ADwin

Driver for LabVIEW Version 5 and higher

October 03

Table of Contents

1 How to work with LabVIEW and *ADwin*

More features for LabVIEW with *ADwin*

The ***ADwin*** system consists of an independent computer, which executes measurement and control tasks very fast and reliably, as well as of an interface under Windows in order to control the ***ADwin*** system with LabVIEW. Consequently you transfer all time-critical processes to the ***ADwin*** system, but with LabVIEW you still have control of the processes and data processing.

How to program the *ADwin* system

ADwin systems are fast, reliable and flexible. You apply the easy-to-learn programming language ***ADbasic*** in order to use all these advantages. Before you can apply the here described LabVIEW instructions, we recommend to familiarize yourself with ***ADbasic***. Please use the ***ADbasic*** manual and the programming instructions as help. The descriptions will help you to understand the ***ADwin*** system more easily.

Controlling *ADwin* systems with LabVIEW

Now it's time to start working with this manual.

Chapter 2 explains how LabVIEW and ***ADwin*** communicate with each other and deepens your knowledge for the ***ADwin*** concept.

In chapter 3 the installation and application of the LabVIEW driver software is described, as well as the integration of the new commands.

The functions of the LabVIEW drivers are explained in chapter 4, which can also be used as a kind of reference documentation.

We wish you success when working with ***ADwin***, the real-time system, accurate to a microsecond.

This manual describes how to use the ADwin driver for **LabVIEW, version 5 and higher**.

The driver can not be used for earlier versions.

2 *ADwin* driver concept

2.1 *ADwin* operating system

The ***ADwin*** operating system can be found in the file <ADwin9.btl>[1] (ADSP / T9) or <ADwin10.btl> (ADSP / T10).

After each start-up of the power supply you first have to load (boot) the operating system to the ***ADwin*** system. After successful loading, the system will be able to accept commands coming from the PC or to exchange data with it.

The operating system executes the following tasks:

- Management of up to 10 processes, where you can distinguish two individually selectable priority levels (see the following table).

Priority	Characteristics	Application
low	can be interrupted by high-priority processes	for non-time-critical calculations and slow measurements
high	cannot be interrupted by any other process	for measurements and controls requiring exact timing

- Providing 80 pre-defined integer variables (PAR_1 to PAR-80) and 80 pre-defined float variables (FPAR_1 to FPAR_80). Moreover, 200 data sets whose length can be individually defined (DATA_1 to DATA_200) are also provided. You can read or change the values of these variables or data from the PC at any time.

- Organisation and execution of the communication between the ***ADwin*** system and the PC.

An essential element of the ***ADwin*** operating system is the communication process. This process is running at mid-priority on the ***ADwin*** system, so it can interrupt low-priority processes for a short time. The communication process compiles all commands, the PC transfers to the ***ADwin*** system. The main commands can be divided into 2 groups: Control commands and commands for data exchange. The following tables show some examples from each of these groups.

Control commands	
Boot	transfers a process to the ***ADwin*** system
Start_process	starts a process

Commands for data exchange	
get_par	reads the current value of a parameter
set_par	changes the value of a parameter
get_data	reads the values of a data set

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.

[1] When working with an ADwin-2-, -4-, -5- or –8 system, please use the files <ADwin2.btl>, <ADwin4.btl>, <ADwin5.btl> oder <ADwin8.btl> respectively.

2.2 Windows-*ADwin* interface

The <ADwin32.dll> (**D**ynamic **L**ink **L**ibrary) is the interface between Windows and the ***ADwin*** system. All Windows programs – and LabVIEW, too – being able to call DLL-functions, can communicate with the ***ADwin*** system (see the following figure).

ADwin interface to Windows applications

The interface performs the following tasks:

- Forwarding commands

  Every command to the ***ADwin*** system passes 'through' the <ADwin32.dll>. Therefore the interface is identical for all Windows programs. The <ADwin32.dll> is the only "module" connecting the ***ADwin*** system.

- Data transfer

  The data between the ***ADwin*** system and the PC are transferred via <ADwin32.dll>.
  Under Windows NT the <ADwin32.dll> accesses the ***ADwin*** system via ***ADwin*** device driver. All other Windows versions allow direct access.
  Since the <ADwin32.dll> needs neither interrupts nor DMA channels, a reliable and easy operation of the ***ADwin*** system is possible in every PC, no complex configuration procedures are necessary.

