

Version: 1.0.1  
Modified: November 18  
Products: ASC500 Remote Control

# User Manual

## ASC500

### Remote Control with LabVIEW

attocube systems AG, Eglfinger Weg 2, D – 85540 Haar Germany  
Phone: +49 89-420 797 – 0  
Fax: +49 89-420 797 201 90  
E-Mail: [info@attocube.com](mailto:info@attocube.com)

**For technical queries, contact:**  
[support@attocube.com](mailto:support@attocube.com)

attocube systems office in Haar:  
Phone +49 89 420 797 0  
Fax +49 89 420 797 201 90



## Table of Contents

Table of Contents .....	3
I. Introduction .....	4
I.1. Overview .....	4
I.2. Safety Information .....	4
II. Setup Procedure .....	5
II.1. Parts of the Package .....	5
II.2. Placing the LabVIEW VIs and Controls .....	5
II.3. The LabVIEW project and the example.vi .....	5
III. Virtual Instruments (VIs) .....	7
III.1. Overview .....	7
III.2. General Use of a Basic VI .....	7
III.3. Differences between Version 2.6 and 2.7 .....	8
III.4. Detailed Description of Basic VIs .....	10
III.4.a. General .....	11
III.4.b. Data Acquisition .....	15
III.4.c. Scanner .....	19
III.4.d. Assorted .....	21
III.4.e. Spectroscopy .....	22
III.4.f. Frequency and Amplitude Control .....	23
III.4.g. Z Control .....	24
III.4.h. LockIn .....	26
III.4.i. DAC .....	28
III.4.j. Coarse .....	29
III.4.k. Path Mode .....	29
III.4.l. Generic Feedback .....	32
III.4.m. Actor Scaling .....	32
III.5. Example VI .....	34
IV. Individual programs .....	37
IV.1. Pattern of a simple program .....	37
IV.2. Coexistence of Daisy and a LabVIEW program .....	37
IV.1. Use of the Data Channels .....	39
IV.2. Data handling in ASC500_Wrapper.dll .....	40
V. Problem solving .....	42
V.1. Common errors .....	42

## I. Introduction

### I.1. Overview

The package described in this document provides an interface between the ASC500 and LabVIEW. This enables the user to remotely control and program the ASC500.

With a large number of Virtual Instruments (VIs) it is possible to use all functionality the ASC500 offers in individual LabVIEW programs, making it possible to use those function in exactly the desired way. Additionally, it will be very easy to connect to other systems of the laboratory.

The manual will describe the steps necessary to write own LabVIEW programs to control the ASC500. It gives an overview of the included VIs, their function and usage. A section will also cover common problems and errors.

### I.2. Safety Information

For the continuing safety of the operators of this equipment, and the protection of the equipment itself, the operator should take note of the **Warnings, Cautions, and Notes** throughout this handbook and, where visible, on the product itself.

The following safety symbols may be used throughout the handbook:



**Warning.** An instruction which draws attention to the risk of injury or death.



**Caution.** An instruction which draws attention to the risks of damage to the product, process or surroundings.



**Note.** Clarification of an instruction or additional information.

## II. Setup Procedure

### II.1. Parts of the Package

There are two main parts which come with the package. On the one hand there are the libraries called 'ASC500\_Wrapper.dll' and "daisybase.dll" located in the folder 'Files\_ASC500', which encapsulate the functionality of the ASC500 in a way LabVIEW can cope with it. On the other hand there is the LabVIEW project that includes type definitions and lots of individual LabVIEW VIs representing functions of the controller and making them accessible.

### II.2. Placing the LabVIEW VIs and Controls

If you want to be able to access the VIs and controls directly from the functions palette, you have to copy them into the folder "\_express". It is located in the folder "users.lib" in the installation directory of LabVIEW. If you sort the VIs in different folders, it will be reflected in the function palette, too.

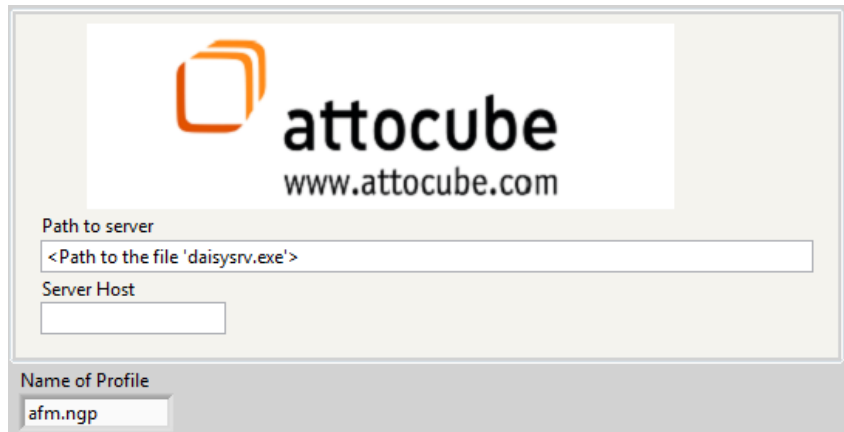
### II.3. The LabVIEW project and the example.vi

The VI "example.vi" might help you to find the right structure for your program. It can be found in ..\VIs\Example\example.vi in the project folder. This VI contains all Basic VIs in a way they can be used. The user interface of example.vi has been created quite similar to the user interface of the Daisy software. This enables you to find the required VIs in the example.vi quite easily, if you know where the functionality can be controlled in the user interface of the Daisy software. For more information about the example.vi, refer to section "Example VI".

If you want to create your own LabVIEW VIs, it is advisable to add them to the ASC500\_Wrapper-LabVIEW-Project. By doing that, LabVIEW automatically knows where to find all required data files such as subVIs and TypeDefs. The easiest way to add your created LabVIEW VI to the project is to save it in one of the project folders. The LabVIEW project will add your VI automatically if it finds it inside these folders. You can also create an additional folder in the project directories to have your VIs separated from the basic VIs. However, you'd have to add the created folder to the project manually, which is done by right clicking at "My Computer" in the LabVIEW project window and select "Add" and "Folder (Auto-populating) ...". For detailed information how to create your own VIs refer to the section "Pattern of a simple program".

As a first test you can open example.vi. Before it works properly there might be some things to configure. It is possible that LabVIEW asks you to enter the storage path of the DLL "ASC500\_Wrapper". It can be found in the folder "Files\_ASC500". Note that it is best to enter the storage path after opening example.vi, as the path will be saved for all subVIs either. If you have any problems regarding the storage path of the DLL, please refer to the problem solving section. As soon as LabVIEW found the DLLs, the front panel will be

displayed. Before example.vi can be executed successfully, you must fill out the init-Cluster and a separated variable as you can see it in the picture below. The variable 'Path to server' has to be filled with the storage path of the file 'daisysrv.exe'. Keep in mind that the backslashes have to be escaped (double backslash instead of a single one). 'Server Host' only has to be entered, if you want to run Daisy on a different computer as the server (See detailed description of **init.vi** in section "Detailed description of Basic VIs"). The separated variable has to be filled with the name of the standard profile file for the ASC500. The correct name is, as displayed in the picture, "afm.ngp". Now example.vi can be executed. If there are any problems, please refer to the problem solving section.



The screenshot shows a configuration window for attocube. At the top, there is the attocube logo and the website address www.attocube.com. Below this, there are three input fields:

- Path to server**: The input field contains the text "<Path to the file 'daisysrv.exe'>".
- Server Host**: The input field is empty.
- Name of Profile**: The input field contains the text "afm.ngp".

## III. Virtual Instruments (VIs)

### III.1. Overview

All VIs are organized in a LabVIEW project. The project contains different folders with different files that are necessary for the remote control of the ASC500.

The folder 'Controls' contains some type definitions that are used by some of the VIs. It is not necessary to access them directly.

The folder 'Files\_ASC500' contains the DLLs and some other files. It is strongly recommended not to change the data inside this folder.

The folder "VIs" is subdivided into two folders. One of these is called "Basic VIs", which contains VIs helping you to configure or read out general settings. Inside the folder 'Example' you can find the 'example.vi' that has been created to show you one way the VIs can be used.

### III.2. General Use of a Basic VI

This section describes some basic things one has to consider when making a LabVIEW program with ASC500 VIs.

First of all, every basic VI has an error in- and output. These wirings allow the user to determine where and why an error occurred. It comprises a string explaining in which VI the error occurred and an error code indicating the cause of the error. Furthermore, a VI, which is called with an incoming error, will not execute the routine it would normally do. It will just pass the error message to the next VI of the program. For example, if an error occurred within the 'Init.vi' and the connection to the server was not established correctly there is no use in working with other VIs and the user might want the program to end.

All other wirings a VI might possess are parameters or data going to the ASC500 or coming from the controller. If there is a large number of parameters handled by a VI, a cluster is used to pass or return the parameters. This means all inputs were bundled into a cluster and this cluster is then passed to the VI. This is done analogically for values returned by the VI. It is advisable to create those clusters by right clicking at the VI's in- or output terminal and select 'create' and 'control' or 'indicator'. This will automatically create the correct cluster.

