JÄGER
Computergesteuerte
Messtechnik GmbH

ADwin

Driver for LabVIEW 2009 and later

ADwin driver LabVIEW, Manual Apr. 2023

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

Hotline:	+49 6251 96320
Fax:	+49 6251 963299
E-Mail:	info@ADwin.de
Internet	www.ADwin.de

JÄGER
Computergesteuerte
Messtechnik GmbH

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

Table of contents

Typographical Conventions

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

You find a "note" next to

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

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

<C:\ADwin\ …>

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

Program text

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

Var_1

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

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

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

1 Information about this Manual

This manual contains comprehensive information about the *ADwin*-LabVIEW® driver for LabVIEW® version 2009 and later.

The driver is not applicable for LabVIEW versions 8 or before. If necessary, please contact our hotline.

Additional information are available in

– the manual "*ADwin* Driver Installation". Begin your installation here.

– if you work with Linux or Mac OS: the manual "ADwin for Linux / Mac", which describes the software installation and the *ADbasic* compiler usage from Linux and Mac OS.

– the manuals "*ADbasic*", "*ADsim*" und "*ADwinC*". Each program package enables you to program your *ADwin* hardware and have *ADwin* processes run on it.
  - *ADbasic* comprises the comfortable real-time development environment and the instructions of the *ADbasic* compiler.
  - The package *ADsim* makes Simulink® models executable on *ADwin* systems.
  - With ADwinC for Visual Studio, you can develop *ADwin* processes in the language C.

– the hardware manuals for the *ADwin* systems you are using.

It is assumed that you are familiar with your LabVIEW® environment.

Please note:

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

Qualified personnel

The manufacturer of the systems being described in this manual demands that only qualified personnel will work with the provided equipment.

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

This documentation, including all pictures is protected by copyright. Reproduction, translation as well as electronical and photographical 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.

Subject to change.

Hotline address: see inner side of cover page.

2 *ADwin*-LabVIEW driver

More features for LabVIEW with *ADwin*

The *ADwin* system consists of an independent on-board CPU, which executes measurement and control tasks very fast and reliably, as well as of an interface under Windows or Linux 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 determine the behaviour of the *ADwin* hardware on your own. There are the following ways to do it:

- *ADbasic*: You program real-time processes using the development environment *ADbasic*, create a binary file and transfer it to the *ADwin* system (see *ADbasic* manual or online help).
- *ADwinC*: You program real-time processes in Visual Studio with the programming language C, using the program package ADwinC for Visual Studio, You create a binary file and transfer it to the *ADwin* system (see *ADwinC* manual).
- *ADsim*: You create a model in Simulink, export it and compile the model with *ADsimDesk* for the *ADwin* hardware (see *ADsim* manual).

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.

Alternatively with *ADsim*, you can export a Simulink model and run it as process on *ADwin* hardware (see *ADsim* manual).

Controlling *ADwin* systems with LabVIEW

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

The sections 2.1 and 2.2 explain how LabVIEW and *ADwin* communicate with each other and deepens your knowledge for the *ADwin* concept.

In chapter 3, the installation and the integration of the new commands are described.

The general use of the LabVIEW drivers is explained in chapter 4, its functions in chapter 5, which can also be used as a kind of reference documentation.

2.1 Interface for the development environment

The *ADwin*-LabVIEW driver is an interface for the development environment LabVIEW from version 2009 used for the communication with *ADwin* systems.

The combination of LabVIEW® and an *ADwin* system offers you totally new possibilities. The most fast reaction and computing power of an *ADwin* system on the one hand and the LabVIEW functions for managing, analysis and documentation of measuring values on the other hand join into a powerful concept.

Typical applications are:

– Control of test stands
– Generating signals
– Measure with intelligence, collect data under complex trigger conditions
– Open-loop and closed-loop control
– Online processing, data reduction
– Hardware-in-the-Loop, simulation of sensor signals

2.2 Communication with the *ADwin* system

With the development environment, you can control processes in the *ADwin* system, as well as getting data from there or sending data. You are programming processes with the real-time development tool *ADbasic*, create a binary file and transfer it to the *ADwin* system (see *ADbasic* manual or online help).

Data and instructions between LabVIEW® and the *ADwin* system are processed according to the following illustration.

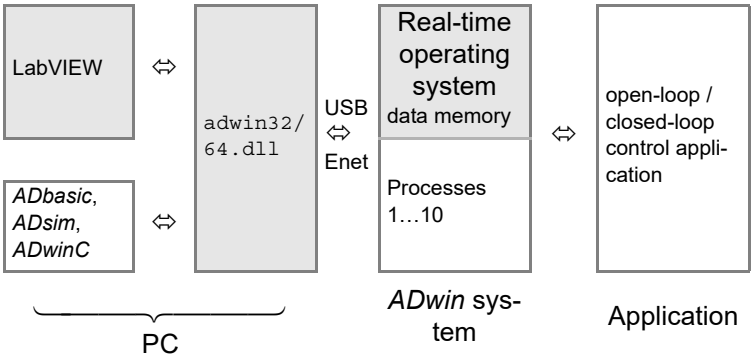


Fig. 1 – *ADwin*-LabVIEW interface

adwin32/64.dll

The `adwin32.dll` (64 bit: `adwin64.dll`) is the only interface for Windows applications to the *ADwin* system and is therefore used by *ADwin* LabVIEW driver, too. With this interface, several Windows programs can communicate with the *ADwin* system at the same time: Thus, the development environments, *ADbasic*, and *ADtools* can work with the *ADwin* system simultaneously.

Real-time operating system

The interface `adwin32/64.dll` communicates with the real-time operating system of the *ADwin* device. Therefore, you must load the operating system (e.g. the file `<adwin12.btl>`) after each power-up. After a successful loading the system will be able to receive and execute processes, receive instructions from the PC and exchange data with it. The processes programmed in *ADbasic*, include the program code for measurement, open-loop or closed-loop control of your application.