- Multitasking management

  Several Windows programs can communicate with the ***ADwin*** system at the same time via <ADwin32.dll> (time slicing). If a program exchanges data with the ***ADwin*** system, access for all other programs is denied by the <ADwin32.dll>. Immediately after finish of the data exchange the <ADwin32.dll> releases the ***ADwin*** system for other tasks. The time needed for data exchange are only few milliseconds.

3 Software installation

3.1 Installation of the *ADwin* drivers

Insert the provided CD-ROM into the drive of your computer. Normally the setup program starts automatically.

If not, please start there the program <`setup.exe`>.

Please pay attention to the notes in <`readme.txt`> and install the software according to the information given there.

After successfull installation you will find on your hard disk (among other things) the following files, which you will need later. After a standard installation you will find the files in the directory <`C:\ADwin\`>.

`..\Developer\LabVIEW\ADwin_v2.lib\*.VI`	Driver (VI collection) LabVIEW 5/6/7
`..\Developer\LabVIEW\ADwin_v2.lib\Bas_dmo*.VI`	Examples for the LabVIEW 5/6/7 driver
`..\Developer\LabVIEW\ADwin.lib\*.VI`	Driver (VI collection) for LabVIEW 4
`..\Developer\LabVIEW\ADwin.lib\Bas_dmo*.VI`	Examples for the LabVIEW 4 driver
`..\Tools\Test\ADTEST.EXE`	Test program for ***ADwin*** PC plug-in boards, ***ADwin-Gold*** and ***ADwin-L16***
`..\Tools\Test\ADPRO.EXE`	Test program for ***ADwin-Pro*** system

3.2 Loading the *ADwin* operating system

The computer will only be able to communicate with the ***ADwin*** system after the operating system has been loaded. You have the following 3 possibilities to load the operating system:

a) Under LabVIEW

With ***ADwin***-VI 'Boot'. For more information see chapter 4.

b) With ADbasic

- Start ***ADbasic*** by selecting

  `Start ► Programs ► ADwin ► ADbasic 3.2` in the Windows menu.

- Check the settings in the menu `Options ► Compiler` to be corresponding to your ***ADwin*** system.

- In the ***ADbasic*** toolbar, click on the following symbol: 
  (optional: `Project ► Boot ADwin`)

The successful loading of the operating system is confirmed in the status line of the editor, where the message "ADwin is booted" appears. If any problems occur you will find more information in the manual and in the programming instructions.

c) With the program ADtest

The program ***ADtest*** is a means that helps to control all functions for the ***ADwin*** plug-in boards and also offers the possibility to load the operating system. Start the program `ADtest.exe`. If the operating system has not yet been loaded to the ***ADwin*** system since start-up of the PC, a dialogue box appears.

Make sure that all settings correspond to those on the ***ADwin*** system and then press the OK button. In case no error message is displayed, the operating system has been loaded successfully. In case the operating system has not been loaded successfully, you may repeat the procedure with a different configuration by pressing the settings button.

The program ***ADtest*** displays the momentary voltage levels of the analogue inputs 1-12 as well as those of the digital inputs. Moreover, you can set the analogue outputs and the digital outputs. For testing, a 10 Hz sine or triangle signal can be sent to the analogue outputs.

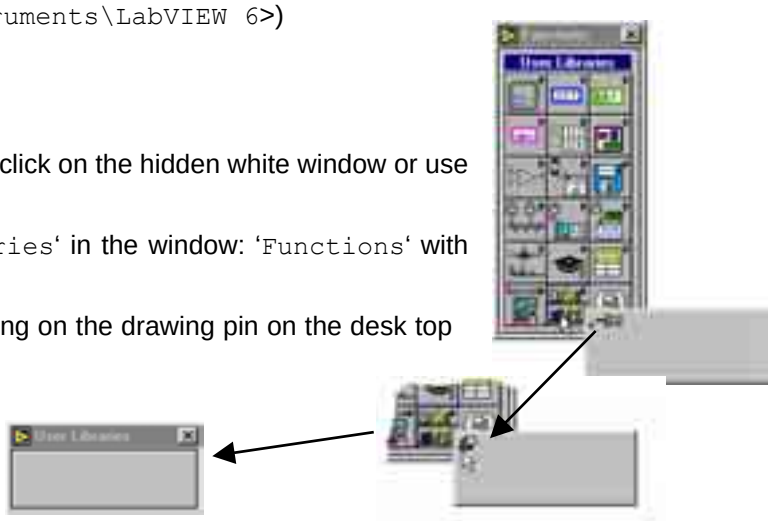
By loading the operating system, all processes on the ***ADwin*** system are cleared and all global variables are set to the value 0. After loading the operating system, the default setting for the global delay is 25 µs, using an ADSP, and 1000 µs for other processors.