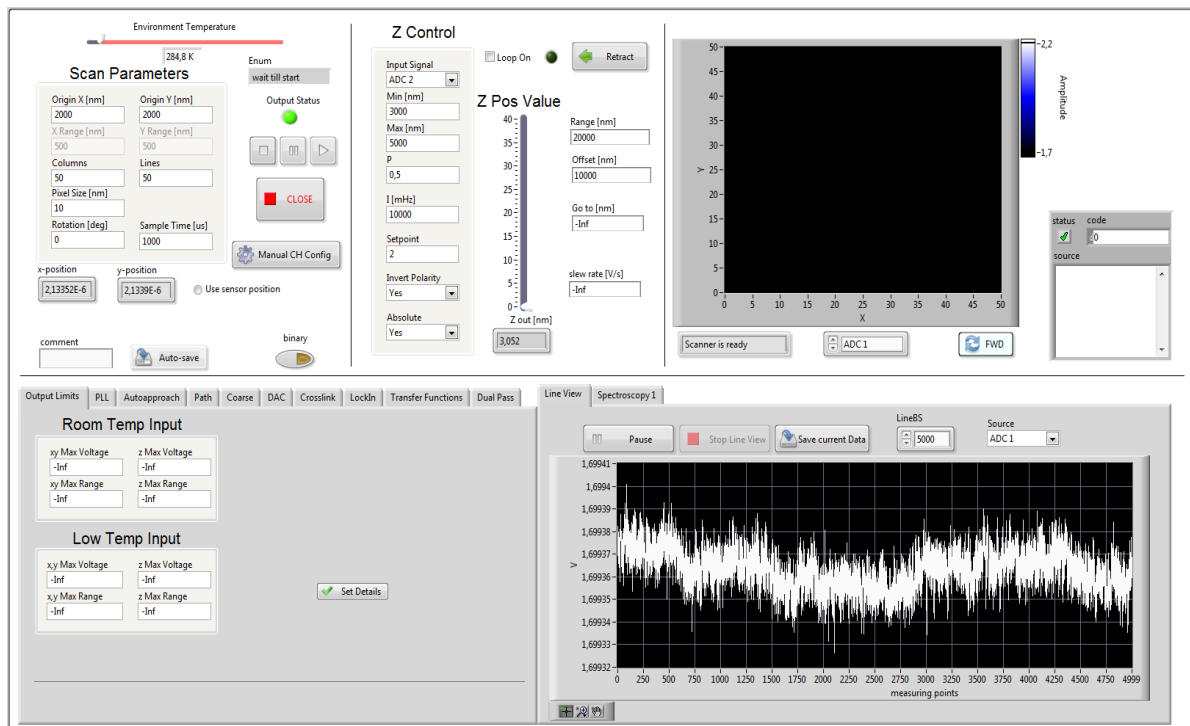
A technique that needs clarification is surely the usage of '-Inf' and '-2147483648' as default input values for almost all parameters concerning a default cluster. This is used to tell the VI (and a bit deeper the underlying DLL) not to write this parameter to the ASC500, but to read out the actual value. The result is then shown in the output cluster. '-Inf' is used for floating point variables and '-2147483648' for integer values.

But not only if a special value is wired to a VI the output cluster or output

indicators are used. Through them the VI will return the value that has been set within the ASC500. This opens the possibility to check whether the value that should be written, has really been set. This is not always true, as there is a bit-resolution for every parameter. As a consequence, the ASC500 might be forced to set the parameter to the nearest possible value. Another case that might occur is that the parameter which should be changed is somehow correlated with another input value and the ASC500 automatically corrects the value to a sensible one.

For many VIs there is a context help text, which gives the user a short overview about the usage and the in- and output parameters of the VI. By pressing 'CTR + h' in LabVIEW a window will appear showing the context help.

### III.3. Differences between Version 2.6 and 2.7



There are two different currently supported versions of ASC500 Daisy. Depending on the hardware version of the ASC500 one or the other has to be used<sup>1</sup>.

Many VIs of version 2.7 are identical to their version 2.6 counterparts. However, there are some things to consider when working with version 2.7 (The picture above shows the front panel of the example.vi of version 2.7).

<sup>1</sup> For detailed information on this topic, please check the ASC500 manual, section 2.3 Front and Rear Panel connections.

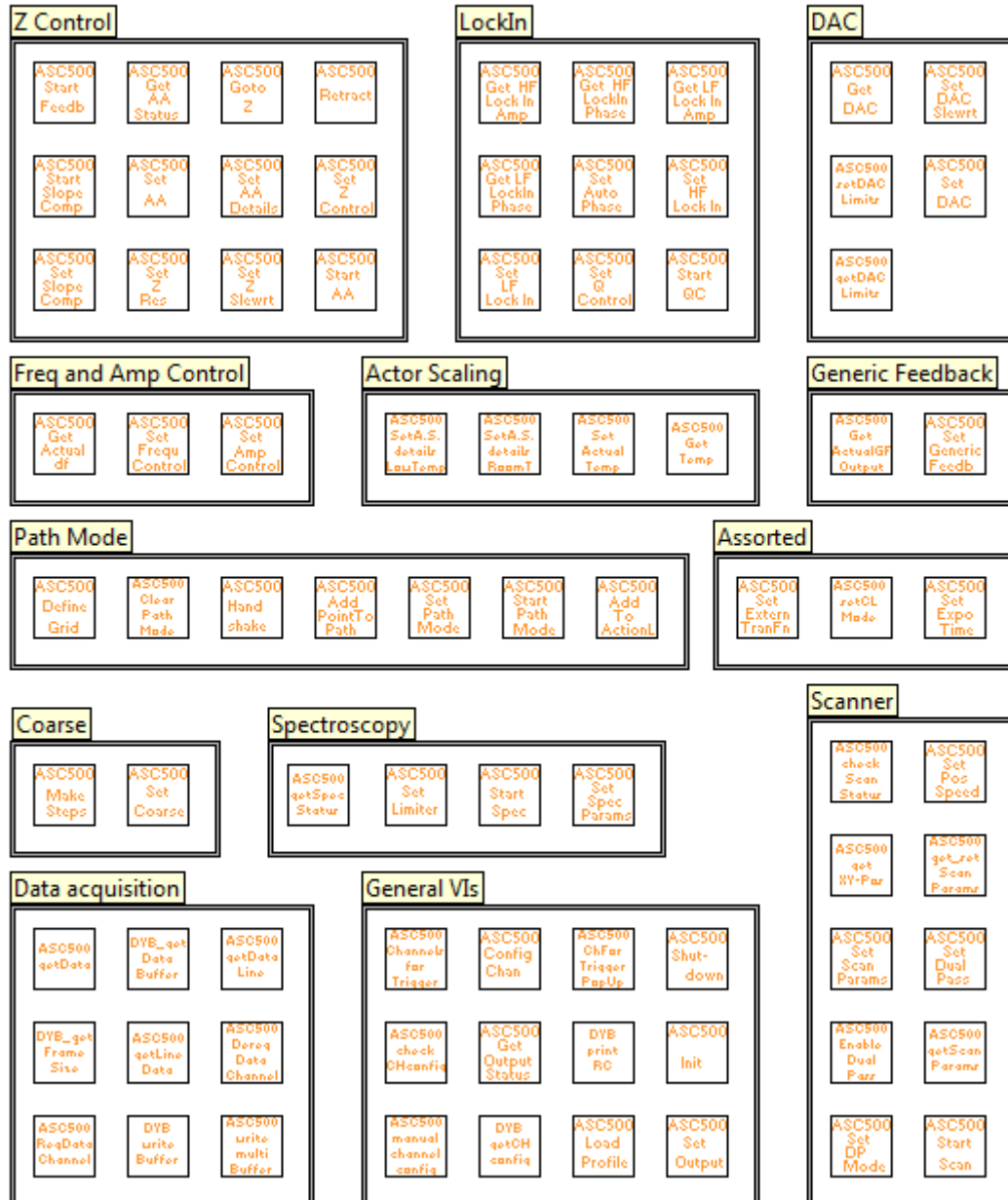


In V27 the scan engine was completely changed to enable Closed Loop Scanning.

With this new scan engine, it is e.g. necessary to click the “stat scanner”-button in the scan VI twice before the scanner starts. After the first start-command the scanner will move to the start position of the scanner field. When clicking the start button the second time, after the approach to the starting point has been finished, a scan will be started. Furthermore, it is possible to use the Closed Loop feature for the scanner position which is done by activating “Use sensor position”.

The temperature handling has been improved, too. In V27 it is possible to choose a specific temperature between room and low temperature. The ASC500 will interpolate the limits based on the chosen temperature. For this purpose the VI **setActualTemperature.vi** has been created.

### III.4. Detailed Description of Basic VIs



Similar to the organization in Daisy, there are several groups of VIs that will work on specific topics.

### III.4.a. General

#### Init.vi



This VI establishes a connection to the server and starts a new server if none is running at that moment. The init cluster contains:

- server path (string): Path to the file "daisysrv.exe". It can usually be found in the installation directory of Daisy.
- server host (string): Hostname or IP address in "dotted decimal" notation for the host where the application server resides. Empty if the server should run locally.

If there is no server running already, the server path specifies the location of the server executable. Server host should only be declared if you want to access a running server on a different computer.

#### LoadProfile.vi



Loads the settings and parameters of an existing profile, which is found at 'path' (string-input) into the controller. The profiles are saved as .ngp-files and can be found in the Daisy installation directory.

