Nanonis TCP Protocol

TCP Programming Interface

October 2022 (R12202)

Table of Contents

TCP Protocol

General

Architecture

The Nanonis software works as a TCP Server, and a remote application works as a TCP Client. The TCP Server listens at the ports specified in the Options window (under the System menu). The Client can open one, two, three or four different connections to the Server at these ports.

Each individual connection handles commands serially (one command after another) guaranteeing synchronized execution. These connections can be found and configured in the Options window (under the System menu).

Every message sent from the client to the server (request message), and viceversa (response message) when *Send response back* is set to True in the request message (see Request message>Header section), consists of header and body.

All numeric values are sent in binary form (e.g. a 32 bit integer is encoded in 4 bytes). The storage method of binary encoded numbers is big-endian, that is, the most significant byte is stored at the lowest address.

Data Types

There are thirteen data types which appear in the header and body of both request message and response message:

string
This is an array of characters, where every character has a size of one byte. In the header, the strings have a fixed size (*Command name* is 32 bytes), and in the body, the strings have a variable size which is always prepended as an integer 32.

int
32 bit signed integer. Its range is -2147483648 to 2147483647.

unsigned int16
16 bit unsigned integer. Its range is 0 to 65535.

unsigned int32
32 bit unsigned integer. Its range is 0 to 4294967295.

float32
32 bit (single precision) floating point number.

float64
64 bit (double precision) floating point number.

1D array string
This is a unidimensional array of strings. The array size (number of elements) and its size in bytes are sent as independent arguments right before the array. Each element of the array is obviously a string, preceded by its size. See Functions for more details.

1D array int
This is a unidimensional array of 32 bit signed integers. The array size (number of elements) is usually sent as an independent argument right before the array. See Functions for more details.

1D array unsigned int8
This is a unidimensional array of 8 bit unsigned integers. The array size (number of elements) is usually sent as an independent argument right before the array. See Functions for more details.

1D array unsigned int32
This is a unidimensional array of 32 bit unsigned integers. The array size (number of elements) is usually sent as an independent argument right before the array. See Functions for more details.

1D array float32
This is a unidimensional array of 32 bit floating point numbers. The array size (number of elements) is usually sent as an independent argument right before the array. See Functions for more details.

1D array float64
This is a unidimensional array of 64 bit floating point numbers. The array size (number of elements) is usually sent as an independent argument right before the array. See Functions for more details.

2D array float32
This is a bi-dimensional array of 32 bit floating point numbers. The array size (number of rows and columns) is usually sent as two independent arguments right before the array. See Functions for more details.

2D array string
This is a bi-dimensional array of strings. The array size (number of rows and columns) is usually sent as two independent arguments right before the array. Each element of the array is obviously a string, preceded by its size. See Functions for more details.

Request message

This is the message sent from the client to the server.

Header

Header size is fixed to 40 bytes (last 2 bytes are currently not used and they should be set to zero) and contains the following elements:

- **Command name** (string) (32 bytes) is the name of the executed command. It matches one of the function names described in the Functions section of this document (i.e. *BiasSpectr.Open*). Maximum number of characters is 32.
- **Body size** (int) (4 bytes) is the size of the message body in bytes.
- **Send response back** (unsigned int16) (2 bytes) defines if the server sends a message back to the client (=1) or not (=0). All functions can return at least the error information returned after executing the specified function.

Body

The body size is variable and contains the argument values (if any) sent to the server. Each function has its own arguments described in detail in the Functions section.

Response message

This is only sent from the server to the client if the flag to send the response back (in the request message to the server) is true.

Be aware that without the response message the order of execution between different TCP connections cannot be guaranteed. On the other hand, the order of execution of commands sent through the same TCP connection is guaranteed as the commands are serialized.

Header

Header size is fixed to 40 bytes (last 4 bytes are currently not used and they should be set to zero) and contains the following elements:

- **Command name** (string) (32 bytes) is the name of the executed command. It matches one of the function names described in the Functions section of this document (i.e. *BiasSpectr.Open*). Maximum number of characters is 32.
- **Body size** (int) (4 bytes) is the size of the message body in bytes.

