR_{80}: & \quad -2147483648 \dots +2147483647 \\ & \quad = -2^{31} \dots +2^{31}-1 \end{aligned}$$

#### Set\_Par

`Set_Par` sets a global LONG variable to the specified value.

```
ErrorCode = Set_Par (Index, Value)
```

##### Parameters

<i>Index</i>	Number (1 ... 80) of the global LONG variable <i>PAR_1 ... PAR_80</i> .
<i>Value</i>	Value to be set for the LONG variable.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

##### Example

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

#### Get\_Par

`Get_Par` returns the value of a global LONG variable.

```
[ret_val, ErrorCode] = Get_Par (Index)
```

##### Parameters

<i>Index</i>	Number (1 ... 80) of the global LONG variable <i>PAR_1 ... PAR_80</i> .
<i>ret_val</i>	Current value of the variable.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

##### Example

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

**Get\_Par\_Block** transfers a specified number of global LONG variables into a row vector.

```
[ret_val, ErrorCode] = 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 ( $\geq 1$ ) of the LONG variables to be transferred.

*ret\_val*       Row vector with transferred values

*ErrorCode*    ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### Example

Read the parameters *PAR\_10*...*PAR\_39* and write the values to the row vector *v*:

```
[v,ErrCode] = Get_Par_Block(10, 30);
```

**Get\_Par\_All** transfers all global long variables (*PAR\_1*...*PAR\_80*) into a row vector.

```
[ret_val, ErrorCode] = Get_Par_All ()
```

### Parameters

*ret\_val*       Row vector with transferred values.

*ErrorCode*    ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### Example

Read the parameters *PAR\_1*...*PAR\_80* and write the values to the row vector *v*:

```
[v,ErrCode] = Get_Par_All;
```

## Get\_Par\_Block

## 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:  $-3.402823 \cdot 10^{+38} \dots -1.175494 \cdot 10^{-38}$   
                                  positive:  $+1.175494 \cdot 10^{-38} \dots +3.402823 \cdot 10^{+38}$

**Set\_FPar**

Set\_FPar sets a global FLOAT variable to a specified value.

*ErrorCode* = **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.  
*ErrorCode*                *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

**Example**

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

**Get\_FPar**

Get\_FPar returns the value of a global FLOAT variable.

[*ret\_val*, *ErrorCode*] = **Get\_FPar** (*Index*)

**Parameters**

*Index*                      Number (1 ... 80) of the global FLOAT variable *FPAR\_1 ... FPAR\_80*.  
*ret\_val*                    Current value of the variables.  
*ErrorCode*                *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

**Example**

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

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

```
[ret_val, ErrorCode] = 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 ( $\geq 1$ ) of the FLOAT variables to be transferred.

*ret\_val*       Row vector with transferred values.

*ErrorCode*    ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### Example

Read the values of the variables *FPAR\_10* ... *FPAR\_34* and store in a row vector *v*:

```
[v,ErrCode] = Get_FPar_Block(10,25);
```

`Get_FPar_All` transfers all global FLOAT variables (*FPAR\_1*...*FPAR\_80*) into a row vector.

```
[ret_val, ErrorCode] = Get_FPar_All ()
```

### Parameters

*ret\_val*       Row vector with transferred values.

*ErrorCode*    ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### Example

Read the values of the variables *FPAR\_1* ... *FPAR\_80* and store in a row vector *v*:

```
[v,ErrCode] = Get_FPar_All();
```

### Get\_FPar\_Block

### Get\_FPar\_All

## 5.4 Transfer of Data Arrays

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

#### Data\_Length

*Data\_Length* returns the length of an *ADbasic* array, that is the declared number of elements.

```
[ret_val, ErrorCode] = Data_Length (DataNo)
```

#### Parameters

<i>DataNo</i>	Array number (1...200).
<i>ret_val</i>	Declared length of the array (= number of elements).
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

#### 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, 0]
```



SetData\_Long transfers data from a row vector into a DATA array of the ADwin system.

```
ErrorCode = SetData_Long (DataNo, Vector,
                          StartIndex)
```

### Parameters

<i>DataNo</i>	Number (1...200) of destination array <i>DATA_1</i> ... <i>DATA_200</i> of type LONG.
<i>Vector</i>	Row vector from which data are transferred.
<i>StartIndex</i>	Number ( $\geq 1$ ) of the first element in the destination array, into which data is transferred.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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]:

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

SetData\_Float transfers data from a row vector into a DATA array of the ADwin system.

```
ErrorCode = SetData_Float (DataNo, Vector,
                          StartIndex)
```

### Parameters

<i>DataNo</i>	Number (1...200) of destination array <i>DATA_1</i> ... <i>DATA_200</i> of type FLOAT.
<i>Vector</i>	Row vector from which data are transferred.
<i>StartIndex</i>	Number ( $\geq 1$ ) of the first element in the destination array, into which data is transferred.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC.

### 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]:

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

## SetData\_Long

## SetData\_Float

**GetData\_Long**

GetData\_Long transfers parts of a DATA array from an ADwin system into a row vector.