#### ConfigChannel.vi



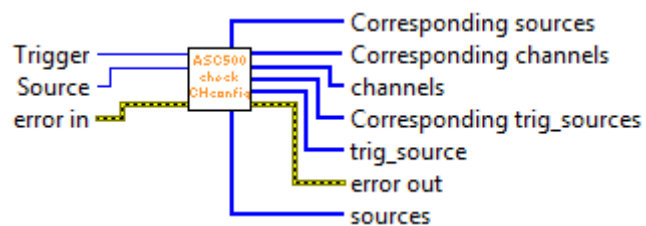
Prepares a channel for data reception. The channel cluster contains:

- channel (integer) [0 to 13] **required input**
- trigger (integer) [0 -- disabled; 1 -- scanner; 2 -- permanent; 3,4,5,6 -- spectroscopy 0,1,2,3; 9 -- command]
- source (integer) [0-5 -- ADC 1-6; 7 -- aexc; 8 -- fexc; 9 -- zOut; 12 -- signal; 13 -- ampl; 14 -- phase; 15 -- mampl; 17 -- mphase; 18 - zoutinv; 20 -- ADCext\_min; 21 -- ADCext\_max]
- average (integer) [0 to 1]

- sample time (float)

The 'channel' is used to identify the data channel later. 'Trigger' specifies the expected data format coming from the 'source'. If the trigger was set to *permanent*, a 'sample time' may be stated over which is averaged if the 'average' flag has been set. To read out the current configuration of a data channel, please use **getChannelConfig.vi**.

#### CheckCHConfig.vi



This VI provides information about the configuration of the fourteen data channel regarding one of the input variables. Apart from the error in cluster it has two input variables. Trigger defines the event that causes data acquisition on a specific channel. "Source" defines where the data comes from. It works as follows. If you connect a specific trigger to the same named input, the VI will output arrays with different content:

- "Corresponding channels" contains the numbers of channel (0-13) that are triggered by the given input.
- "channels" is an array filled with 14 numbers that are either -1 or 1. The index of the array element equals the channel number. If the element of index 4 equals 1, it means that channel 4 is triggered by the same trigger specified by the input-variable. If the array element of index 4 equals -1 it means, that channel 4 is triggered by a different as the requested trigger.
- "Corresponding trig\_sources" contains the sources that are set to the requested trigger in any channel. The numbers in the array represent the sources as defined in the typedef "channel-source\_selection".
- "trig\_sources" is an array of 14 elements which contains either a -1 or a number representing the source of the channel as defined in the typedef "channel-source\_selection". The index of the array where the number is located tells you which channel is set to the configuration.

If the input "source" is connected, the VI provides information about the channel configuration in the following arrays:

- The array "Corresponding sources" contains the numbers of

channels that are set to retrieve data from the “source” given by the input variable.

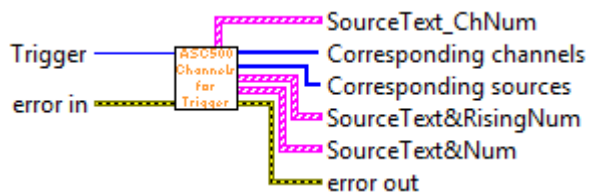
- “sources” is an array of 14 elements that works similar to “channels” with the difference that it checks the sources instead of triggers.

#### manualChannelConfiguration.vi



A pop up window will appear when this VI is called. It shows the current channel configuration. If you want to change the configuration, enter the desired values into the channel-parameters-cluster and press “apply changes”. The changes can be checked in the table beneath. Furthermore you can define a storage path for each channel. A string array with fourteen elements containing the storage paths will be provided by the VI. The storage path of channel 1 will be stored in the array at index 1.

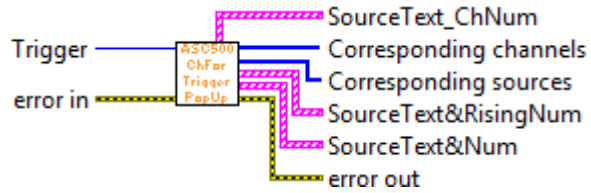
#### channelsforTrigger.vi



This VI gives you information about the channels. If you pass a trigger as input to the VI, you will get the following outputs:

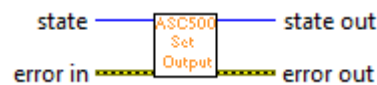
- sourceText\_ChNum: This array of clusters contains the channel numbers which are triggered by the input-trigger as the second element of the cluster. The first element is the name of the source that is set for the channel.
- Corresponding channels: This array of integer contains all channel numbers that are triggered by the input-trigger.
- Corresponding sources: This array contains all source-numbers of the channels that are triggered by the input-trigger.
- SourceText&RisingNum: Array of clusters of two elements. It contains all input-trigger-related sources. The second element is a number that equals the array index.
- SourceText&Num: This array of clusters of two elements contains the input-trigger-related sources. The second element is the corresponding number of the source as it is defined in the typedef “channel-source\_selection”.

### ChannelsForTriggerPopUp.vi



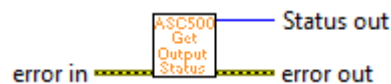
Same functionality as the **channelsForTrigger.vi** with the difference that there will be a pop-up window if there is no channel configured with the trigger defined by the input variable. Furthermore an error will be reported in that case.

### SetOutput.vi



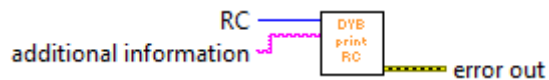
Activates (state = 1) or deactivates (state = 0) the output. Activation of the output is necessary prior to many operations such as scanning, zControl or performing a spectroscopy.

### GetOutputStatus.vi



Returns the output status of all outputs (1=activated; 0=deactivated)

### printRC.vi



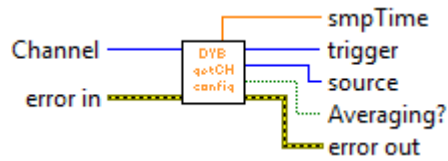
Depending on the input "RC" (**R**eturn **C**ode) this VI fills up the error out-Cluster with additional information about the emerged error. You can add additional information by typing it into the corresponding input (e.g. the function name where the error occurred). The string will be added to the error description of the error-Cluster in brackets.

### ShutdownServerCon.vi



This VI closes the connection to the server and therefore presents the last VI concerning the ASC500 in an application.

### getChannelConfig.vi



Reads out the channel configuration. The parameters are

- Channel: Number of the channel of interest (0 to 13)
- Trigger: Trigger source for data output
- Source: Data source for the channel
- Averaging?: If data is averaged over the sample time
- smpTime: Time per sample in seconds

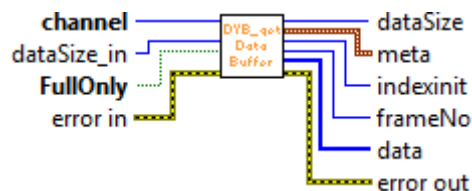
## III.4.b. Data Acquisition

### RegDataChannel.vi



Registers a data channel, which enables the reception of data over the specified 'channel'. The buffer size must be at least one frame size (see **getFrameSize.vi**) and greater than 129.

### getDataBuffer.vi

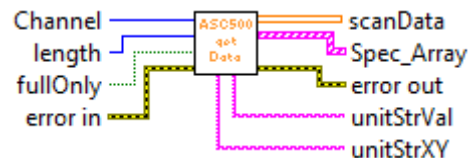


Retrieves data from a channel specified by the input with the same name. The data array contains raw data that has not been processed yet. We recommend to use **getData.vi** or **getDataLine.vi** instead of this one to access data from the ASC500. The additional inputs are:

- dataSize\_in: Size of the data array to be read out. The required information is how many 32 bit integers will be in the data array.
- FullOnly: specifies whether only completely filled data arrays should be read out.
- The outputs are:

- dataSize: Size of the output data array
- meta: Cluster that contains information which data has been read out
- indexinit: index of the first element in the output array
- frameNo: number of frames that has been retrieved.

#### getData.vi



Processes the data that is retrieved over a registered channel and has the corresponding data arrays as outputs. The inputs are:

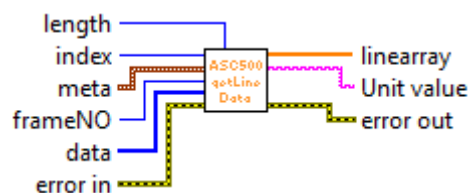
- Channel: The data-channel which has been registered with regDataChannel.vi
- Length: The length of one frame. This input is to be calculated by getFrameSize.vi
- fullOnly: If you want to have the data updated during execution of the process (e.g. a scan or spectroscopy), this value should be set to "False".

The outputs are:

- unitStrXY: String indicating the Unit of the independent value
- unitStrVal: String indicating the Unit of the dependent value
- scanData: This 3D-array contains the data of fwd- and bwd scans of multiple channels, if more than one channel is triggered by the scanner.
- Spec\_Array: The Spec\_Array is actually a cluster that contains two 1 D arrays of numbers with datatype single. The "first" array contains the independent values and the "second" array contains the dependent values.

This VI cannot be used to retrieve data from the LineVIEW! See **getDataLine.vi** for doing that.

#### getLineData.vi



Processes data from the LineView passed by the input "data". This VI is part of the VI **getDataLine.vi**. We recommend to use **getDataLine.vi** to access data from the lineVIEW instead of this one. The inputs are

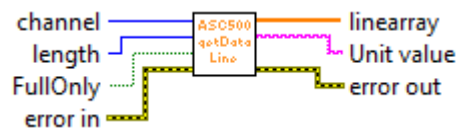


- Length: number of the datapoints in the input "array"
- Index: index of the newest data point
- Meta: meta data to process the raw data
- frameNO: Number of frames that have already been processed
- data: array with raw data points

The outputs are:

- Linearray: array that contains the processed LineView-data. The newest data point has always the highest index in this array
- Unit value (string): Unit of depended variable

#### getDataLine.vi



Like the getData.vi this VI processes data. But it has been created only for the lineView. The in- and outputs are quite similar to getData.vi. 'linearray' contains the retrieved data.