Body

The body size is variable and contains the argument values returned by the function (sent from the server to the client). Each function has its own return arguments described in detail in the Functions section.
After the return arguments values, the body includes the error information containing the following elements:

- **Error status** (unsigned int32) (4 bytes) returns 1=True if there is an error when executing the function.
- **Error description size** (int) (4 bytes) returns the size of the error description which follows.
- **Error description** (string) (variable size) returns the description of the error.

Examples

The following examples show the encoded strings sent through TCP/IP in the request message to execute the desired functions, i.e. from the client (remote application) to the server (Nanonis software).

The encoded strings in the response message received by the client (when the *Send response back* flag in the request message is set to True) are also explained.

The commands are displayed in hexadecimal format (one hexadecimal number pair corresponds to one byte) BUT when sending the TCP commands, the strings (like the command names) should NOT be sent using their hexadecimal representation.

In the examples we use the hexadecimal representation of a string as a way to explain the functions because a string might contain non-printable characters.

Return the tip position

This example uses the function *FolMe.XYPosGet*.

This function returns the X,Y tip coordinates (oversampled during the Acquisition Period time, Tap).
Arguments:

- **Wait for newest data** (unsigned int32) selects whether the function returns the next available signal value or if it waits for a full period of new data.
  If 0, this function returns a value 0 to Tap seconds after being called.
  If 1, the function discards the first oversampled signal value received but returns the second value received. Thus, the function returns a value Tap to 2*Tap seconds after being called

Return arguments (if Send response back flag is set to True when sending request message):

- **X (m)** (float64) is the current X position of the tip
- **Y (m)** (float64) is the current Y position of the tip
- **Error** described in the Response message>Body section

REQUEST MESSAGE:

The Header is fixed always to 40 bytes (last 2 bytes not used), containing the following elements:

- **Command name (string) (maximum is 32 bytes)** is the hexadecimal representation of the command name *FolMe.XYPosGet* padded with zeros to length 32:
  466F 6C4D 652E 5859 506F 7347 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000
- **Body size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the message body in bytes (=4 bytes of the argument *Wait for newest data*) padded with zeros to length 4:
  0000 0004
- **Send response back (unsigned int16) (2 bytes)** defines if the server sends a message back to the client. In this example we set it to True (=1):
  0001
- **Not used (2 bytes)**:
  0000

The Body is of variable size and contains the argument values (if any) sent to the server. In this example this function has the following arguments:

- **Wait for newest data (unsigned int32) (4 bytes)** is the hexadecimal representation of 1 (=true) to length 4:
  0000 0001

Request message					
	Header				Body
	Command name	Body size in bytes	Send response back	Not Used	Wait for newest data
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (2)	Fixed (2)	4
Readable value representation	FolMe.XYPosGet	4	True	-	True
Hex representation of string to be sent over TCP	466F 6C4D 652E 5859 506F 7347 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0004	0001	0000	0000 0001

RESPONSE MESSAGE:

The Header is fixed always to 40 bytes (last 4 bytes not used), containing the following elements:

- **Command name (string) (32 bytes)** is the hexadecimal representation of the command name *FolMe.XYPosGet* padded with zeros to length 32 (like in the request message):
  466F 6C4D 652E 5859 506F 7347 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000
- **Body size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the message body in bytes (=24 bytes of the arguments *X(m)*, *Y(m)*, *Error status*, and *Error size*) padded with zeros to length 4:
  0000 0018
- **Not used (4 bytes)**:
  0000 0000

The Body is of variable size and contains the argument values (if any) sent from the server to the client. In this example this function has the following arguments:

- **X (m) (float64) (8 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 5nm:
  3E35 798E E230 8C3A
- **Y (m) (float64) (8 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to -5nm:
  BE35 798E E230 8C3A
- **Error status (unsigned int32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 0 (no error):
  0000 0000
- **Error size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the error description in bytes (=0 in this case because there is no error):
  0000 0000
- **Error description (string) (variable)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the error description in bytes (=0 in this case because there is no error):
  0000 0000

Response message								
	Header			Body				
	Command name	Body size in bytes	Not Used	X(m)	Y(m)	Error status	Error size in bytes	Error description
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (4)	8	8	4	4	0
Readable value representation	FolMe.XYPosGet	24	-	5n	-5n	False	0	-
Hex representation of string to be sent over TCP	466F 6C4D 652E 5859 506F 7347 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0018	0000 0000	3E35 798E E230 8C3A	BE35 798E E230 8C3A	0000 0000	0000 0000	-

Change the tip position

This example uses the function *FolMe.XYPosSet*.

This function moves the tip to the specified X and Y target coordinates (in meters). It moves at the speed specified by the "Speed" parameter in the Follow Me mode of the Scan Control module.
This function will return when the tip reaches its destination or if the movement stops.
Arguments:

- **X (m)** (float64) sets the target X position of the tip
- **Y (m)** (float64) sets the target Y position of the tip
- **Wait end of move** (unsigned int32) selects whether the function immediately (=0) or if it waits (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

REQUEST MESSAGE:

The Header is fixed always to 40 bytes (last 2 bytes not used), containing the following elements:

- **Command name (string) (32 bytes)** is the hexadecimal representation of the command name *FolMe.XYPosSet* padded with zeros to length 32:
  466F 6C4D 652E 5859 506F 7353 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000
- **Body size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the message body in bytes (=20 bytes of the arguments *X (m)*, *Y (m)*, and *Wait for end of move*) padded with zeros to length 4:
  0000 0014
- **Send response back (unsigned int16) (2 bytes)** defines if the server sends a message back to the client. In this example we set it to True (=1):
  0001
- **Not used (2 bytes)**:
  0000

The Body is of variable size and contains the argument values (if any) sent to the server. In this example this function has the following arguments:

- **X (m) (float64) (8 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 10nm:
  3E45 798E E230 8C3A
- **Y (m) (float64) (8 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 15nm:
  3E50 1B2B 29A4 692B
- **Wait end of move (unsigned int32) (4 bytes)** is the hexadecimal representation of 1 (=true) to length 4:
  0000 0001

Request message							
	Header				Body		
	Command name	Body size in bytes	Send response back	Not Used	X(m)	Y(m)	Wait end-of-motion
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (2)	Fixed (2)	8	8	4
Readable value representation	FolMe.XYPosSet	20	True	-	10n	15n	True
Hex representation of string to be sent over TCP	466F 6C4D 652E 5859 506F 7353 6574 0000 0000 0000 0000 0000 0000 0000 0000	0000 0014	0001	0000	3E45 798E E230 8C3A	3E50 1B2B 29A4 692B	0000 0001

RESPONSE MESSAGE:

The Header is fixed always to 40 bytes (last 4 bytes not used), containing the following elements:

- **Command name (string) (32 bytes)** is the hexadecimal representation of the command name *FolMe.XYPosSet* padded with zeros to length 32 (like in the request message):
  466F 6C4D 652E 5859 506F 7353 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000
- **Body size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the message body in bytes (=8 bytes of the arguments *Error status*, and *Error size*) padded with zeros to length 4:
  0000 0008
- **Not used (4 bytes)**:
  0000 0000

The Body is of variable size and contains the argument values (if any) sent from the server to the client. In this example this function has the following arguments:

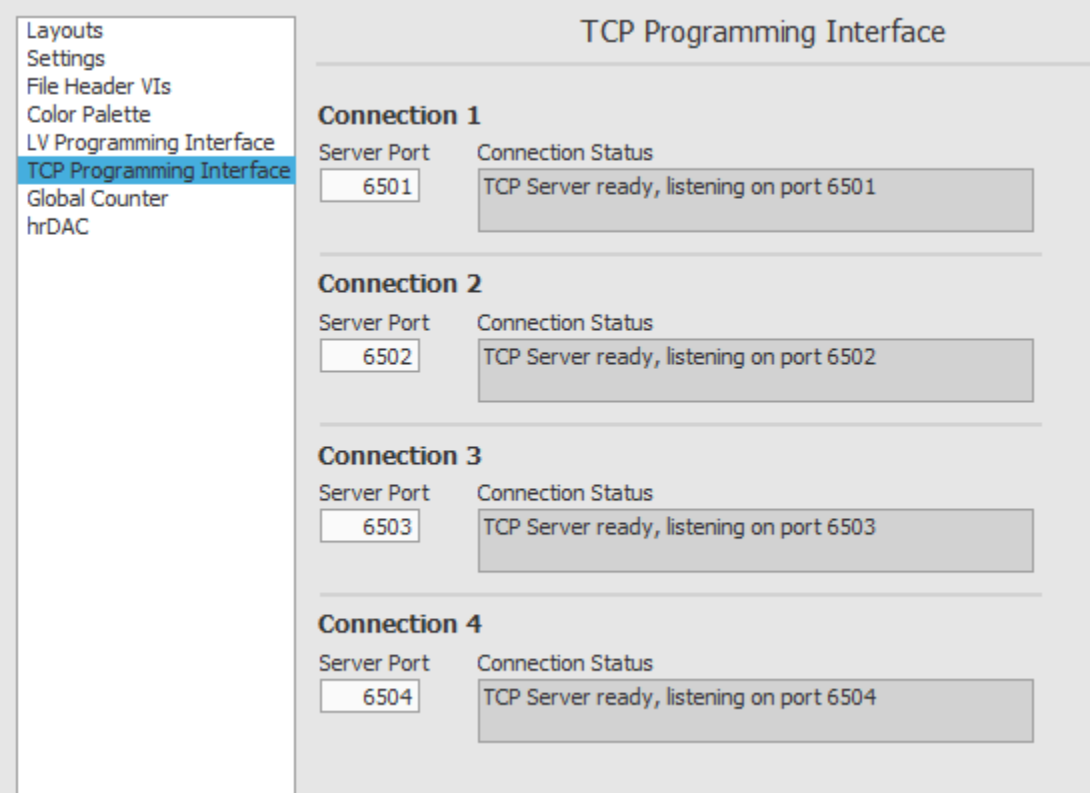
- **Error status (unsigned int32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 0 (no error):
  0000 0000
- **Error size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the error description in bytes (=0 in this case because there is no error):
  0000 0000
- **Error description (string) (variable)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the error description in bytes (=0 in this case because there is no error):

Response message						
	Header			Body		
	Command name	Body size in bytes	Not Used	Error status	Error size in bytes	Error description
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (4)	4	4	0
Readable value representation	FolMe.XYPosSet	8	-	False	0	-
Hex representation of string to be sent over TCP	466F 6C4D 652E 5859 506F 7353 6574 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000 0008	0000 0000	0000 0000	0000 0000	-