3.3 Including the *ADwin* VIs

In LabVIEW programs you can use VIs (=virtual instrument) in order to send commands and data to or to get data from the ***ADwin*** system. The VIs are using functions, which are included in the <ADwin32.dll>.

The following instructions are necessary to include the VIs in LabVIEW:

- Copy the folder <..\ADwin_v2.lib\...> into your LabVIEW directory (e.g. <C:\Programs\National Instruments\LabVIEW 6>)
- Start LabVIEW
- Click on 'New VI'
- Activate the white diagram window (click on the hidden white window or use the keys [Strg]+[E])
- Click on the symbol 'User Libraries' in the window: 'Functions' with the right mouse button.
- Anchor the opened window by clicking on the drawing pin on the desk top (see right)



- With LabVIEW **7**

  In the menu of the block diagram window select the option 'Tools ► Advanced ► Edit Palette Views'

- With LabVIEW **5**

  In the menu of the diagram window select the option 'Edit ► Edit Control & Function Palettes'

- With LabVIEW **6**

  In the window 'User Controls' click on the symbol 'Options' with the left mouse button.

  In the following window press the button 'Edit Palettes ..'

- Click on the grey field of the window 'User Libraries' with the right mouse button and then select the menue item: 'Insert ► Submenu'.

- In the window "`Insert Submenu`" select the setting "`Link to an existing menu file`" and confirm by pressing the OK button.

- Select in the file selection box the directory `ADwin_v2.lib` and the file `ADwin.mnu`.

- Save the changes by pressing the button "`Save Changes`" in the window "`Edit Control and Functions Palettes`".

- Activate the white diagram window once more.

- All ***ADwin*** VIs now appear in the group '`User Libraries`' (see at right) and you can directly drag them to your LabVIEW diagrams. The VIs are described in the following chapter.

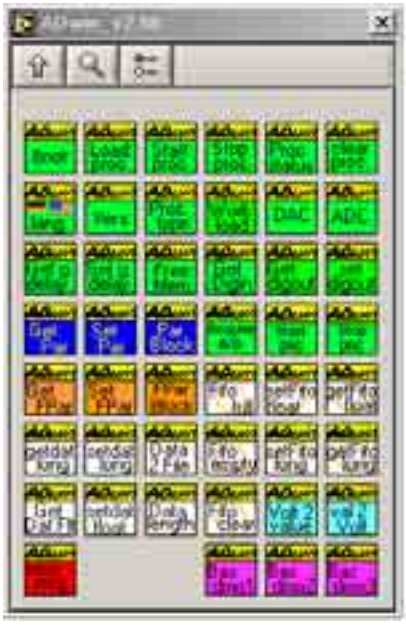
3.4 Porting of VIs between MS Windows and Linux

In order to port VIs from Windows to Linux versions of LabVIEW and vice versa, the path name of the the ***ADwin*** library has to be changed.

Operating system	Windows	Linux
Library path name	`C:\Windows\ADwin32.dll` oder `C:\WinNT\ADwin32.dll`	`/usr/lib/libADwin.so`

4 LabVIEW VIs for *ADwin* systems

4.1 Basic features



The VI icons have different colors, which are partially related to the LabVIEW standard. The colors in particular:

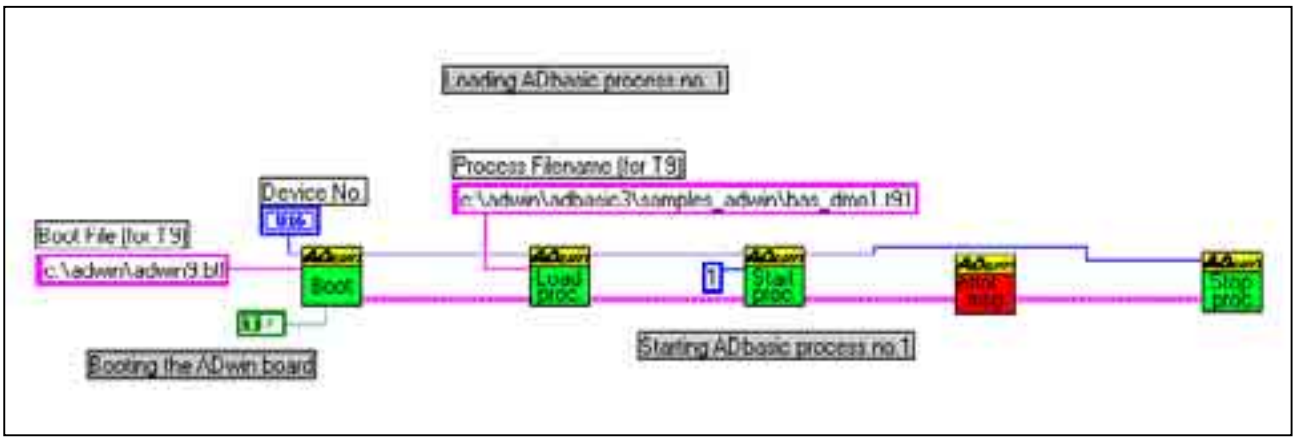
Green	:	System and "low level" functions
Blue	:	Integer parameter functions
Orange	:	Floating point parameter functions
Red	:	Error functions
White	:	DATA functions
White with ring	:	FIFO functions (ring buffer)
Light blue	:	Tools
Pink	:	Demos

The diagram below shows a programming example for LabVIEW illustrating the following notes. The function of the program is not important at this place (the example program boots the ***ADwin*** system (here: ADSP/T9), loads and starts a process, checks errors and stops the process again).

You have to connect all *ADwin*-VIs with "error in" and "error out" using the error wiring. Watch the dotted bottom line (pink), which connects all VIs: This is the error wiring. Without, LabVIEW can get into an endless loop if an error occurs and you will have no longer access to LabVIEW. With error wiring LabVIEW remains stable and you will get an error message in addition (with the VI *error_msg*). While wiring the VI's you will even fix them to be processed in definite order.

A unique "Device No." identifies any ***ADwin*** system and is needed for correct communication with it. Therefore you have to transfer the "Device No." to nearly all VIs as an input value. For easier use, the VIs have an output "Device No. out", which you can connect with the next VI input "Device No. in", thus generating a continuous connection (in the example it is the upper blue line). Otherwise you have to allocate the "Device No." separately to each VI.

For all VIs you can use a context help, which you can start with [`Strg`]+[`H`]. When you use the help and move the mouse to a VI symbol the corresponding help text is displayed (in English), consisting of a graphical representation of the VI and a help text (the help text can also be found in this manual, but not the graphic representation).



4.2 LabVIEW driver before version 5.0

If you use a LabVIEW driver before version 5.0, you may use its VIs further on, but we do not recommend it. The current driver uses revised VI names, is adapted for better use with LabVIEW 5/6/7 and enables easy error handling.

On our CD-ROM you will find the previous as well as the new VI collection (see 'driver installation').

The following description applies to the new nomenclature.

4.3 Error handling

This driver version provides a new possibility to handle errors: "error wiring".

It is characteristic for all new ***ADwin*** VIs to take a given error signal from the input "error in" and transfer it to the output "error out", without accessing the ***ADwin*** system. Thus, LabVIEW avoids to establish contact to the ***ADwin*** system if previously an error had occurred.

Please use the error wiring consequently: Connect all VIs with each other via the "Error in" and "Error out" connections. Use the error messages to handle the errors, by querying them with *error_msg* (e.g. once per loop cycle).

4.4 The "DeviceNo."

All ***ADwin*** systems are accessed via a unique "DeviceNo". The "DeviceNo" to the ***ADwin*** system is allocated by the program ***ADconfig***.

In LabVIEW each of the VIs has to know the "Device No.". It is enough to connect the "Device No." with the first VI and to loop through the wiring respectively (just like the error wiring).

4.5 Example programs

The example programs give a detailed description of the cooperation between LabVIEW and the ***ADwin*** system. They each consist of a LabVIEW and an ***ADbasic*** program. The LabVIEW program gives you control to the ***ADwin*** processes and displays the transferred data, whereas the ***ADbasic*** program determines the processing on the ***ADwin*** system.