The real-time operating system performs the following tasks:

10 processes

– Management of up to 10 real-time processes with low or high priority (selectable). Processes with low priority can be interrupted by processes with high priority, the latter cannot be interrupted by other processes.
  With *ADsim*, there is only a single real-time process of high priority, which is the compiled Simulink model; as an option, an additional low priority process can belong to the model.

Data memory

– Providing global variables:
  - 80 integer variables (`Par_1` ... `Par_80`), predefined
  - 80 floating-point variables (`FPar_1` ... `FPar_80`), predefined
  - 200 arrays (`Data_1` ... `Data_200`), length and data type can be set individually

  You can read and change the values of these variables or data arrays from the development environment at any time.

Communication

– Communication between *ADwin* system and PC (`adwin32/64.dll`).

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

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

2.3 *ADsim* driver for LabVIEW

As an addition to the *ADwin* driver for LabVIEW Simulink users can also use the *ADsim* driver for LabVIEW.

The *ADsim* driver works only under Windows.

While the *ADwin* driver for LabVIEW enables you to access *ADwin* variables of the Simulink model, the *ADsim* driver provides access to all other variables and arrays of the Simulink model, i.e. parameters, signals and block states. Both drivers can be used independently from each other. Please note that the drivers use a very different error handling.

We recommend preferrably using the *ADwin* driver for LabVIEW. In comparison, it provides several advantages:

– The driver supports continuous data flow via Fifo arrays.
– Compiled Simulink models can be transferred to *ADwin* hardware using the `Boot` instruction.
– The driver runs under Linux and Mac OS.

ADsim users can use all functions of the *ADwin* driver for LabVIEW except for the following:

– Free_Mem.VI
– String_Length.VI, SetData_String.VI, GetData_String.VI

Users of *ADsim* T11 must furthermore do without the following functions:

– Load_Process.VI, Start_Process.VI, Stop_Process.VI, Clear_Process.VI, Process_Status.VI

3 Installing the LabVIEW Driver

Please follow the installation steps described below in order to get easy access to your *ADwin* system from LabVIEW®.

3.1 Do the "*ADwin* driver installation"

For the installation you need an up-to-date *ADwin* software package.

3.1.1 Installation under Linux or Mac

Please follow the installation guide in the manual "ADwin Linux / Mac". Please pay attention to installing the archive `adwin-labview-x.y.tar.gz` at last.

After successful installation you will find the files in folders below `</opt/adwin/share>` (standard installation):

VI collection and LabVIEW examples	`./ADwin_v3.2.lib`
VI collection for older LabVIEW versions	`./ADwin_v2.2.lib`
Examples for *ADbasic*	`./samples_ADwin`

Continue with chapter 3.2 "Including the ADwin-VIs".

3.1.2 Installation under Windows

If *ADwin* is installed yet

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

Else: New installation

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

- install the software from the *ADwin* software package .
- Install the communication driver under Windows.
- Install the hardware in the PC (if necessary) and set up the hardware connections between PC and *ADwin* system.

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

VI collection and LabVIEW examples	`.\Developer\LabVIEW\ADwin_v3.2.lib`
VI collection for older LabVIEW versions	`.\Developer\LabVIEW\ADwin_v2.2.lib`
Examples for *ADbasic*	`.\ADbasic\samples_ADwin`
Test program for *ADwin-Gold*, *ADwin-light-16* and plug-in boards	`.\Tools\Test\ADtest`
Test program for *ADwin-Pro*	`.\Tools\Test\ADpro`

Continue with the next chapter "Including the ADwin-VIs".

3.2 Including the *ADwin*-VIs

In LabVIEW programs, ready-made VIs will send instrudctions and data to an *ADwin* system and read data from the system. The VIs will use functions, which are implemented in the interface (see chapter 2.2).

Include the VIs into LabVIEW as follows:

– Copy the folder `<ADwin_v3.2.lib>` from the installation folder into the LabVIEW folder, under Windows e.g. `<C:\Programs\National Instruments\LabVIEW 2021\user.lib>`.

  You are already done.

– If you move the mouse in LabVIEW over the icon `ADwin_v3.2.lib` in the group `User Libraries`, all VIs appear in a new windows (see right).

  You can directly drag the VIs from the group into your LabVIEW diagram. The VIs are described in chapter 5.

3.3 Converting *ADwin* Vs

The *ADwin* VIs are stored from LabVIEW 2009. If you use a later LabVIEW version, National Instruments® recommends converting the VIs in order to have LabVIEW use less memory resources and avoid large run-time degradation of performance for the VIs.

Find more advice in the LabVIEW manual from National Instruments® under „Converting VIs“; to convert a directory of VIs look under the keyword "mass compile".

4 General about *ADwin* VIs

4.1 Basic features

The colors of the VI icons show their function. The colors are partially related to the LabVIEW standard:

Green:	System information and process control
Blue:	Transfer of global long variables
Orange:	Transfer of global float variables
Red:	Error function
White:	Transfer of Data arrays
White + ring:	Transfer of Fifo arrays (ring buffer)
White + frame:	Transfer of string data
Light blue:	Tools
Pink:	Demos

Please note the following information for the use of *ADwin* VIs:

– Connect all *ADwin*-VIs with "`error in`" and "`error out`" using the error wiring (see chapter 4.3).

– Give a "Device No." (see chapter 4.4) as input value to the VIs. Only few VIs do not need a "Device No.".

– For every VI there is a context help window, which is started with `[Strg]`+`[H]`.

  When you move the mouse to a VI symbol with the help window opened a help text and a graphic of the VI are displayed. The help text can also be found in this manual, but not the graphic.

4.2 Using a Driver of LabVIEW 4…8

If you use an older driver for LabVIEW 4…8, you may use its VIs further on in 32-bit program versions, but we do not recommend it. With 64-bit program versions, older VIs are not applicable.