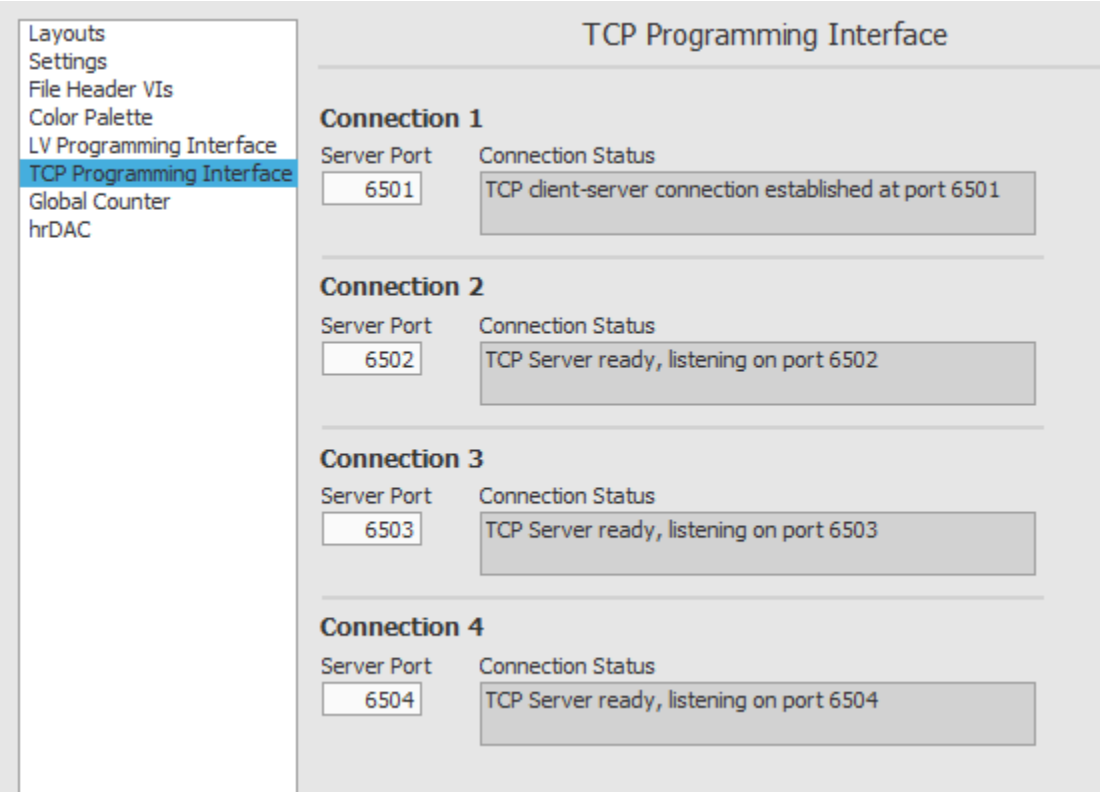
Troubleshooting

1) Make sure that the TCP connections between the TCP client and the Nanonis software (TCP server) are successfully established by using the corresponding remote ports specified in the Nanonis. The default ports can be changed.

   This is available in the TCP Programming Interface section of the Main Options under the System menu. The displayed message shows if the corresponding connection has been established:



Before a TCP Connection is established



After a TCP Connection is established

2) When sending the TCP commands, the strings (like the command names) should NOT be sent using their hexadecimal representation.

   In the examples we use the hexadecimal representation of a string as a way to explain the functions because a string might contain non-printable characters.

3) In the Request message, set the “Send response back” always to true to get a Response message from the Nanonis software.

   If the formatting of the Request message is correct, you will get at least the error information (if any) when executing the function in the Nanonis software.

Functions

Bias Module

Bias.Set

Sets the Bias voltage to the specified value.

Arguments:

- **Bias value (V)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Bias.Get

Returns the Bias voltage value.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Bias value (V)** (float32)
- **Error** described in the Response message>Body section

Bias.RangeSet

Sets the range of the Bias voltage, if different ranges are available.

Arguments:

- **Bias range index** (unsigned int16) is the index out of the list of ranges which can be retrieved by the function *Bias.RangeGet*.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Bias.RangeGet

Returns the selectable ranges of bias voltage and the index of the selected one.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Bias ranges size** (int) is the size in bytes of the bias ranges array
- **Number of ranges** (int) is the number of elements of the bias ranges array
- **Bias ranges** (1D array string) returns an array of selectable bias ranges. Each element of the array is preceded by its size in bytes
- **Bias range index** (unsigned int16) is the index out of the list of bias ranges.
- **Error** described in the Response message>Body section

Bias.CalibrSet

Sets the calibration and offset of bias voltage.

If several ranges are available, this function sets the values for the selected one.

Arguments:

- **Calibration** (float32)
- **Offset** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Bias.CalibrGet

Gets the calibration and offset of bias voltage.