The source code of the ***ADbasic*** programs can be found under
<C:\ADwin\ADbasic3\samples_ADwin*.BAS>, the LabVIEW programs have the same path
<..\bas_dmo.VI>.

For better understanding, please have a look at the source codes as well as to the corresponding LabVIEW diagrams.

4.6 Functions (VIs) for the control of *ADwin* systems

4.6.1 Load operating system

Boot.VI

Parameters:

ADwin type	Filename
ADwin-2	ADwin2.btl
ADwin-4	ADwin4.btl
ADwin-5	ADwin5.btl
ADwin-8	ADwin8.btl
T9 (*ADwin*-ADSP)	ADwin9.btl
T10 (*ADwin*-ADSP)	ADwin10.btl

Memory size	Memsize
64 KB	*0*
1 MB	*1*
2 MB	*2*
4 MB	*3*
8 MB	*4*
16 MB	*5*
32 MB	*6*

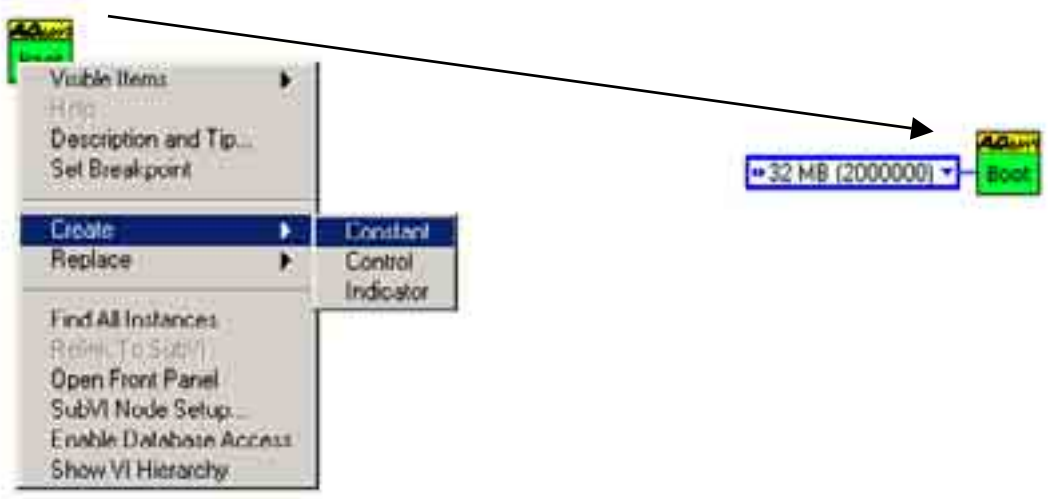
Return value memory:

	ADwin-ADSP (T9, T10)	*ADwin-2*, -4, -5 and -8
<1000	error	error
8000	OK	
Recognized memory expansion		OK

Showerror *true*: show message window in case of error

Note:

- When using a T9 or T10 processor (ADSP), the parameter "*MemSize*" is ignored.
- With standard installation fou find the files <ADwin2.btl> ... <ADwin9.btl> in directory <C:\ADwin\..>.
- You set the memory size (*Memsize*) using right mouse click on connection. Then select "Create ► Constant". In the appearing window you can manually select the memory expansion (see below).



4.6.2 Load a process

Load_Process.VI

Parameter:

Filename	name of the binary file to be loaded.
Showerror	*true*: show message window in case of error

Return value:

1	OK
unequal to 1	error

The VI loads a process (binary file), which was generated by ***ADbasic*** with the function '*Make Bin File'* .

4.6.3 Start a process

Start_Process.VI

Parameter:

ProcessNo	number of process to be started (1...10).

Return value:

255	error
unequal to 255	OK

4.6.4 Stop a process

Stop_Process.VI

Parameter:

ProcessNo	number of process to be stopped (1...10).

Return value:

255	error
unequal to 255	OK

4.6.5 Clear a process

This function can only be used by ***ADwin*** systems with a T9 or T10 (ADSP) processor. The function is not available under Linux.

Clear_Process.VI

Parameter:

ProcessNo	number of process to be cleared: 1…12, 15

Return value:

1	error
unequal to 1	OK

This function may clear the standard processes 11, 12 and 15 as well as the user defined processes 1…10.