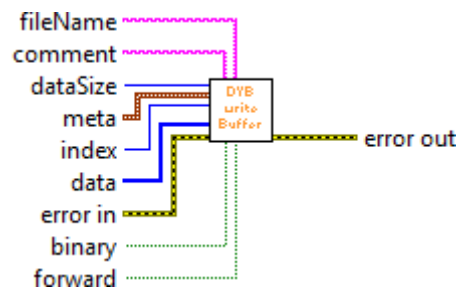
The number of data points in the linearray can be modified by the user by changing the length-input.

#### DeregDataChannel.vi



By calling this vi the data reception into the buffer for the specified 'channel' is stopped. This prevents the buffer from consuming more and more memory.

#### WriteBuffer.vi



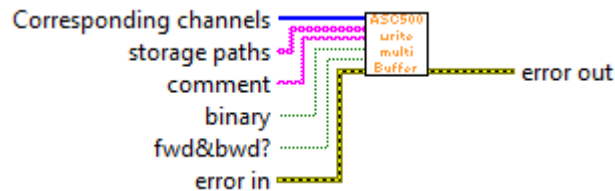
Writes the data from the "data"-array into a file. We recommend to use the VI 'writeMultiBuffer.vi' instead of this one. The inputs are:

- fileName: Name of the file to be created. You can also pass a

whole path to that input.

- Comment: String that will be written into the header of the generated file
- DataSize: Number of data points (32-bit elements) to be written to a file
- Meta: Meta data for the buffer as retrieved by getDataBuffer
- Index: index of first element in the buffer
- Data: data buffer as retrieved by getDataBuffer
- Binary: True if binary format is required. Only for scan-triggered data
- Forward: If only the data from the forward scan is to be written. Only relevant for scan-triggered data.

#### writeMultiBuffer.vi



Writes the data points of the channels into a file. The inputs are:

- Corresponding channels: This array contains channel numbers. The data, that can be retrieved from these channels, will be saved into the output-file.
- Storage paths: This input is of type array of strings. You can write the desired storage path for every channel into this array. Every line corresponds to a data channel. For example the third element of the array contains the storage path for the data file of channel 2. The zeroth element of the array is occupied and does not contain any storage path.
- comment: You can add a comment to the output file. It can be found in the header of the file.
- Binary: If this input is "false" the output file will be of type .asc. If it is "true" the output file will be of type .bcrf.
- Fwd&bwd?: Determines whether forward and backward ('true') data or only forward data ('false') should be written to file.

#### getFrameSize.vi



"FrameSize" is an integer, which tells you how many data points will be retrieved from a given data channel (input). This information is usually required by **getData.vi**. Please use this VI to calculate the frame size of a

channel instead of doing it manually.

### III.4.c. Scanner

#### SetScanParameters.vi

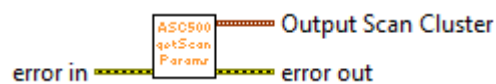


Sets the parameters for a scan. Make sure to check if the chosen arguments are consistent with each other. The individual parts of the 'Scan Parameters Cluster' are:

- Lines (integer) [0 to infinity]
- Columns (integer) [0 to infinity]
- Pixel Size [nm] (float) [0 to infinity]
- Origin X [nm](float) [0 to infinity]
- Origin Y [nm] (float) [0 to infinity]
- Rotation (float) [0 to infinity]
- Sample Time [ $\mu$ s] (float) [0 to infinity]

'Lines' and 'Columns' define the number of pixels the scan will have. The 'Pixel Size' states how large one single pixel will be. The 'Origin X' and 'Origin Y' parameters tell the controller where the field of scan shall be located. 'Rotation' fixes an angle by which the field is rotated. Every pixel will be measured with a certain time given by 'Sample Time'.

#### getScanParameters.vi



Retrieves the current values that are set in the scan cluster. For more information on the parameters that are read out by this VI, please refer to the description of **SetScanParameters.vi**.

#### setgetScanParameters.vi



You can set or/and retrieve the values of the scanParameters Cluster. If the input Set/Get equals "true" the values of the input cluster will be set. If it equals "false" the current values will be retrieved and written into the Output Cluster. The output "IN = OUT?" indicates whether the Scan Parameters Cluster equals the Output Cluster. If it is true, you can be sure, that the values you entered into the input-cluster, has been accepted and written to the server.

### StartScan.vi



Starts ('State' = 1) or stops ('State' = 0) the scanner.

### CheckScanStatus.vi



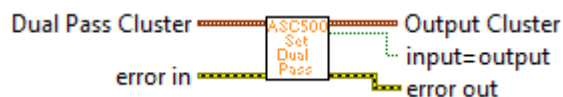
This VI returns the scanner status. Distinguished are "ready", "paused", "running", "adjusting" and "returning to origin".

### getXYPos.vi



Returns the current X- and Y-coordinates of the scanner in  $\mu\text{m}$ .

### SetDualPass.vi

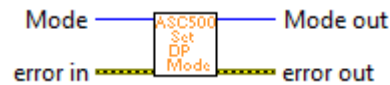


Sets the parameters for the second pass mode. The 'Dual Pass Cluster' contains the following values:

- Alternative Setpoint (integer) [0 -- no; 1 -- yes]
- Setpoint (float) [without restraint]
- Alternative DAC (integer) [0 -- no; 1 -- yes]
- Number of DAC (integer) [0-3 -- *DAC1-4*] required input
- Value of DAC in V(float) [-10 to 10]
- Alternative Fexc (integer) [0 -- no; 1 -- yes]
- Value of Fexc (float) [0 to 2000]
- Lift Offset (float) [0 to 1000000]
- Wait Time (float) [without restraint]
- Lift Slewrate (float) [0.001863 to 1000]

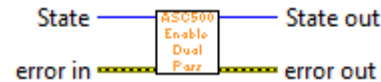
The 'Alternative Setpoint' flag activates or deactivates the use of an alternative setpoint. Its value would be given by 'setpoint'. The same applies for the fexc and an alternative DAC, with the constraint for the DAC that the number of it has to be specified as 'Number of DAC'. The output "input = output" indicates if in- and output-cluster contain the same values. If this is not the case at least one of the entered values has not been accepted by the server. When you pass -Inf (float) respectively -2147483648 (Integer) the variable will be read out.

#### SetDualPassMode.vi



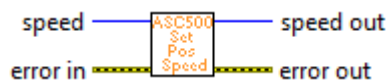
Two modes are available for the Dual Pass. The first is *feedback* (mode = 0) and the second one is *first line profile* (mode = 1). By passing -Inf (float) or -2147483648 (integer) as input values, the corresponding parameter(s) will be read out.

#### EnableDualPass.vi



Activates (state = 1) or deactivates (state = 0) the dual pass mode.

#### SetPositioningSpeed.vi



Sets the positioning speed of the scanner. The unit is nm/s. If you pass -2147483648 as input for "speed", the value will be read out and written into the output "speed out".

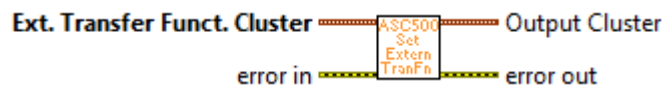
#### goToXY.vi



By the help of this VI the scanner can be moved to a specific xy-Position (input values). Note that this VI changes the scan range and the scan origin such that the scan starting point is the desired xy-Position. Please do not execute this VI twice with the same values for the input variables, as it can cause the scanner to start scanning.

### III.4.d. Assorted

#### SetExternalTransferFunction.vi



This VI sets up an external transfer function. The input cluster contains the following parameters.

- Number (integer) [0 to 2]
- Gain in Unit per volt (float) [without restraint]
- Offset in mV (float) [without restraint]
- Unit (integer) [0 -- mm; 1 -- um; 2 -- nm; 3 -- pm; 4 -- V; 5 -- mV; 6 -- uV; 7 -- nV; 8 -- MHz; 9 -- kHz; 10 -- Hz; 11 -- mHz; 12 -- s; 13 -- ms; 14 -- us; 15 -- ns; 16 -- A; 17 -- mA; 18 -- uA; 19 -- nA; 20 -- [cos]; 24 -- [dB]; 32 -- W; 33 -- mW; 34 -- uW; 35 -- nW]

By passing -Inf (float) or -2147483648 (integer) as input values, the

corresponding parameter(s) will be read out.

#### SetExposureTime.vi



This VI sets the exposure time in  $\mu\text{s}$ . If you pass  $-\text{Inf}$  the value will be read out.

#### setCLMode.vi



When „CL-Mode“ is set to 1, closed loop scanning is active. Note that the CL-Mode can only be changed, when the outputs are deactivated.

### III.4.e. Spectroscopy

#### SetSpecParameters.vi



This VI sets all parameters associated with spectroscopy mode. The input cluster contains the following:

- DAC (integer) [DAC 1-DAC 4; Z; Low Frequency]
- Mode (integer) [0-2 -- Spectroscopy1-3; 3 -- Calibration]
- Start (float) [depends on DAC input: mV, Hz or degree]
- End (float) [depends on DAC input: mV, Hz or degree]
- Points (integer) [0 to 65535]
- fwd (integer) [0 to 1]
- Repeat (integer) [0 to 65535]
- Delay in  $\mu\text{s}$  (float) [0 to 10000000]
- Average Time in  $\mu\text{s}$  (float) [0 to 10000000]

By passing  $-\text{Inf}$  (float) or  $-2147483648$  (integer) as input values, the corresponding parameter(s) will be read out.