If several ranges are available, this function returns the values of the selected one.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration** (float32)
- **Offset** (float32)
- **Error** described in the Response message>Body section

Bias.Pulse

Generates one bias pulse.

Arguments:

- **Wait until done** (unsigned int32), if True, this function will wait until the pulse has finished. 1=True and 0=False
- **Bias pulse width (s)** (float32) is the pulse duration in seconds
- **Bias value (V)** (float32) is the bias value applied during the pulse
- **Z-Controller on hold** (unsigned int16) sets whether the controller is set to hold (deactivated) during the pulse. Possible values are: 0=no change, 1=hold, 2=don't hold
- **Pulse absolute/relative** (unsigned int16) sets whether the bias value argument is an absolute value or relative to the current bias voltage. Possible values are: 0=no change, 1=relative, 2=absolute

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Bias Sweep

BiasSwp.Open

Opens the Bias Sweep module.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSwp.Start

Starts a bias sweep in the Bias Sweep module.

Before using this function, select the channels to record in the Bias Sweep module.

Arguments:

- **Get data** (unsigned int32) defines if the function returns the sweep data (1=True) or not (0=False)
- **Sweep direction** (unsigned int32) defines if the sweep starts from the lower limit (=1) or from the upper limit (=0)
- **Z-Controller status** (unsigned int32) where 0=no change, 1=turn off, 2=don't turn off
- **Save base name string size** (int) defines the number of characters of the Save base name string
- **Save base name** (string) is the basename used by the saved files. If empty string, there is no change
- **Reset bias** (unsigned int32) where 0=Off, 1=On

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the numer of rows of the Data array
- **Data columns** (int) defines the numer of columns of the Data array
- **Data** (2D array float32) returns the sweep data
- **Error** described in the Response message>Body section

BiasSwp.PropsSet

Sets the configuration of the parameters in the Bias Sweep module.

Arguments:

- **Number of steps** (unsigned int16) defines the number of steps of the sweep. 0 points means no change
- **Period (ms)** (unsigned int16) where 0 means no change
- **Autosave** (unsigned int16) defines if the sweep is automatically saved, where 0=no change, 1=On, 2=Off
- **Save dialog box** (unsigned int16) defines if the save dialog box shows up or not, where 0=no change, 1=On, 2=Off

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSwp.LimitsSet

Sets the limits of Bias in the Bias Sweep module.

Arguments:

- **Lower limit** (float32) defines the lower limit of the sweep range
- **Upper limit** (float32) defines the upper limit of the sweep range

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Bias Spectroscopy

BiasSpectr.Open

Opens the Bias Spectroscopy module.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.Start

Starts a bias spectroscopy in the Bias Spectroscopy module.

Before using this function, select the channels to record in the Bias Spectroscopy module.

Arguments:

- **Get data** (unsigned int32) defines if the function returns the spectroscopy data (1=True) or not (0=False)
- **Save base name string size** (int) defines the number of characters of the Save base name string
- **Save base name** (string) is the basename used by the saved files. If empty string, there is no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the spectroscopy data
- **Number of parameters** (int) is the number of elements of the Parameters array
- **Parameters** (1D array float32) returns the list of fixed parameters and parameters (in that order). To see the names of the returned parameters, use the *BiasSpectr.PropsGet* function.
- **Error** described in the Response message>Body section

BiasSpectr.Stop

Stops the current Bias Spectroscopy measurement.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.StatusGet

Returns the status of the Bias Spectroscopy measurement.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Status (**unsigned int32**)** where 0=not running and 1=running
- **Error** described in the Response message>Body section

BiasSpectr.ChsSet

Sets the list of recorded channels in Bias Spectroscopy.

Arguments:

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.ChsGet

Returns the list of recorded channels in Bias Spectroscopy.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.

- **Error** described in the Response message>Body section

BiasSpectr.PropsSet

Configures the Bias Spectroscopy parameters.