If you work with a LabVIEW version 6…8, use the VI collection 2.2. The manual is found in the appropriate folder, see chapter 3 on page 5.
For drivers of LabVIEW versions 4…5 please refer to our support; you find the address on the inner side of the cover page of the manual.

In comparison to older versions, the current driver provides the following changes:

– supports LabVIEW with 32 bit and 64 bit

– enhanced help

– several new VIs

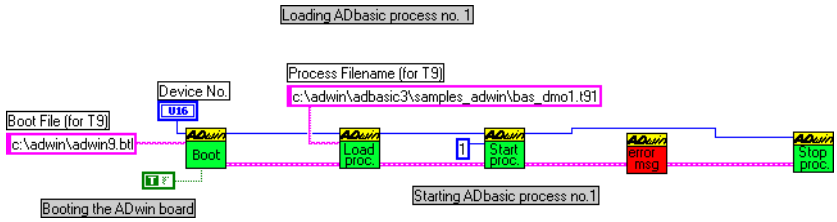
– default value of Device No. is 1 (see chapter 4.4)

– obsolete VIs have been dropped

4.3 Error handling

This driver version provides "error wiring" as the possibility to handle errors. Furthermore, the error wiring defines the order LabVIEW will process the VIs.

The diagram below contains a dotted line (pink) connecting the bottom of the VIs: This is the error wiring. The error input of the first VI is left unconnected.



If an error occurs in an *ADwin*-VI, the VI sends an error signal via the error wiring to the following VIs, which then stop accessing the *ADwin* system. Using the VI `error_msg`, an error message can be generated.

The error wiring is to be used consequently:

– Connect all VIs in a line via the connectors "`error in` " and "`error out`".

– Query occuring error signals with the VI `error_msg` to establish a simple error handling, e.g. once in a loop.

Please note: For reasons of compatibility with previous versions the most VIs return in case of an error a defined value on a data output. With some VIs (Free_Mem.VI, Get_Processdelay.VI, Get_Par.VI, Get_FPar.VI, Get_FPar_Double.VI, Fifo_Empty.VI, Fifo_Full.VI), this value can either be a valid data value. The meaning of the return value is therefore not clearly defined.

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, nearly all of the VIs have to know the "Device No.". Connect the Device No. to the first VI. Connect the output "Device No. out" to the next Vi and this way create a complete wiring (see example in "Error handling", upper blue line).

4.5 Invalid floating-point values from ADwin hardware

With 32-Bit floating-point values until processor T11, bit patterns of invalid values in the *ADwin* hardware are converted during transfer to the PC into different values, see following table. Numbers inside the valid value range (normalized numbers) stay unchanged.
With processor T12, the IEEE denominations as `#INF` are displayed.

IEEE denomination	Bit pattern area	Value or display on the PC
+0	`00000000h`	0
Positive denormalized numbers	`00000001h` `007FFFFFh`	0

IEEE denomination	Bit pattern area	Value or display on the PC
Positive normalized numbers	00800000h 7F7FFFFFh	+1,175494 · 10-38 +3,402823 · 10+38
+∞ (Infinity, #INF)	7F800000h	3.402823E+38
Signaling Not a number (SNaN)	7F800001h 7FBFFFFFh	3.402823E+38
Quite Not a number (QNaN)	7FC00000h 7FFFFFFFh	3.402823E+38
-0	80000000h	0
Negative denormalized numbers	80000001h 807FFFFFh	0
Negative normalized numbers	80800000h FF7FFFFFh	-1,175494 · 10-38 -3,402823 · 10+38
-∞ (Infinity, #INF)	FF800000h	3.402823E+38
Signaling Not a number (SNAN)	FF800001h FFBFFFFFh	3.402823E+38
Indeterminate	FFC00000h	3.402823E+38
Quite Not a number (QNAN)	FFC00001h FFFFFFFFh	3.402823E+38

4.6 Example programs

The example programs show the cooperation between LabVIEW and the *ADwin* system. Every example consists of a LabVIEW and an *ADbasic* program:

– The LabVIEW program gives you control to the *ADwin* processes and displays the transferred data.

– The *ADbasic* program determines the processing on the *ADwin* system.

The LabVIEW programs are stored in the installation directory (see chapter 3.1), the *ADbasic* programs can be found in the folder

– Windows: <C:\ADwin\ADbasic\samples_ADwin\>.

– Linux: </opt/adwin/share/samples_ADwin/>.

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

5 Functions of the *ADwin*-LabVIEW® Driver

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

Please note the general notes on the use of *ADwin* VIs in chapter 4.

5.1 System control and information

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

Boot.VI

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

Inputs

Device No.	Device No. of the system.
Filename	Path and filename of the operating system file, depending on processor type: T2: `ADwin2.btl` T4: `ADwin4.btl` T5: `ADwin5.btl` T8: `ADwin8.btl` T9: `ADwin9.btl` T10: `ADwin10.btl` T11: `ADwin11.btl` T12: `ADwin12.btl` T12.1: `ADwin121.btl`
Memsize	Processor type T2…T8 only: Built-in memory size (64 KiB…32 MiB)
error in	Connector for error signal.
showError	true: show error message with active error signal. false: no error message.

Outputs

Device No. out	The number connected to the input „Device No.“.
Memory	Status information: <1000: Error while booting 8000: Booting o.k.; T9, T10, T11 only >8000: Booting o.k. and correct MemSize value; T2…T8 only. Possible values MemSize: 10000: 64 KiB 100000: 1 MiB 200000: 2 MiB 400000: 4 MiB 800000: 8 MiB 1000000: 16 MiB 2000000: 32 MiB
error out	Error signal

Notes

The boot process clears the whole memory of the *ADwin* system: all program codes are lost, global variables are set to 0 (zero) and the parameter `Processdelay` is set to 1000.