The processes 11 and 12 are standard measurement processes which may be used by – for instance – the development environments TestPoint and LabVIEW for simple applications. Process 15 starts blinking the LED at the ***ADwin-Gold*** and ***ADwin-Pro*** systems. By clearing the processes 11, 12 and 15 the size of available program memory will increase.

Before you clear a process you have to

1. stop the process (see: ***Stop_Process***)
2. check, if the process has really stopped (see: ***Process_Status***).

Only then are you allowed to delete the process from memory!

4.6.6 Read process status

Process_Status.VI

Parameter:

ProcessNo	number of process (1...10) whose status is to be determined

Return value:

0	process is not active
unequal to 0	process is active

4.6.7 Set parameter Globaldelay

Set_Globaldelay.VI

Parameter:

ProcessNo	number of process (1...10) where *Globaldelay* is to be set.
Globaldelay	*Globaldelay* value to be set (see below).

Return value:

255	error
unequal to 255	OK

By setting *Globaldelay* you set the time interval between two events of a time-controlled process (see ***ADbasic*** manual).

When using an ADSP (T9, T10), the *Globaldelay* is set in steps of 25 ns at processes with high priority and in steps of 100 µs at processes with low priority.

When using T2, T4, T5 and T8 processors, the *Globaldelay* is set in steps of 1 µs at processes with high priority and in steps of 64 µs at processes with low priority.

4.6.8 Read parameter Globaldelay

Get_Globaldelay.VI

Parameter:

ProcessNo number of process (1...10) where *Globaldelay* is to be read

Return value:

The currently set *Globaldelay* value (in case of error: 255)

For more information see ***Set_Globaldelay*** or ***ADbasic*** manual.

4.6.9 Set an integer parameter

Set_Par.VI

Parameter:

Index	Meaning
1	PAR_1
2	PAR_2
…	...
80	PAR_80

Value new value for the selected integer parameter

Return value:

255 error
unequal to 255 OK

4.6.10 Set float parameter

Set_Fpar.VI

Parameter:

Index	Meaning
1	FPAR_1
2	FPAR_2
…	...
80	FPAR_80

Value new value for the selected float parameter

Return value:

255 error
unequal to 25 OK

4.6.11 Read integer parameter

Get_Par.VI

Parameter:

Index	Meaning
1	PAR_1
2	PAR_2
…	...
80	PAR_80

***Parameter value*:**

Current value of the selected integer parameter (in case of error: 255)

4.6.12 Read several integer parameters

Get_Par_Block.VI

Parameter:

Startindex position, where the block is read first (see below)
Count number of parameters to be read

Data array of the specified parameter

***Return value*:**

0 OK
unequal to 0 error

Startindex determines the element 0 of the array: If you start with *Startindex = 1*, then element 0 of the DATA array is set to PAR_1; with *Startindex = 5*, element 0 of the array is set to PAR_5.

4.6.13 Read float parameter

Get_Fpar.VI

Parameter:

Index	Meaning
1	FPAR_1
2	FPAR_2
…	...
80	FPAR_80

***Return value*:**

Current value of the selected FLOAT parameter (in case of error: 255)

4.6.14 Read several float parameters

Get_FPar_Block.VI

Parameter:

Startindex	position, where the block is read first (see below)
Count	number of parameters to be read
Data	array of the specified parameters

***Return value*:**

0	OK
unequal to 0	error

Startindex determines the element 0 of the array: If you start with *Startindex = 1*, then element 0 of the DATA array is set to PAR_1; with *Startindex = 5*, element 0 of the array is set to PAR_5.

4.6.15 Transfer data set into long or float array

GetData_Long.VI
GetData_Float.VI

Parameter:

DataNo	data set number (1...200)
Startindex	index of the first element to be transferred
Count	number of elements to be transferred
Data	data array

***Return value*:**

255	error
unequal to 255	OK

With the FLOAT instruction you can also read LONG data from the ***ADwin*** system, which will then be converted to a 32 bit FLOAT format.

With the LONG instruction you can also read FLOAT data from the ***ADwin*** system, which will then be converted to a 32 bit LONG format.