```
[ret_val, ErrorCode] = GetData_Long (DataNo,
                                     StartIndex, Count)
```

**Parameters**

<i>DataNo</i>	Number (1...200) of the source array <i>DATA_1</i> ... <i>DATA_200</i> of type LONG.
<i>StartIndex</i>	Number ( $\geq 1$ ) of the first element in the source array to be transferred.
<i>Count</i>	Number ( $\geq 1$ ) of the data to be transferred.
<i>ret_val</i>	Row vector with transferred values.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

**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,ErrCode] = 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, ErrorCode] = GetData_Float (DataNo,
                                     StartIndex, Count)
```

**Parameters**

<i>DataNo</i>	Number (1...200) of the source array <i>DATA_1</i> ... <i>DATA_200</i> of type FLOAT.
<i>StartIndex</i>	Number ( $\geq 1$ ) of the first element in the source array to be transferred.
<i>Count</i>	Number ( $\geq 1$ ) of the data to be transferred.
<i>ret_val</i>	Row vector with transferred values.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

**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 1 000 values from `DATA_1` starting from index 100 into row vector `x`:

```
[x,ErrCode] = GetData_Float(1, 100, 1000);
```

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

```
ErrorCode = Data2File (Filename, DataNo, StartIndex, Count, Mode)
```

### Parameters

<i>Filename</i>	Path and file name. If no path is indicated, the file is saved in the project directory.
<i>DataNo</i>	Number (1...200) of the source array <code>DATA_1</code> ... <code>DATA_200</code> .
<i>StartIndex</i>	Number ( $\geq 1$ ) of the first element in the source array to be transferred.
<i>Count</i>	Number ( $\geq 1$ ) of the first data to be transferred.
<i>Mode</i>	Write mode: 0: File will be overwritten. 1: Data is appended to an existing file.
<i>ErrorCode</i>	<i>ADwin</i> error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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>`:

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

### Data2File

### 5.4.2 FIFO Arrays

Commands 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, ErrorCode] = Fifo_Empty (FifoNo)
```

#### Parameters

<i>FifoNo</i>	Number (1...200) of the FIFO array <i>DATA_1 ... DATA_200</i> .
<i>ret_val</i>	Number of empty elements in the FIFO array.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

#### 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, 0]
```

#### Fifo\_Full

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

```
[ret_val, ErrorCode] = Fifo_Full (FifoNo)
```

#### Parameters

<i>FifoNo</i>	Number (1...200) of the FIFO array <i>DATA_1 ... DATA_200</i> .
<i>ret_val</i>	Number of used elements in the FIFO array.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

#### 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, 0]
```

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

```
ErrorCode = Fifo_Clear (FifoNo)
```

### Parameters

*FifoNo*            Number (1...200) of the FIFO array *DATA\_1 ... DATA\_200*.

*ErrorCode*        *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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
ErrCode = Fifo_Clear(45);
```

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

```
ErrorCode = 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.

*ErrorCode*        *ADwin* error number which was recognized last on the PC. For more information, see chapter 4.1.

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

### Fifo\_Clear

### SetFifo\_Long

**SetFifo\_Float**

SetFifo\_Float transfers data from a row vector into a FIFO array.

```
ErrorCode = 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.

*ErrorCode*        ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

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

**GetFifo\_Long**

GetFifo\_Long transfers FIFO data from a FIFO array to a row vector.

```
[ret_val, ErrorCode] = GetFifo_Long (FifoNo, Count)
```

**Parameters**

*FifoNo*            Number (1...200) of the FIFO array *DATA\_1 ... DATA\_200* of type LONG.

*Count*            Number ( $\geq 1$ ) of elements to be transferred.

*ret\_val*           Row vector with transferred values.

*ErrorCode*        ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

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

GetFifo\_Float transfers FIFO data from a FIFO array to a row vector.

```
[ret_val, ErrorCode] = GetFifo_Float (FifoNo, Count)
```

### Parameters

<i>FifoNo</i>	Number (1...200) of the FIFO array <i>DATA_1</i> ... <i>DATA_200</i> of type FLOAT.
<i>Count</i>	Number ( $\geq 1$ ) of elements to be transferred.
<i>ret_val</i>	Row vector with transferred values.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

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

### GetFifo\_Float

### 5.4.3 Data Arrays with String Data

Commands 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`.

Each element in the *DATA* array of type *STRING* may contain a letter. The termination char (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, ErrorCode] = String_Length (DataNo)
```

#### Parameters

<i>DataNo</i>	Number (1...200) of the array <i>DATA_1 ... DATA_200</i> .
<i>ret_val</i>	String length = number of characters.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

#### 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, 0]
```

#### SetData\_String

*SetData\_String* transfers a string into *DATA* array.

```
ErrorCode = SetData_String (DataNo, String)
```

#### Parameters

<i>DataNo</i>	Number (1...200) of the FIFO array <i>DATA_1 ... DATA_200</i> .
<i>String</i>	String variable or text in quotes which is to be transferred.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

#### Notes

*SetData\_String* appends the termination char (ASCII character 0) to each transferred string.