The PC will only be able to communicate with the *ADwin* system, after the operating system has been loaded. Load the operating system every time you power-up the *ADwin* system.

For users of *ADsim* with processor T11:

- As `Filename` you enter the Simulink model being compiled via *ADsimDesk*, which also contains the operating system for the processor. The model file is stored in the model folder in the subfolder `<model>_ert_rtw/ADwin/` with the name `<model>11c.btl`.
- `<model>` stands for the name of the Simulink model. The notation `11c` refers to the processor type T11 of the *ADwin* hardware.
- Please note that *ADbasic* processes and a compiled Simulink model (from *ADsim*) run on the *ADwin* hardware at the same time.

Loading the operating system successfully with "`Boot`" lasts only 1 second.

To set the memory size you do a right click on the `memsize` input and select "`Create` ▸ `Constant`". In the next window, you select the correct memory size.

Test_Version.VI

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

Inputs

Device No.	Device No. of the system.
error in	Connector for error signal.
showError	true: show error message with active error signal. false: no error message.

Outputs

Device No. out	The number connected to the input „Device No.“.

Version	0: OK ≠0: Operating system has wrong version or does not respond.
error out	Error signal

Sanitize_Floating_Point_Value.VI

`Sanitize_Floating_Point_Value.VI` sets the sanitize mode for NaN values in floating point arrays.

Inputs

flag	The flag value sets the sanitize mode: 0: NaN values pass through as they are, they are left unchanged. 1: The ADwin DLL converts NaN values in floating point arrays into regular values, usually into MAX_FLOAT (default setting).
error in	Connector for error signal.
showError	true: show error message with active error signal. false: no error message.

Outputs

error out	Error signal

Notes

The default setting is to convert NaN values into regular values (flag=1).

Processor_Type.VI

`Processor_Type.VI` returns the processor type of the system.

Inputs

Device No.	Device No. of the system.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Processor type	Processor type: 0: error 2: T2 4: T4 5: T5 8: T8 9: T9 1010: T10 1011: T11 1012: T12 10121:T12.1
error out	Error signal

Workload.VI

`Workload.VI` returns the processor workload.

Inputs

Device No.	Device No. of the system.
priority	0: Total processor workload ≠0: no function
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Workload	≠255: Processor workload in % 255: Error
error out	Error signal

Free_Mem.VI

`Free_Mem.VI` determines the free memory.

Inputs

Device No.	Device No. of the system.
MemType	Memory type: 0 : all memory types; T2, T4, T5, T8 only 1 : internal program memory (PM_LOCAL); T9…T11 2: internal data memory (DM_LOCAL); T9…T11 3: external DRAM memory (DRAM_EXTERN); T9…T11 4: internal data memory (EM_LOCAL); T11 only 5: Cacheable: Memory, which can provide data to the cache; T12/T12.1 only. 6: Uncacheable: Memory, which cannot provide data to the cache; T12/T12.1 only.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Memory	≠255: Currently free memory (in bytes). With `MemType` =5/6, the value is given in units of kB. 255: Ambigous, error or memory size.
error out	Error signal

Notes

Please note the details about error handling in chapter 4.3.

5.2 Process control

The control of *ADbasic* and *TiCoBasic* processes is different:

– ADbasic Processes

– TiCoBasic Processes

5.2.1 ADbasic Processes

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

Load_Process.VI

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

Inputs

Device No.	Device No. of the system.
Filename	Path and filename of the binary file to be loaded
error in	Connector for error signal.
showError	true: show error message with active error signal. false: no error message.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	1: OK ≠1:Error
error out	Error signal

Notes

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

The last character of the binary file’s filename shows the numberof the process in the *ADwin* system. A "0" stands for process 10.

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

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

Start_Process.VI

`Start_Process.VI` starts a process on the system.

Inputs

Device No.	Device No. of the system.
ProcessNo	Number (1…10) of the process.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255:OK 255:Error
error out	Error signal

Notes

The process starts with the priority it has been compiled with in *ADbasic*.

Stop_Process.VI

`Stop_Process.VI` stops a process on the system.

Inputs

Device No.	Device No. of the system.
ProcessNo	Number (1…10) of the process.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255:OK 255:Error
error out	Error signal

Clear_Process.VI

`Clear_Process.VI` deletes a process from memory.

Inputs

Device No.	Device No. of the system.
ProcessNo	Number (1…10, 15) of the process.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠1:OK 1: Error
error out	Error signal

Notes

This function is available only for systems with T9 or T10, T11 processor and not with Linux.

Loaded processes need memory in the system. You can delete processes from memory, in order to get more memory space; follow the order of the following steps:

1. stop the running process with `Stop_Process.VI`.
2. check if the process has stopped with `Process_Status.VI`.
3. delete the process from memory with `Clear_Process.VI`

Process 15 is responsible for the flashing of the LED in the Gold and Pro systems, after deleting it, the LED does not flash any more.

Processes should be cleared in reverse to the order they were loaded, i.e. het last loaded process is cleared first. This will avoid fragmentation of the program memory.

Please note, that clearing a process does NOT clear the arrays, which were declared in the process.

Process_Status.VI

`Process_Status.VI` returns the status of a process.

Inputs

Device No.	Device No. of the system.
ProcessNo	Number (1…10, 15) of the process.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	0: Process is stopped ≠0: Process is running 255: Error
error out	Error signal

Set_Processdelay.VI

`Set_Processdelay.VI` sets the parameter `Processdelay` for a process.

Inputs

Device No.	Device No. of the system.
ProcessNo	Number (1…10) of the process.
Processdelay	Value of `Processdelay` to be set.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

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

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

Processor type	Process priority	
	high	low
T2, T4, T5, T8	1000ns	64µs
T9	25ns	100µs
T10	25ns	50µs
T11	3.3ns	0.003µs = 3.3ns
T12	1ns	1ns
T12.1	1.5ns	1.5ns

Get_Processdelay.VI

`Get_Processdelay.VI` returns the parameter `Processdelay` for a process.