4.6.16 Transferring a long or float array as data set

SetData_Long.VI
SetData_Float.VI

Parameter:

DataNo	data set number (1...200)
Startindex	index of the first element to be transferred
Count	number of elements to be transferred
Data	array to be transferred

***Return value*:**

255	error
unequal to 255	OK

With the FLOAT instruction you can also write LONG data to the ***ADwin*** system, which will then be converted to a 32 bit FLOAT format.

With the LONG instruction you can also write FLOAT data to the ***ADwin*** system, which will then be converted to a 32 bit LONG format.

4.6.17 Save data set to file

Data2File.VI

Parameter:

Mode	Meaning
0	file will be overwritten
1	if the file already exists, the data will be attached

Filename	name of the file where data are saved
DataNo	data set number
Startindex	index of the first element to be saved
Count	number of the elements to be saved

Return value:

0	OK
unequal to 0	error

4.6.18 Get length of a data set

Data_Length.VI

Parameter:

DataNo	data set number

length:

0	error or data set does not exist
unequal to -1	length of the data set

Information about FIFO data set:

The following functions have direct access to the FIFO data sets. With ***ADbasic***, data sets (`DATA`) can be defined as FIFO (**F**irst **I**n **F**irst **O**ut) ring buffer. The elements of a FIFO data set are read in the same order as they were written into the FIFO.
The FIFO data sets have to be declared under ***ADbasic*** (see ***ADbasic*** manual).

4.6.19 Transfer FIFO data to integer or float array

Before calling this function, make sure that you have enough elements in the FIFO with the function FIFO_FULL.

GetFifo_Long.VI
GetFifo_Float.VI

Parameter:

FifoNo	FIFO data set number
Count	number of elements to be transferred
Data	array to be read

***Return value*:**

255	error
unequal to 255	OK

With the FLOAT instruction you can also read LONG data from the ***ADwin*** system, which will then be converted to a 32 bit FLOAT format.

With the LONG instruction you can also read FLOAT data from the ***ADwin*** system, which will then be converted to a 32 bit LONG format.

4.6.20 Write integer or float array data into FIFO

Before calling this function, make sure that you have enough space in the FIFO with the function FIFO_ EMPTY.

SetFifo_Long.VI
SetFifo_Float.VI

Parameter:

FifoNo	FIFO data set number
Count	array, whose values are transferred to the FIFO
Data	number of elements to be transferred

Return value:

255	error
unequal to 255	OK

With the FLOAT instruction you can also write LONG data to the ***ADwin*** system, which will then be converted to a 32 bit FLOAT format.

With the LONG instruction you can also write FLOAT data to the ***ADwin*** system, which will then be converted to a 32 bit LONG format.

4.6.21 Get number of elements in the FIFO

Fifo_Full.VI

Parameter:

FifoNo	number of the data set declared as FIFO

Count (= return value):

255	error
unequal to 255	number of elements written into the FIFO

4.6.22 Get number of empty FIFO elements

Fifo_Empty.VI

Parameter:

FifoNo	number of the data set declared as FIFO

Return value:

255	error
unequal to 255	number of empty elements in FIFO

4.6.23 Clear FIFO

Fifo_Clear.VI

Parameter:

FifoNo	number of the data set declared as FIFO

***Return value*:**

255	error
unequal to 255	OK

4.7 System functions

4.7.1 Get workload of the *ADwin* system

Workload.VI

Parameter:

Priority	is not yet supported

Workload (= return value):

current processor workload in % (in case of error: 255).

4.7.2 Test if operating system is loaded properly

Test_Version.VI

Return value:

0	OK
unequal to 0	system is not loaded

4.7.3 Get free memory

Free_Mem.VI

Parameter:

Mem_Spec	Memory type
0	program memory and data memory in a T2, T4, T5 or T8 processor
1	program memory (PM) integrated in an ADSP (T9, T10) processor
2	processor-internal data memory (DM) in an ADSP (T9, T10) processor
3	external data memory (SDRAM) in an ADSP (T9, T10)

Return value:

255	error
equal to 255	free memory in bytes

4.7.4 Output of error message

Error_msg.VI

Parameter:

Show Error Dialog ?	*true:* error message is displayed *false:* no error message is displayed

Return value:

Error ?	becomes *true* if errors occur

This VI is very suitable for displaying errors and can also be used for stopping a loop.