#### SetLimiter.vi



This VI configures the limiter for any spectroscopy and activates it, if this is requested. The limiter cluster contains the following parameters:

- Mode (integer) [0-2 -- Spectroscopy1-3]
- Active (integer) [0 -- off; 1 -- on]
- Source (integer) [0-5 -- ADC1-6; 6 -- SPM Z out; 7 -- SPM Z out inv; 8 -- AFM Signal; 9 -- AFM df; 10 -- AFM Aexc; 11 -- AFM Aosc; 12 -- AFM Phase; 13 -- LockIn Ampl; 14 -- LockIn Phase]

-Sign (integer) [0 -- Source > Value; 1-- Source < Value]

By passing -Inf (float) or -2147483648 (integer) as input values, the corresponding parameter(s) will be read out.

#### SetLimiterPopUp.vi



This VI will show its front panel when it is called. You can now enter the desired variables into the input cluster. If you want to set the entered values, press the "set values" button below the clusters. The values will be set and the pop-up window will be closed afterwards. For a detailed description of the values, refer to **SetLimiter.vi**.

#### StartSpec.vi



Activates the output and starts a spectroscopy for state = 1 and deactivates the output for state = 0 as a spectroscopy will stop autonomously. 'Mode' defines, which spec engine should be started (index 0-2 corresponds to the engines 1-3).

#### getSpecStatus.vi

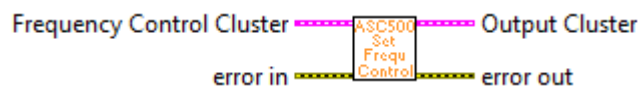


Returns the current status of the spectroscopy engine. Differentiated is specstatus = 0 if the engine does not run and specstatus = 1 if the engine runs.

With 'index'-input you can define from which spec-engine you want to know the status (index 0-2 corresponds to the engines 1-3).

### III.4.f. Frequency and Amplitude Control

#### SetFrequencyControl.vi



This VI sets up all parameters concerning the frequency control. The input cluster contains the following items:

- Actual Value (integer) [0-5 -- ADC1-6; 8 -- AFM df; 13 -- AFM df; 13 -- AFM Aosc; 14 -- AFM Phase; 16 -- LockIn Ampl; 17 -- LockIn Phase]
- Min in kHz (float) [0 to 500]
- Max in kHz (float) [0 to 500]
- P (float) [0 to 1]
- I in uHz (float) [0 to 1000000000]
- Setpoint; Unit depends on the input "Actual Value": mV, Hz or degree

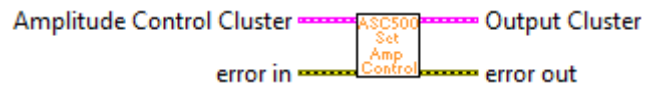
(float) [without restraint]  
 -Invert Polarity (integer) [0 -- no; 1 -- yes]  
 -df in kHz (float) [without restraint]  
 -f excite in kHz (float) [0 to 2000]  
 By passing -Inf (float) or -2147483648 (integer) as input values, the corresponding parameter(s) will be read out.

#### GetActualDf.vi



Returns the actual value of df in mHz.

#### SetAmpControl.vi



Sets all relevant parameters for the operation of an amplitude control:

- Amplitude in V (float) [0 to 10]
- Min in kHz (float) [0 to 10]
- Max in kHz (float) [0 to 10]
- P (float) [0 to 100]
- I in mHz (float) [0 to 1000000]
- Setpoint in V (float) [-10 to 10]
- Invert Polarity (integer) [0 -- no; 1 -- yes]

By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.

### III.4.g. Z Control

#### SetZControl.vi



This VI prepares the z control by setting up all necessary parameters. Those are bundle in 'Z Control Cluster':

- Actual Value (integer) [0-5 -- ADC1-6; 8 -- AFM df; 13 -- AFM Aosc; 14 -- AFM Phase; 16 -- LockIn Ampl; 17 -- LockIn Phase]
- Min in nm (float) [0 to infinity]
- Max in nm (float) [0 to infinity]
- P (float) [0 to 100]
- I in mHz (float) [without restraint]
- Setpoint (float) [without restraint]
- Invert Polarity (integer) [0 -- no; 1 -- yes]
- Absolute (integer) [0 -- off; 1 -- on]

By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.



### StartFeedback.vi



Starts (state = 1) or stops (state = 0) the z control.

### GotoZ.vi



Changes the z position to the given 'Value'. By passing -Inf (float) z position will be read out.

### Retract.vi



Retracts the tip.

### SetAutoapproach.vi



Configures the auto approach functionality. The cluster contains:

- Delay (float) [0 to 100000]
- Threshold in mV, deg or Hz depending on input signal from Z Control (float) [without restraint]
- Stop Condition (integer) [0 -- > Threshold; 1 -- < Threshold]
- Speed in V/s (float) [0.0005 to 1000]
- Steps per Approach (integer) [0 to 100]
- Approach Mode (integer) [0 -- ramp; 1 -- loop]
- Target Mode (integer) [0 -- loop on; 1 -- retract; 2 -- loop off]

By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.

### SetAutoapproachDetails.vi



Sets up some additional and more special parameters for the auto approach mode. The cluster contains:

- Axis (integer) [0 -- Axis 1; 1 -- Axis 2; 2 -- Axis 3] required input
- Direction (integer) [0 -- forward; 1 -- backward]
- Device (integer) [0 -- ANC150; 1 -- TTL via DAC2; 2 -- LVTTTL via DAC2]
- Trigger Pol. (integer) [0 -- high active; 1 -- low active]
- Hold Time in  $\mu$ s (integer) [0 to 300000]
- GND while Approach (integer) [0 -- no; 1 -- yes]

By passing -Inf (float) or -2147483648 (integer) the corresponding

value will be read out.

#### StartAA.vi



Starts (state = 1) or stops (state = 0) the auto approach.

#### GetAAStatus.vi



Returns the current state of auto approach (1 = on; 0 = off).

#### SetSlopeCompensation.vi



In both directions 'X' and 'Y' a value between -20% and +20% is accepted.

#### StartSlopeCompensation.vi



Activates (state = 1) or deactivates (state = 0) the slope compensation.

### III.4.h. LockIn

#### SetHFLockIn.vi



Sets all parameters relevant to the high frequency lock in. The input cluster contains:

- Amplitude in V (float) [0 to 10]
- Frequency in kHz (float) [0 to 2000]
- Sensitivity in V (float) [0 to 20]
- Phase Shift in deg (float) [0 to 360]
- Integration Time in us (float) [1 to 1000000]

By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.

#### SetLFLockIn.vi



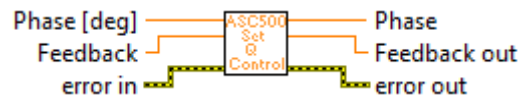
Sets all parameters relevant to the low frequency lock in. The input

cluster contains:

- Amplitude in V (float) [0 to 10]
- Frequency in kHz (float) [0 to 20]
- Sensitivity in V (float) [0 to 10]
- Phase Shift in deg (float) [0 to 360]
- Integration Time in us (float) [1 to 163000]
- Output Connector (integer) [0 -- off; 1 -- DAC1; 2 -- DAC2; 3 -- DAC1+DAC2]
- Input Connector (integer) [0-5 -- ADC1-6; 7 -- AFM Aexc; 8 -- AFM df; 9 -- SPM Z out; 12 -- AFM Signal; 13 -- AFM Aosc; 14 -- AFM Phase; 16 -- LockIn Ampl; 17 -- LockIn Phase; 18 -- SPM Z out inv]

By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.

#### SetQControl.vi



This VI sets the 'Phase' in degrees between -180° and +180° and the 'Feedback' between 0 and 1 for the Q Control. By passing -Inf as input values, the corresponding parameter(s) will be read out.

#### GetActHFLockInAmp.vi



Returns the actual value of the high frequency lock-in amplitude in V.

#### GetActLFLockInAmp.vi



Returns the actual value of the low frequency lock-in amplitude in V.

#### StartQC.vi



Starts (state = 1) or stops (state = 0) the Q Control.

#### GetActHFLockInPhase.vi



Returns the actual value of the high frequency lock-in phase in degrees.

#### GetActLFLockInPhase.vi



Returns the actual value of the low frequency lock-in phase in degrees.

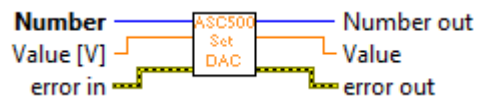
#### SetAutoPhase.vi



This VI comes with no input parameter, as it just performs an auto phase.

### III.4.i. DAC

#### SetDac.vi



The DAC addressed by 'number', which is a required input [0 to 3], is set to the 'value'. The value has to be given in V and is generally restricted from -10V to +10V. In addition, the value cannot exceed the set DAC limits which can be set with the **setDACLimits.vi**.

#### GetDac.vi



Returns the value in V at the DAC specified by 'Number of DAC', which is a required input [0 to 3].

#### SetDACSlewrate.vi



Changes the general slew rate for the DACs. It has to be given in V per second.

#### GetDACLimits.vi



Reads out the DAC-Limits for a DAC defined by the input "DAC".