Inputs

Device No.	Device No. of the system.
ProcessNo	Number (1…10) of the process.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: Value of the parameter `Processdelay`. 255: Error or valid value for `Processdelay`.
error out	Error signal

Notes

See more information under `Set_Processdelay.VI` or in the *ADbasic* manual.

Please note the details about error handling in chapter 4.3.

For *ADsim* users: The parameter `Processdelay` corresponds to the fixed-step size in Simulink. While the fixed-step size is set in seconds, the `Processdelay` is a multiple of processor cycles.

5.2.2 TiCoBasic Processes

On an *ADwin* hardware with *TiCo* processor, you can transfer a *TiCoBasic* binary file as process to the *ADwin* hardware and start the process. You can use an *ADbasic* program, the development environment *TiCoBasic*, or LabVIEW.

To transfer a TiCoBasic process with LabVIEW, the following steps are necessary:

– Create a binary file with *TiCoBasic*.

  At first, the binary file must be transferred into a global array `Data_x` of the *ADwin* processor, where an *ADbasic* process copies it to the *TiCo* processor.

– Create an *ADbasic* process, which fulfills the following tasks:
  - Dimensioning a global array `Data_x` of data type `Long`. Please make sure that the array size exceeds the size of the *TiCoBasic* binary file.
  - Transferring the data from `Data_x` to the *TiCo* processor using the instruction **`TiCo_Load`** / **`P2_TiCo_Load`**. The process starts automatically.
  - Saving the instruction's return value to a global variable `Par_x`.
  - Exiting the *ADbasic* proces with `Exit`.

  You find example *ADbasic* proceses in the installation directory (see chapter 3.1) under
  `<.\ADbasic\samples_ADwin_GoldII>`
  `<.\ADbasic\samples_ADwin_ProII>`

– Create the binary file of the process in *ADbasic*.

– Do the following steps in LabVIEW:
  - Transfer the *ADbasic* binary file with `Load_Process` as process to the *ADwin* hardware, but do not start the process yet.
  - Transfer the *TiCoBasic* binary file with `Data2File` to the correct array `Data_x` of the *ADwin* processor.
  - Start the *ADbasic* process with `Start_Process`.
  - Read the global variable `Par_x` and check if transferring has been successful.

5.3 Transfer of global variables

Instructions for data transfer between PC and *ADwin* system with the standard global variables `Par_1` ... `Par_80` (long) and `FPar_1` ... `FPar_80` (float), which are also called parameters.

5.3.1 Global variables `Par_1...Par_80`

The global variables `PAR_1...PAR_80` on the *ADwin* system have the following range of values:

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

The VIs transfer values with 32-bit precision.

Set_Par.VI

`Set_Par.VI` sets a global variable `Par_1...Par_80` to the specified value.

Inputs

Device No.	Device No. of the system.
Index	Number (1…80) of a global variable `Par_1` ... `Par_80`.
Value	New value for the variable.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Get_Par.VI

`Get_Par.VI` returns the value of a global variable `Par_1...Par_80`.

Inputs

Device No.	Device No. of the system.
Index	Number (1…80) of a global variable `Par_1` ... `Par_80`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Parameter value	≠255: Value of the selected variable 255: Error or valid value for variable `Par`.
error out	Error signal

Notes

Please note the details about error handling in chapter 4.3.

Get_Par_Block.VI

`Get_Par_Block.VI` returns a specified number of consecutive global variables `Par` into an array.

Inputs

Device No.	Device No. of the system.
Startindex	Number (1…80) of the first global variable `Par_1` … `Par_80` to be transferred.
Count	Number of variables to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	0: OK ≠0: Error
Data	Array with the read values of data type `Long`.
error out	Error signal

Notes

The read values are saved in the array Data starting from array element 0. If e.g. Startindex = 5, the value of `Par_5` is saved in the element 0 of Data.

5.3.2 Global variables FPar_1...FPar_80

The global variables FPar_1...FPar_80 on the *ADwin* system have the following range of values, depending on the processor type:

- FLOAT until T11
  negative: $-3{,}402823 \cdot 10^{+38} \ldots -1{,}175494 \cdot 10^{-38}$
  positive: $+1{,}175494 \cdot 10^{-38} \ldots +3{,}402823 \cdot 10^{+38}$
- FLOAT64 with T12/ T12.1
  negative: $-1{,}7976931348623 15 \cdot 10^{+308} \ldots -2{,}2250738585072014 \cdot 10^{-308}$
  positive: $+2{,}2250738585072014 \cdot 10^{-308} \ldots +1{,}7976931348623 15 \cdot 10^{+308}$

Set_FPar.VI

Set_FPar.VI sets a global variable FPar to a specified value of data type Single.

Inputs

Device No.	Device No. of the system.
Index	Number (1…80) of a global variable FPar_1 ... FPar_80.
Value	New value for the variable of data type Single.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

Set_FPar always transfers a 32-bit float value even though FPar may have 64-bit precision.

Set_FPar_Double.VI

Set_FPar_Double.VI sets a global variable FPar to a specified value of data type Double.

Inputs

Device No.	Device No. of the system.
Index	Number (1…80) of a global variable FPar_1 ... FPar_80.
Value	New value for the variable of data type Double.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

With processors until T11, the destination variable on the ADwin system has single precision only.

Get_FPar.VI

`Get_FPar.VI` returns the `Single` value of a global variable `FPar`.

Inputs

Device No.	Device No. of the system.
Index	Number (1…80) of a global variable `FPar_1` … `FPar_80`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Parameter value	≠255: Value of the selected variableof data type `Single`. 255: Error or valid value for variable `FPar`.
error out	Error signal

Notes

Please note the details about error handling in chapter 4.3.

With processor T12/T12.1, `FPar` variables in the *ADwin* system have 64-bit precision. Nevertheless, the returned value has data type `Single`.