#### Example

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

The string "Hello World" is written into the array *DATA\_2* and the termination char is added.



GetData\_String transfers a string from a *DATA* array into a string variable.

```
[ret_val, ErrorCode] = GetData_String (DataNo,
    MaxCount)
```

### Parameters

<i>DataNo</i>	Number (1...200) of the array <i>DATA_1</i> ... <i>DATA_200</i> .
<i>MaxCount</i>	Max. number ( $\geq 1$ ) of the transferred characters without termination char.
<i>ret_val</i>	String variable with the transferred chars.
<i>ErrorCode</i>	ADwin error number which was recognized last on the PC. For more information, see chapter 4.1.

### 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.

If *MaxCount* is greater than the number of string chars defined in *AD-basic*, 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

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

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

### GetData\_String

## 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

With Linux, this command 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` will only return the number of the error that occurred last.

#### Example

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

---

`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` sets the language for error messages.

```
Set_Language (Language)
```

### Parameters

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

### Notes

With Linux, this command has no effect. Error messages will always be in english.

The command 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);
```

## Get\_Last\_Error\_Text

## Set\_Language

## Annex

### A.1 Program Examples

The following examples are written for a *ADwin-Gold* system, which is accessed via the Device No. 1. 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

The Scilab program loads the *ADbasic* process to your *ADwin* system.

If you use an other system (than *ADwin-Gold*), you have to adjust the command **ADC** in the *ADbasic* programs. If you use a different Device No. than 1, you have to set it in Scilab® with the command `Set_DeviceNo`.

#### BAS\_DMO1

#### Online Evaluation of Measurement Data

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 il, iw, max, min As Long
Init:
  il = 1
  max = 0
  min = 65535

Event:
  iw = ADC(1)
  If (iw>max) Then max = iw
  If (iw<min) Then min = iw
  il = il+1
  If (il>1000) Then
    il = 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_dm01.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)
```

### 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, actual As Long
```

#### Event:

```
deviation = Par_1 - ADC(1)
actual = deviation * Par_2 + 32768
DAC(1, actual)
```

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

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

### 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);
```

### BAS\_DMO2

### BAS\_DMO3

### A.2 List of 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. <sup>1</sup>
204	The file is not a BTL.
2000	Network error (Tcplp).
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

Boot (Filename) . . . . .	9
Clear_Process (ProcessNo) . . . . .	13
Data_Length (DataNo) . . . . .	20
Data2File (Filename, DataNo, StartIndex, Count, Mode) . . . . .	23
Fifo_Clear (FifoNo) . . . . .	25
Fifo_Empty (FifoNo) . . . . .	24
Fifo_Full (FifoNo) . . . . .	24
Free_Mem (MemSpec) . . . . .	11
Get_DeviceNo () . . . . .	8
Get_FPar (Index) . . . . .	18
Get_FPar_All () . . . . .	19
Get_FPar_Block (StartIndex, Count) . . . . .	19
Get_Last_Error () . . . . .	30
Get_Last_Error_Text (LastError) . . . . .	31
Get_Par (Index) . . . . .	16
Get_Par_All () . . . . .	17
Get_Par_Block (StartIndex, Count) . . . . .	17
Get_Processdelay (ProcessNo) . . . . .	15
GetData_Float (DataNo, StartIndex, Count) . . . . .	22
GetData_Long (DataNo, StartIndex, Count) . . . . .	22
GetData_String (DataNo, MaxCount) . . . . .	29
GetFifo_Float (FifoNo, Count) . . . . .	27
GetFifo_Long (FifoNo, Count) . . . . .	26
Load_Process (Filename) . . . . .	12
Process_Status (ProcessNo) . . . . .	14
Processor_Type (ErrorCode) . . . . .	10
Set_DeviceNo (DeviceNo) . . . . .	8
Set_FPar (Index, Value) . . . . .	18
Set_Language (Language) . . . . .	31
Set_Par (Index, Value) . . . . .	16
Set_Processdelay (ProcessNo, Processdelay) . . . . .	14
SetData_Float (DataNo, Vector, StartIndex) . . . . .	21
SetData_Long (DataNo, Vector, StartIndex) . . . . .	21
SetData_String (DataNo, String) . . . . .	28
SetFifo_Float (FifoNo, Vector) . . . . .	26
SetFifo_Long (FifoNo, Vector) . . . . .	25
Show_Errors (OnOff) . . . . .	30
Start_Process (ProcessNo) . . . . .	12
Stop_Process (ProcessNo) . . . . .	13
String_Length (DataNo) . . . . .	28
Test_Version () . . . . .	9
Workload (Priority) . . . . .	10