The output Cluster contains the following parameters

- RT-Limit: set limit value for room temperature

- LT-Limit: set limit value for low temperature
- Current Limit: limit value as a result of interpolation regarding the current temperature.

#### setDACLimits.vi



Sets the DAC limits. These values restrict the values that can be set with the **setDAC.vi**.

### III.4.j. Coarse

#### SetCoarse.vi



Prepares an axis for coarse movement. The number of the axis is a required input and restricted to 0 to 2 corresponding to axis 1 to axis 3. The frequency should be given in hertz and may range from 1 to 8000Hz. Whereas the amplitude is given in V and is restricted to a range from 0 to 70V.

#### MakeSteps.vi



This VI initiates a 'number' of steps for the specified axis, which is of course a required input and runs from 0 to 2. For steps in bwd-direction enter the number of steps as a negative number.

### III.4.k. Path Mode

#### SetPathmode.vi



This VI sets all parameters necessary for the path mode and the external trigger. The cluster contains the following:

- Time [ $\mu$ s] (float) [0 to 65535]
- Edge (integer) [0 -- *Raising*; 1 -- *Falling*]
- Z Home [nm] (float) [without restraint]

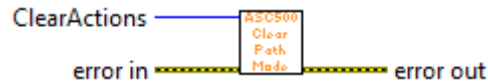
By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.

### AddPointToPath.vi



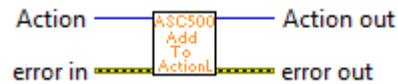
Adds a point specified by its coordinates (relative to the scanning start point) to the path mode list.

### ClearPathModeList.vi



Clears the list of points saved in the path mode. If the variable "ClearActions" is set to 1, the action list will be cleared, too.

### AddToActionList.vi



Adds an action to the list, which is performed at every point in the path list. The possible actions are:

- 0 -- *Handshake*
- 1 -- *Z Spectroscopy*
- 2 -- *DAC Spectroscopy*
- 3 -- *Low Frequency Spectroscopy*
- 4 -- *External Spectroscopy*
- 5 -- *Move Z to Home*
- 6 -- *Z Approach*

### checkHandshakeRequest.vi



Verifies whether a manual handshake is expected by the system (HsRequest = True). It can be sent by executing Handshake.vi.

### Handshake.vi



If the path mode is waiting for a manual handshake, this VI will give this handshake and the path will be continued.

### DefineGrid.vi



This VI creates a grid which is then processed point by point in path mode when started with **StartPathmode.vi**. The input parameters are:

- NumberOfPointsX (integer) [without restraint]

- NumberOfPointsY (integer) [without restraint]
- CoordX [ $\mu\text{m}$ ] (Array of floats) [without restraint]
- CoordY [ $\mu\text{m}$ ] (Array of floats) [without restraint]

The grid is defined by three points, which are passed by the arrays CoordX and CoordY. The first point should be passed at index 0 in the arrays CoordX and CoordY defining the origin point of the grid. The second point passed at index 1 defines the furthest away point from the origin point on the x-axis. The third point passed at index 2 defines the furthest away point on the y-axis.

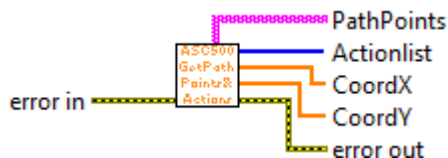
By passing  $-\text{Inf}$  (float) or -2147483648 (integer) the corresponding value will be read out.

#### StartPathmode.vi



Starts (state = 1 for single point mode; state = 2 for grid mode) or stops (state = 0) the path mode.

#### getPathPointsAndActions.vi



This VI provides information about the path mode. The outputs are:

- PathPoints: 2D-Array of strings which can be connected to a LabVIEW table indicator. It contains all path points that has been set since the daisy server had been started and shows them clearly in a table.
- Actionlist: 1D-Array that contains all actions, which has been set.
- CoordX: 1D-Array of floats with all X coordinates of all set path points
- CoordY: 1D-Array of floats with all Y coordinates of all set path points

Note that if you defined a grid of points with **definegrid.vi**, this VI will provide the three points, that are necessary to define the grid in the output variables.

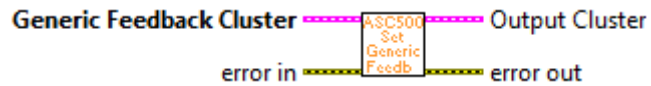
#### getPathModeStatus.vi



Reads out the current status (True = running, false = not active) of the path mode.

### III.4.l. Generic Feedback

#### SetGenericFeedback.vi



This VI changes the settings for a generic feedback. The cluster includes:

- Number (integer) [0 -- *Function 1*; 1 -- *Function 2*] required input
- Input Channel (integer) [0-5 -- *ADC1-6*; 8 -- *AFM df*; 13 -- *AFM Aosc*; 14 -- *AFM Phase*; 16 -- *LockIn Ampl*; 17 -- *LockIn Phase*]
- Output Channel (integer) [0-3 -- *DAC1-4*]
- Min in V (float) [without restraint]
- Max in V (float) [without restraint]
- P (float) [0 to 100]
- I in mHz (integer) [without restraint]
- Setpoint: Unit depends on "Input Channel": mV, Hz or degree (float) [without restraint]
- Invert Polarity (integer) [0 -- *no*; 1 -- *yes*]
- Transferfunction (boolean): starts or stops feedback function

By passing -Inf (float) or -2147483648 (integer) the corresponding value will be read out.

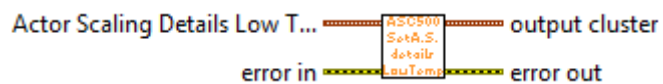
#### GetActualGFOutput.vi



Returns current the current general transfer function output. The 'Number' identifies the transfer function (0 or 1).

### III.4.m. Actor Scaling

#### SetActorScalingDetailsLowT.vi



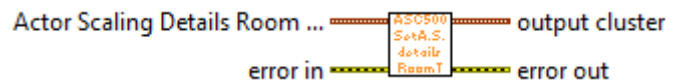
Specifies the details of the low temperature profile. The clusters contains the following variables:

- x, y Max Voltage [V] (float)[1 to 10]
- z Max Voltage [V] (float) [1 to 10]
- x, y Max Range [μm] (float)[0 to 2000000]
- z Max Range [μm] (float) [0 to 2000000]

By passing -Inf (float) or -2147483648 (integer) as input values, the corresponding parameter(s) will be read out.



### SetActorScalingDetailsRoomT.vi

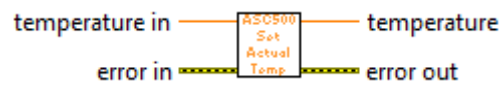


Sets the details for the room temperature actor scaling. The cluster contains:

- x, y Max Voltage (float) [1 to 10]
- z Max Voltage (float) [1 to 10]
- x, y Max Range (float) [0 to 2000000]
- z Max Range (float) [0 to 2000000]

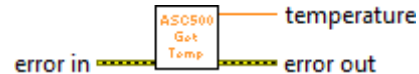
By passing -Inf (float) or -2147483648 (integer) as input values, the corresponding parameter(s) will be read out.

### SetActualTemperature.vi



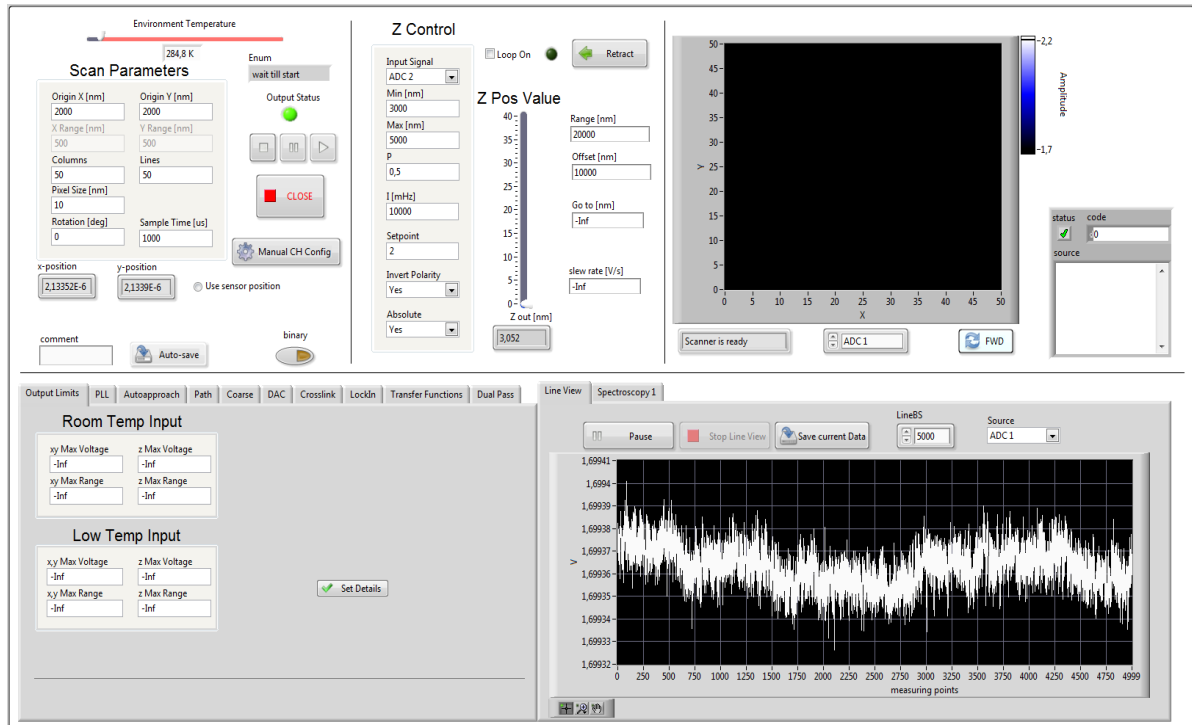
Sets a temperature between 0 and 300 Kelvin. The ASC500 will interpolate the limits based on this temperature and the set limits for "room"- and "low" temperature (Only for V2.7). The unit of the input is Kelvin.