Get_FPar_Double.VI

`Get_FPar_Double.VI` returns the `Double` value of a global variable `FPar`.

Inputs

Device No.	Device No. of the system.
Index	Number (1…80) of a global variable `FPar_1` … `FPar_80`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Parameter value	≠255: Value of the selected variable of data type `Double`. 255: Error or valid value for variable `FPar`.
error out	Error signal

Notes

Please note the details about error handling in chapter 4.3.

Get_FPar_Block.VI

`Get_FPar_Block.VI` returns a specified number of consecutive global variables `FPar` into an array of data type `Single`.

Inputs

Device No.	Device No. of the system.
Startindex	Number (1…80) of the first global variable `FPar_1` … `FPar_80`, which is read.
Count	Number of variables, which are transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	0: OK ≠0: Error
Data	Array with the read values of data type `Single`.
error out	Error signal

Notes

The read values are saved in the array Data starting from array element 0. If e.g. Startindex = 9, the value of `FPar_9` is saved in the element 0 of Data.

With processor T12/T12.1, `FPar` variables in the *ADwin* system have 64-bit precision. Nevertheless, the returned values have data type `Single`.

Get_FPar_Block_Double.VI

`Get_FPar_Block_Double.VI` returns a specified number of consecutive global variables `FPar` into an array of data type `Double`.

Inputs

Device No.	Device No. of the system.
Startindex	Number (1…80) of the first global variable `FPar_1` … `FPar_80`, which is read.
Count	Number of variables, which are transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	0: OK ≠0: Error
Data	Array with the read values of data type `Double`.
error out	Error signal

Notes

Until T11, please note: floating-point values in the *ADwin* system have 32-bit precision.

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 strings

You must define each array under *ADbasic* before usage (see *ADbasic* manual).

5.4.1 DATA arrays

You must define each `Data` array under *ADbasic* before usage (see manual "*ADbasic*"): `DIM Data_x[n] AS LONG/FLOAT`.

You have to declare each array in *ADbasic* before usage (see manual "*ADbasic*"):
`DIM DATA_x AS LONG/FLOAT/FLOAT32/FLOAT64`

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

- `LONG` -2147483648 ... +2147483647
- `FLOAT` (until T11), `FLOAT32` negative: $-3{,}402823 \cdot 10^{+38}$ ... $-1{,}175494 \cdot 10^{-38}$; positive: $+1{,}175494 \cdot 10^{-38}$ ... $+3{,}402823 \cdot 10^{+38}$
- `FLOAT64` negative: $-1{,}7976931348623 15 \cdot 10^{+308}$ ... $-2{,}2250738585072014 \cdot 10^{-308}$; positive: $+2{,}2250738585072014 \cdot 10^{-308}$ ... $+1{,}7976931348623 15 \cdot 10^{+308}$

Data_Length.VI

`Data_Length.VI` returns the length of an *ADbasic* array of data type `Long`, `Float`, `Float32`, or `Float64`, that means the number of elements.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the Data array `Data_1`...`Data_200`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠0: Declared length of Data array 0: Error or Data array is not declared
error out	Error signal

Notes

To determine the length of a string in a `Data` array of type `STRING` you use the String_Length.VI.

SetData_Long.VI

`Set_Data_Long.VI` transfers `Long` values from a LabVIEW array into a `Data` array of the *ADwin* system.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1`...`Data_200`. `Data` may have data type `Long`, `Float`, `Float32`, or `Float64`.
Data	Array with the `Long` values to be transferred.
Startindex	Index of the first element in Data to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

All values of the LabVIEW array are transferred (from the start element). The `Data` array must be greater than the number of transferred values plus Startindex.

The `Long` values of the source array are automatically converted to the data type of the destination array on the *ADwin* system during transfer.

GetData_Long.VI

`GetData_Long.VI` transfers values from a `Data` array on an *ADwin* system into a LabVIEW vector of data type `Long`.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1`...`Data_200`.
Startindex	Index of the first element in the selected array to be transferred.
Count	Number of values to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
Data	Array with the read values of data type `Long`.
error out	Error signal

Notes

The values of the source array are automatically converted to data type `Long` during transfer.

SetData_Float.VI

`SetData_Float.VI` transfers `Single` values from a LabVIEW array into a `Data` array on the *ADwin* system.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1`…`Data_200`. `Data` may have data type `Long`, `Float`, `Float32`, or `Float64`.
Data	Array with `Single` values to be transferred.
Startindex	Index of the first element in Data to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

All values of the LabVIEW array are transferred (from the start element). The `Data` array must be greater than the number of transferred values.

The `Single` values of the source array are automatically transferred to the data type of the destination array on the *ADwin* system during transfer.

GetData_Float.VI

`Get_Data_Float.VI` transfers values from a `Data` array on an *ADwin* system into a LabVIEW vector of data type `Single`.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1`…`Data_200`.
Startindex	Index of the first element in the selected array to be transferred.
Count	Number of values to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
Data	Array with the read values of data type `Single`.
error out	Error signal

Notes

The values of the source array are automatically converted to data type `Single` during transfer.

SetData_Double.VI

`SetData_Double.VI` transfers `Double` values from a LabVIEW array into a `Data` array on the *ADwin* system.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1`...`Data_200`. `Data` may have data type `Long`, `Float`, `Float32`, or `Float64`.
Data	Array with `Double` values to be transferred.
Startindex	Index of the first element in Data to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

All values of the LabVIEW array are transferred (from the start element). The `Data` array must be greater than the number of transferred values.

The `Double` values of the source array are automatically transferred to the data type of the destination array on the *ADwin* system during transfer.

GetData_Double.VI

`Get_Data_Double.VI` transfers values from a `Data` array on an *ADwin* system into a LabVIEW vector of data type `Double`.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1`...`Data_200`.
Startindex	Index of the first element in the selected array to be transferred.
Count	Number of values to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
Data	Array with the read values of data type `Double`.
error out	Error signal

Notes

The values of the source array are automatically converted to data type `Double` during transfer.