4.7.5 Set language of error messages

Set_Language.VI

Parameter:

0	Use the language set under Windows. While only German and English are available other languages will result in usage of English, too.
1	language is English
2	language is German

4.7.6 Get processor type

Processor_Type.VI

Return value:

Value	Meaning
0	error
2	T2
4	T4
5	T5
8	T8
9	T9
1010	T10

4.8 Functions only to be used under LabVIEW

4.8.1 Convert voltage to value

Volt2Value.VI

This utility VI helps to obtain a defined voltage at an analogue output. From a given voltage in [Volt], the VI calculates the corresponding digital value. This digital value can then be given to a DAC (using **DAC.VI**) to generate the voltage at the output.

If needed you may connect a single value, an 1D array or a 2D array to the input and output of the VI. You can set *Converter resolution* and *Voltage range* according to the ***ADwin*** system you use.

4.8.2 Convert value to voltage

Value2Volt.VI

Being the counterpart to the previous VI, this utility calculates the voltage in [Volt] from a digital value, e.g. given by **ADC.VI**.

If needed you may connect a single value, an 1D array or a 2D array to the input and output of the VI. You can set *Converter resolution* and *Voltage range* according to the ***ADwin*** system you use.

4.8.3 Read several analogue values

This function should only be used for a quick test and not together with an ***ADbasic*** process.

Acquire.VI

Parameter:

Sampling rate	Sampling rate in Hz (1 Hz…30000 Hz)
Processnumber	Measurement process number (1...4)
No. of samples	Number of the values to measure (2...32000)
Channel List	Array for the channels to measure, 1 = measure channel Array index+1: Input channel number (1...12)

4.8.4 Start cyclic analogue output of a buffer

This function should only be used for a quick test and not with an ***ADbasic*** process.

StartDAC.VI

Parameter:

Sample Rate	Output clock rate in Hz (1 Hz…100 000 Hz)
Process	D/A process number (1 or 2)
Values	Array of the values to output
Channel List	Array for the output channels, 1 = output channel, Array index+1: Output channel number (1...6)
Repeat	Number of the repeats of *Values*

4.8.5 Stop cyclic analogue output

This function should only be used for a quick test and not with an ***ADbasic*** process.

StopDAC.VI

Parameter:

Process	D/A process number (1 or 2)

5 Index of Commands

Revisionen

Datum/Version	Name	Revision	Info
bis 01 04	MH		
01 05 03	HR	neu: Kap- 2.4 Portierung MS WIN und Linux	MB
01 05 04	MH	Kap. 2.1 ersetzt Kap. 2.3 überarbeitet (Differenzierung zw. LabVIEW6 u. allen vorigen Versionen) Bei allen Abbildungen Linkadresse durch Device No. ersetzt Neue VIs: Get_Par_Block Get_FPar_Block GetiFiFoPart GetfFiFoPart Fifo_Full Get_Last_Error Get_Last_Error_Text	MH
28 05 01 **Version 5/01**	HR	Hinweis am Ende des Kap 3.5 entfällt Neues PDF	MH
19 02 02	TN	Neue Treiberversion v2, in Anlehnung an Delphi-Treiber (deutsch) VIs gelöscht: Get_Last_Error, Get_Last_Error_Text	MH
22.02.02 **Version 1.6**	GC	Konzept angepasst, Sprache und Format überarbeitet (deutsch)	
25.02.02	RRa	Formatierungen angepasst; Fehler bereinigt (deutsch)	
11.03.02	AKk	Übersetzung in englisch	
12.03.02	GC	Neues PDF englisch	PDF
9.4.02	GC	Versionsnummer 04/02	PDF
9.7.02	GC	Befehle DAC, ADC, Get_Digin, Set_digout, Get_digout entfernt, da sie nur noch über ADbasic angesprochen werden sollen.	PDF Juli 02
11.7.02	GC	Kap. 4.3: Ersetze "neue VI-Funktionsnamen" sinngemäß durch "Umgang mit älteren Treiberversionen"	
27.8.02	GC	Data_Length: Rückgabewert von "-1" in "0" korrigiert	
20.10.03	GC	Clear_Process: Neuer Hinweis „nicht verfügbar unter Linux“	
22.10.03	GC	Informationen für T10 ergänzt. Einbinden der VIs für Version 7 eingefügt. Eignung für LabView 7 ergänzt.	PDF Okt. 03