### GetActualTemperature.vi



Retrieves the current temperature in Kelvin.

### III.5. Example VI



The example.vi-file has been made to visualize the usage of the VIs in a program. The layout of the front panel is quite similar to daisy. If you want to realize functionalities of Daisy with LabVIEW, you should find the VIs necessary by double clicking a button or cluster in the example.vi that corresponds to the one in Daisy. This is how you can figure out the structure of your program. In order to allow parallel execution of the VIs in LabVIEW there had to be done some settings inside the VIs which affect their usage as follows.

When you double-click a VI inside the “example.vi” a clone of the selected VI will be opened. As a consequence, you will not be able to change the VI itself. To actually open the VI (not only a clone of it), it is necessary to access it via the explorer or via the LabVIEW-project.

At the top left section, you will find the settings for the scanner. There you can change scan parameters (lines, columns, pixel size ...), start, pause and stop the scanner. In addition, the current x- and y position of the scanner are indicated below the scan parameters cluster. There is a functionality to save the retrieved data into a file. In the input field “Storage path for scanner file” you can enter a path where you want the program to save the output file to. The “comment” input field will be written into the output file in the header. By adjusting the “binary”-button you can change the file type of the output file.

There are some buttons in this section that do not belong to the scan controlling. By clicking the “close” button you can shut down the whole program. The setting for room or low temperature can be configured with a slider (Daisy version 2.7). The LED below indicates the output status. By clicking on the LED you can turn off or on the outputs. Note that it will take

about 2 seconds until the output status changed. By means of the “Manual CH config”-button you can change the settings for the data channels.

The section at the center on the top contains the settings for the z control. The parameters in the cluster are used similar to Daisy.

At the top right section there is an intensity graph to visualize the measured values of the scanner. In contrast to the user interface of Daisy there is only one graph. But you can switch through the different data sources by selecting the desired source beneath the intensity graph. By means of the button at the right below the intensity graph you can switch between the data of the forward and backward scan. There is also an indicator that gives you information about the current status of the scanner. This section also contains the error cluster that informs you about an occurred error.

Similar to Daisy the bottom part is sorted in tabs. The tabs on the left are ordered quite similar to Daisy. On the right there is a tab that contains the graph to visualize the data from the lineView and there is a tab for the spectroscopy. Above the line graph there are some settings for the lineView channel. By means of the LineBS (Line Buffer Size) value, you can control the number of points that are buffered for the lineView. Furthermore, you can adjust the Source the data is coming from, the sample time and you can switch on or off the averaging. The “Spectroscopy”-Tab has been built up similar to the spec-tabs in Daisy.

Now the basic program structure of the example.vi will be explained more precisely. The program mainly consists of four parallel while loops. In addition, there are executed some VIs before and after the loops.

The first thing to do in such a program is to connect with the server. This is done with the init.vi. If you run this example without running Daisy at the same time, it is necessary to load the profile “afm.ngp” which is usually located in the folder Daisy has been installed to. The VI “example\_init” has been created specifically for the example VI and has to be adjusted to every program. It sets some necessary start values and reads out some start values to update the clusters and indicators of the program.

After the first three VIs has been executed successfully, the program will start to execute the four loops:

- GUI-Loop
- Scanner state machine
- LineView loop
- Spectroscopy loop

The GUI-Loop handles the user input. The event structure inside the while loop “waits” until the user changes a value, clicks a button or something like that. Depending on which value has been changed or which button has been clicked the event structure executes a specific case that can be defined by the programmer. The event structure also includes a “timeout” case. If there has not been any input from the user for a definable time this case will be executed.

The scanner state machine executes all steps necessary to use the scanner (registering,

starting scanner, retrieve, visualize and save the data).

The Line- and spectroscopy loops work similarly to the scanner loop. They register a channel (this actually happens outside the Line-loop but inside the Spec-loop), start the operation, retrieve, visualize and save the data. The scanner-, line- and spectroscopy-loops has been separated to allow parallel data acquisition.

If an error occurred or the “close”-button has been clicked, the program will stop the loops and execute the VI “ShutdownServerCon.vi”. It closes the connection to the server and shuts down the server, if there is no other client connected to it. This VI has to be executed at the end of every program in which the init.vi has been executed successfully.

## IV. Individual programs

### IV.1. Pattern of a simple program

The first step in every program involving communication with the ASC500 is to initialize the connection to the server. If there is no server running already on the system the LabVIEW program is executed on, a new server will be started by 'init.vi'. Another possibility is to connect to a running server which is located on a different computer in a local network. Therefore, it is necessary to specify an IP address under which the server can be reached. This address has been entered in the init-cluster in the input field "Server Host" (for a detailed information refer to the description of the init.vi).

The next step is to load a profile file, which is called "afm.ngb". It can be found in the installation directory of Daisy. (for a detailed information refer to the description of the loadprofile.vi).

Once the initialization is done, an obvious step would be to set certain parameters that are needed in the course of the program. This will be done by using a variety of set-VIs.

The next step is probably to acquire some sort of data, which has to be prepared by opening and configuring a data channel. A data channel is used to pass certain types of measured values from the controller to the client program. Important parameters for the channel configuration are the number of the channel (max 13), the data source (e.g. ADC1) and the trigger mechanism (e.g. scanner).

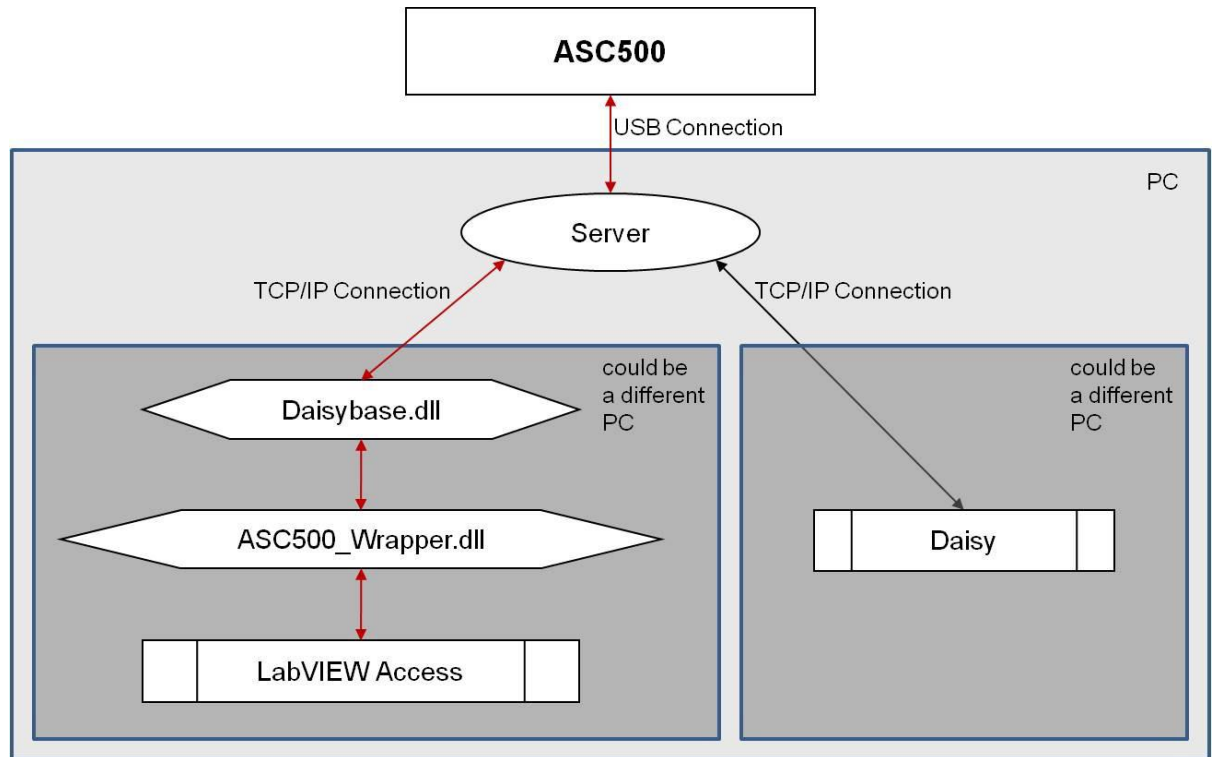
After that, depending on the trigger, it is necessary to start the desired operation (e.g. scan or spectroscopy). Now it is possible to receive the incoming data by calling 'getData.vi/getDataLine.vi', that return data arrays.

As soon as the measurement has been finished and the program shall be stopped, the connection to the server has to be closed. This is done by the VI 'ShutdownServerCon.vi'.