Data2File.VI

`Data2File` saves data of data type `Long`, `Float`/`Float32`, or `Float64` from a `Data` array of the *ADwin* system into a file (on the hard disk).

Inputs

Device No.	Device No. of the system.
Mode	Writing mode: 0 : file will be created or overwritten (if file exists) 1 : data are appended to an already existing file
Filename	Path and file name.
DataNo	Number (1…200) of the array `Data_1`…`Data_200`.
Startindex	Index of the first element in the selected array to be transferred.
Count	Number of values to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	0: OK ≠0: Error
error out	Error signal

Notes

The used `Data` array must not be defined `as Fifo` to use `Data2File`.

The data are saved as binary file in the appropriate data type (see table). If not existing, the file will be created.

Data type of `DATA` array	Saved data type
Long	`Long`
Float (until processor T11) Float32 (processor T12/T12.1)	`Float` (32 bit)
Float64 (Prozessor T12/T12.1)	`Float64`

File2Data.VI

`File2Data` copies data from a file (on the hard disk) into a `Data` array of the *ADwin* system.

The VI doesn't work with Linux or MAC OS.

Inputs

Device No.	Device No. of the system.
Filename	Path and file name of the source file.
DataType	Data type of the values saved in the file. Select one of the following contants: `TYPE_LONG`: Values of type integer (32 bit). `TYPE_FLOAT`: Values of type float 32 bit. `TYPE_FLOAT64`: Values of type float 64 bit.
DataNo	Number (1…200) of the destination array `Data_1` … `Data_200`.
Startindex	Index (≥1) of the first element in the destination array to be written.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	0: OK ≠0: Error
error out	Error signal

Notes

The `Data` array must not be defined as `Fifo`.

The array must be dimensioned great enough to hold all values of the file.

The file values are expected to be saved as binary in one of the formats `Long`, `Float`/`Float32`, or `Float64`.

If the destination array has a different data type than `DataType`, the values of the source file are converted into the destination data type. There are the destination data types `Long` and `Float`.

5.4.2 Fifo Arrays

Instructions for data transfer between the PC and the *ADwin* system with global `Data` arrays (`Data_1`...`Data_200`), which are declared as Fifo ring buffer (First in first out).

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

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

– `LONG`	-2147483648 ... +2147483647
– `FLOAT` (until T11), `FLOAT32`	negative: $-3{,}402823 \cdot 10^{+38}$ ... $-1{,}175494 \cdot 10^{-38}$ positive: $+1{,}175494 \cdot 10^{-38}$ ... $+3{,}402823 \cdot 10^{+38}$
– `FLOAT64`	negative: $-1{,}7976931348623 15 \cdot 10^{+308}$ ... $-2{,}2250738585072014 \cdot 10^{-308}$ positive: $+2{,}2250738585072014 \cdot 10^{-308}$ ... $+1{,}7976931348623 15 \cdot 10^{+308}$

When working with Fifo arrays you should as a rule of thumb not write more elements into the Fifo as you have declared. You should definitely not read out more elements than you have written into the Fifo. Avoid this as follows:

– Check with the function `Fifo_Full`, if a Fifo array contains at least as many elements as you wish to read out with `Get_Fifo`.
– Check with the function `Fifo_Empty`, if a Fifo array contains at least as many free elements as you wish to write into the Fifo with `Set_Fifo`.

Fifo_Empty.VI

`Fifo_Empty.VI` returns the number of free elements of a Fifo array.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1`...`Data_200`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Count	≠255: Number of free elements in the Fifo array. 255: Error
error out	Error signal

Notes

Please note the details about error handling in chapter 4.3.

Fifo_Full.VI

`Fifo_Full.VI` returns the number of used elements of a Fifo array.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1...Data_200`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
Count	≠255: Number of used elements in the Fifo array. 255: Error
error out	Error signal

Notes

Please note the details about error handling in chapter 4.3.

Fifo_Clear.VI

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

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1...Data_200`.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

When an *ADwin* process starts, the pointers of a FIFO array are not initialized automatically. In *ADbasic*, we recommend to call `Fifo_Clear` at the beginning of your program.

Please note for an *ADsim* process: An initialization cannot be performed inside of an *ADsim* process, therefore an initialization with `Fifo_Clear.VI` can be useful.

Initializing the FIFO pointers at run-time is useful, if you want to clear all data of the array, e.g. because of a measurement error.

SetFifo_Long.VI

`SetFifo_Long.VI` transfers `Long` values from a LabVIEW array into a Fifo array of the *ADwin* system.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1`...`Data_200`.
Data	Array with `Long` values to be transferred.
Count	Number of array elements to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

Check with the function `Fifo_Empty.VI`, if a Fifo array contains at least as many free elements as you wish to write into the Fifo with `Set_Fifo.VI`.

The `Long` values of the source array are automatically converted to the data type of the destination array on the *ADwin* system during transfer.

GetFifo_Long.VI

`GetFifo_Long.VI` transfers Fifo data on an *ADwin* system into a LabVIEW vector of data type `Long`.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1`...`Data_200`.
Count	Number of array elements to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
Data	Array with the read values of data type `Long`.
error out	Error signal

Notes

Check with the function `Fifo_Full.VI`, if a Fifo array contains at least as many elements as you wish to read out with `Get_Fifo.VI`.

The values of the source array are automatically converted to data type `Long` during transfer.

SetFifo_Float.VI

`SetFifo_Float.VI` transfers `Single` values from a LabVIEW array into a Fifo array of the *ADwin* system.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1`...`Data_200`.
Data	Array with `Single` values to be transferred.
Count	Number of array elements to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

Check with the function `Fifo_Empty.VI`, if a Fifo array contains at least as many free elements as you wish to write into the Fifo with `Set_Fifo.VI`.