### IV.2. Coexistence of Daisy and a LabVIEW program

Form the fact, that one server is responsible for the communication with the ASC500 and several clients may exist, it is possible and often helpful to operate the LabVIEW program parallel to a running Daisy. Every parameter changed by the LabVIEW program will appear in Daisy and vice versa under the restriction, that in a LabVIEW program no updated value will automatically appear, as one will always have to call a function and ask for the current value.

Another thing to consider is, that changing channel configurations with the LabVIEW program may result in strange behavior in Daisy. This is due to the fact, that Daisy expects certain channels to have certain tasks and there is no notification if the channel trigger is changed. The picture below shows a schematic diagram of how the computer, the server and the ASC500 communicate to each other.



## IV.1. Use of the Data Channels

Any Data produced by the ASC500 (e.g. by performing a scan, an approach or just looking at the LineVIEW) is sent to the server through one (or several) of 14 data channels. Each data channel has 4 Parameters that need to be configured:

1. Signal/Source: Determines what data is sent on this channel.
2. Trigger: Determines when to send the data from the specified source to the server using this data channel. If the trigger is set to "scan" for example, data will be sent via this channel as soon as the scanner has been started.
3. Sample time: Determines the rate at which data from the specified source is sent to the server if the data transfer is being triggered.
4. Averaging: Determines if data being sent through that channel is averaged between each sampled point or if only the last value taken during that interval is sent.

You can check the configuration of the 14 data channels by executing the "getChannelConfig.vi" or by opening the measures.ngc -file in Daisy. If you use the measures.ngc-file, the column "DGroup" can be ignored.

Before you are able to access data from a data channel you have to make sure, that there is at least one channel configured correctly for your purposes. There are two options to achieve that.

1. Stick to the standard channel configuration settings of Daisy
2. Configure a data channel manually. (See description "configChannel.vi")

The standard channel usage as it is used by Daisy is as follows:

0. Scanner 1
1. Scanner 2
2. User
3. Spectroscopy 1
4. Cal Ampl
5. Cal Phase
6. User 2
7. User 3
8. User 4
9. User 5
10. Line View
11. User 6
12. Freq Analysis

### 13. User 7

Within LabVIEW every channel may be used for every purpose, nevertheless it is advisable to stay to this scheme if you want to operate the Daisy in parallel.

## IV.2. Data handling in ASC500\_Wrapper.dll

There has been developed a new data buffer for the remote control of the ASC500.

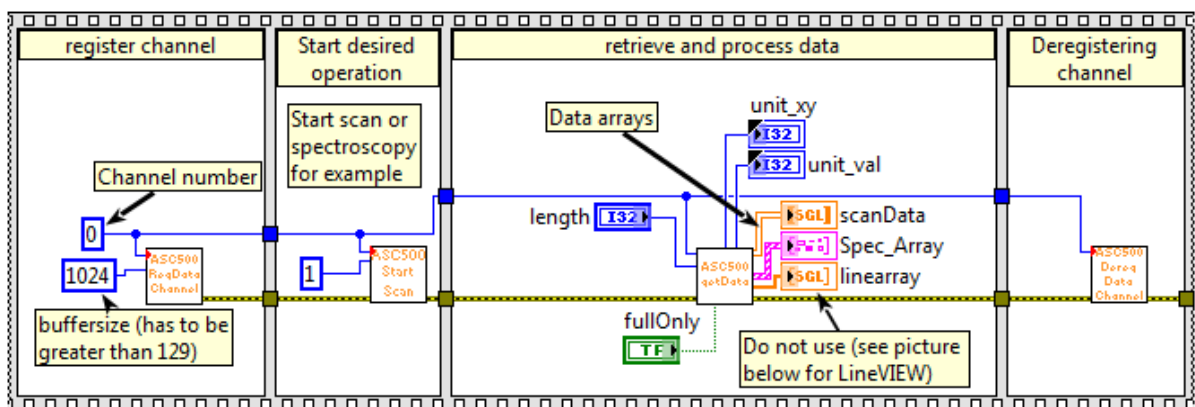
Principle use of the new buffer:

- Firstly, similar to the old buffer, it is necessary to register the data channel to be able to retrieve data from it. This is done by the VI regDataChannel.
- Secondly, if needed, the desired operation (e.g. scan, spectroscopy ...) has to be started (LineVIEW has not to be started).
- Thirdly the corresponding data can be retrieved by using the getData-VI (or the getDataLine-VI for LineVIEW).

If data acquisition has been finished, it is recommendable to deregister the data channel. To do so, execute the VI DeregDataChannel for the specific channel.

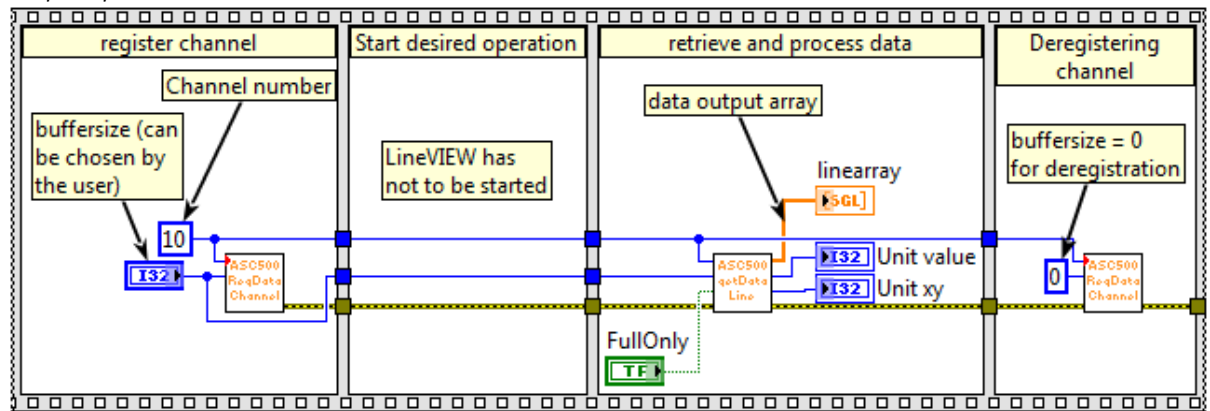
The getData-VI can be used to retrieve data for every trigger except the Permanent. To retrieve data for the LineView the VI getDataLine has to be used (second picture below). To get detailed information about the in- and outputs of the VIs in the picture, refer to the section "Detailed description of Basic VIs".

Steps to perform to access data from a scan or spectroscopy:





Steps to perform to access data from the lineVIEW:



To conclude, from the very beginning, you have to follow these steps to access data from the ASC500:

1. Establish a connection to the server by using the "init.vi"
2. Load the profile file "afm.ngb" by using the "loadprofile.vi" (only necessary when Daisy is NOT running parallel to the LabVIEW program)
3. Set the specific parameters for your needs
4. Configure a data channel by using the "manualConfiguration.vi" or "configChannel.vi"
5. Register the data channel by using "regDataChannel.vi"
6. Start the operation if necessary (scan, spec...)
7. Access data by using "getData.vi" or "getDataLine.vi"
8. Close the connection and shut down the server with the "ShutdownServerCon.vi".

## V. Problem solving

### V.1. Common errors

#### **x.dll was not found**

Apart from the ASC500\_Wrapper.dll there are some other DLLs, which are needed for the ASC500 remote control. Usually they are already placed in the project folder "ASC500\_Files". Please check, if this is still the case. If the DLL of the error message is not in that folder, search for it in the systems folder of windows. Another reason for that error could be that the ASC500\_Wrapper.dll has been placed somewhere else and the program just looks in that folder for the other DLLs. If that is the case, please copy the other DLLs in that folder to the same directory as the ASC500\_Wrapper.dll. Note that it is recommended not to change the storage paths of the DLLs or other files of the LabVIEW project.

#### **LabVIEW keeps asking for the storage path to "ASC500\_Wrapper.dll" when I open a Basic VI**

Usually LabVIEW will ask for the DLL storage path for every Basic VI when opened the first time. That is why it is recommended to open the example.vi once and enter the storage path, as it will then be saved for all subVIs, too. Note that you have to save the VI after entering the path, otherwise the information will be lost and you will be asked again next time you open a VI.

#### **'No contact to the server' in init.vi**

Is the ASC500 connected correctly to the computer? Does daisy.exe start correctly? Is the input-cluster of init.vi filled out correctly (see section "Detailed description of Basic VIs" for more information)?

#### **'Server executable not found' in init.vi**

The path given to the init.vi has to point to the installation directory of Daisy. The important file there is daisysrv.exe.

#### **'Invalid parameter in fct call / Out of range' in a VI**

This happens by trying to pass a value to the ASC500 which is too small or too large. Check if all values make sense. If there is no obvious cause of the error, it might be helpful to try those values given to the relevant VI in daisy.exe and see if they are accepted there.

### **The LabVIEW VIs show strange behavior e.g. when setting or retrieving parameters**

It is important that every program, that established a connection to the server, executes the **ShutdownServerCon.vi** as the last VI executed. If a program stops without executing this VI (e.g. because of a program crash), it can happen that problems with the memory allocation occur. These problems do not cause any instant error popups in some cases. If you notice, that setting or retrieving parameters do not work properly or that retrieving data e.g. from the scanner does not work as expected, we recommend to restart LabVIEW. When you close LabVIEW and there are problems with memory allocation in most cases an error popup comes up telling you that the LabVIEW memory has been corrupted. After restart has been finished the problem should not come up any more.

attocube systems AG  
Eglfinger Weg 2  
D-85540 Haar  
Germany

Phone +49 89 420 797 0  
Fax +49 89 420 797 201 90