The `Single` values of the source array are automatically converted to the data type of the destination array on the *ADwin* system during transfer.

GetFifo_Float.VI

`GetFifo_Float.VI` transfers Fifo data on an *ADwin* system into a LabVIEW vector of data type `Single`.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array `Data_1`...`Data_200`.
Count	Number of array elements to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
Data	Array with the read values of data type `Single`.
error out	Error signal

Notes

Check with the function `Fifo_Full.VI`, if a Fifo array contains at least as many elements as you wish to read out with `Get_Fifo.VI`.

The values of the source array are automatically converted to data type `Single` during transfer.

SetFifo_Double.VI

SetFifo_Double.VI transfers Double values from a LabVIEW array into a Fifo array of the *ADwin* system.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array Data_1...Data_200.
Data	Array with Double values to be transferred.
Count	Number of array elements to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
error out	Error signal

Notes

Check with the function Fifo_Empty.VI, if a Fifo array contains at least as many free elements as you wish to write into the Fifo with Set_Fifo.VI.

The Double values of the source array are automatically converted to the data type of the destination array on the *ADwin* system during transfer.

GetFifo_Double.VI

GetFifo_Double.VI transfers Fifo data on an *ADwin* system into a LabVIEW vector of data type Double.

Inputs

Device No.	Device No. of the system.
FifoNo	Number (1…200) of the Fifo array Data_1...Data_200.
Count	Number of array elements to be transferred.
error in	Connector for error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠255: OK 255: Error
Data	Array with the read values of data type Double.
error out	Error signal

Notes

Check with the function Fifo_Full.VI, if a Fifo array contains at least as many elements as you wish to read out with Get_Fifo.VI.

The values of the source array are automatically converted to data type Double during transfer.

5.4.3 Data arrays with strings

Instructions for string transfer between the PC and the *ADwin* system with global `Data` arrays (`Data_1...Data_200`).

Before using it you have to declare each array under *ADbasic* (see manual *ADbasic*): `DIM Data_x[n] AS STRING`

SetData_String.VI

`SetData_String.VI` transfers a string from LabVIEW into a `Data` array in the *ADwin* system.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1...Data_200`.
Data	Array with the string to be transferred.
error in	Error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠-1: Number of transferred characters -1: Error
error out	Error signal

GetData_String.VI

`GetData_String.VI` transfers a string from a `Data` field in the *ADwin* system to LabVIEW.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array `Data_1...Data_200`.
MaxCount	Max. number of characters to be transferred.
error in	Error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠-1: Number of transferred characters -1: Error
Data	Array with the read values.
error out	Error signal

Notes

A string in a `Data` array ends with a delimiter (ASCII value 0). The transfer ends exactly at this position. The return value is the number of transferred characters without the end delimiter.

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

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 the String_Length.VI.
- You read the string with Getdata_String.VI and pass the actual number of chars as MaxCount.

String_Length.VI

String_Length.VI returns the length of a string in a Data-Feld on the *ADwin* system.

Inputs

Device No.	Device No. of the system.
DataNo	Number (1…200) of the array Data_1…Data_200.
error in	Error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
return value	≠-1: Length of string in the Data array. -1: Error
error out	Error signal

Notes

String_Length.VI counts the characters in a Data array until an end delimiter is found (ASCII value 0). The end delimiter is not counted as a character.

5.5 Control and error handling

Error_msg.VI

`Error_msg.VI` shows the error message corresponding to the last error, which occurred with an *ADwin* system.

Inputs

show error dialog?	true: show error message false: no function
error in	Error signal.

Outputs

error ?	true: error signal is present false: no error
error out	Error signal

Notes

The output "error ?" is applicable to stop the program in case of an error e.g. in a loop.

You find a list of all error messages in section A.2 in the annex.

Set_Language.VI

`Set_Language.VI` sets the language for the error messages.

Inputs

Device No.	Device No. of the system.
Language	Used language: 0 : language is set by Windows 1 : English 2 : German
error in	Error signal.

Outputs

Device No. out	The number connected to the input „Device No.“.
error out	Error signal

Notes

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

5.6 Tools

Volt2Value.VI

`Volt2Value.VI` converts a voltage value into the appropriate digit value for a DAC.

Inputs

Input voltage	Single voltage value in Volt.
Input voltage array	Array with voltage values in Volt.
Input voltage array 2dim.	2-dimensional array with voltage values in Volt.
Converter resolution	Resolution of the DAC.
Voltage range	Voltage range of the DAC.

Outputs

Output value	Digit value to the input "Input voltage".
Output value array	Digit value to the input "Input voltage array".
Output value array 2dim.	Digit value to the input "Input voltage array 2dim.".

Notes

The VI calculates a digit value starting from a voltage value. If a DAC is set to the calculated digit value, the DAC set the voltage on the analog ouput. Set the inputs "Converter resolution" and "Voltage range" according to the used DAC.

The inputs "Input voltage" may be used as single oder in parallel.

Value2Volt.VI

`Value2Volt.VI` converts a digit value of an ADC into the appropriate voltage value.

Inputs

Input value	Single digit value.
Input value array	Array with digit values.
Input value array 2dim.	2-dimensional array with digit values.
Converter resolution	Resolution of the ADC.
Voltage range	Voltage range of the ADC.

Outputs

Output voltage	Voltage value to the input "Input value".
Output voltage array	Voltage value to the input "Input value array".
Output voltage array 2dim.	Voltage value to the input "Input value array 2dim.".

Notes

If an ADC returns a digit value, the `Value2Volt.VI` calculates the voltage value, which is set at the ADC input. Set the inputs "Converter resolution" and "Voltage range" according to the used ADC.

The inputs "Input value" may be used as single oder in parallel.

Annex

A.1 Index of Functions

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.
103	The Data array is not declared.
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.
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.