Arguments:

- **Save all** (unsigned int16) where 0 means no change, 1 means that the data from the individual sweeps is saved along with the average data of all of them, and 2 means that the individual sweeps are not saved in the file. This parameter only makes sense when multiple sweeps are configured
- **Number of sweeps** (int) is the number of sweeps to measure and average. 0 means no change with respect to the current selection
- **Backward sweep** (unsigned int16) selects whether to also acquire a backward sweep (forward is always measured) when it is 1. When it is 2 means that no backward sweep is performed, and 0 means no change.
- **Number of points** (int) defines the number of points to acquire over the sweep range, where 0 means no change
- **Z offset (m)** (float32) defines which distance to move the tip before starting the spectroscopy measurement. Positive value means retracting, negative value approaching
- **Autosave** (unsigned int16) selects whether to automatically save the data to ASCII file once the sweep is done (=1). This flag is off when =2, and 0 means no change
- **Show save dialog** (unsigned int16) selects whether to show the save dialog box once the sweep is done (=1). This flag is off when =2, and 0 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.PropsGet

Returns the Bias Spectroscopy parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Save all** (unsigned int16) where 1 means that the data from the individual sweeps is saved along with the average data of all of them, and 0 means that the individual sweeps are not saved in the file. This parameter only makes sense when multiple sweeps are configured
- **Number of sweeps** (int) is the number of sweeps to measure and average
- **Backward sweep** (unsigned int16), where 1 means that the backward sweep is performed (forward is always measured) and 0 means that there is no backward sweep
- **Number of points** (int) is the number of points to acquire over the sweep range
- **Channels size** (int) is the size in bytes of the Channels string array
- **Number of channels** (int) is the number of elements of the Channels string array
- **Channels** (1D array string) returns the names of the acquired channels in the sweep. The size of each string item comes right before it as integer 32
- **Parameters size** (int) is the size in bytes of the Parameters string array
- **Number of parameters** (int) is the number of elements of the Parameters string array
- **Parameters** (1D array string) returns the parameters of the sweep. The size of each string item comes right before it as integer 32
- **Fixed parameters size** (int) is the size in bytes of the Fixed parameters string array
- **Number of fixed parameters** (int) is the number of elements of the Fixed parameters string array
- **Fixed parameters** (1D array string) returns the fixed parameters of the sweep. The size of each string item comes right before it as integer 32
- **Error** described in the Response message>Body section

BiasSpectr.AdvPropsSet

Sets parameters from the Advanced configuration section of the bias spectroscopy module.

Arguments:

- **Reset Bias** (unsigned int16) sets whether Bias voltage returns to the initial value at the end of the spectroscopy measurement. 0 means no change, 1 means On, and 2 means Off
- **Z-Controller Hold** (unsigned int16) sets the Z-Controller on hold during the sweep. 0 means no change, 1 means On, and 2 means Off
- **Record final Z** (unsigned int16) records the Z position during Z averaging time at the end of the sweep and stores the average value in the header of the file when saving. 0 means no change, 1 means On, and 2 means Off
- **Lockin Run** (unsigned int16) sets the Lock-In to run during the measurement.
  When using this feature, make sure the Lock-In is configured correctly and settling times are set to twice the Lock-In period at least. This option is ignored when Lock-In is already running.
  This option is disabled if the Sweep Mode is MLS and the flag to configure the Lock-In per segment in the Multiline segment editor is set. 0 means no change, 1 means On, and 2 means Off

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.AdvPropsGet

Returns the parameters from the Advanced configuration section of the bias spectroscopy module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Reset Bias** (unsigned int16) indicates whether Bias voltage returns to the initial value at the end of the spectroscopy measurement. 0 means Off, 1 means On
- **Z-Controller Hold** (unsigned int16) indicates if the Z-Controller is on hold during the sweep. 0 means Off, 1 means On
- **Record final Z** (unsigned int16) indicates whether to record the Z position during Z averaging time at the end of the sweep and store the average value in the header of the file when saving. 0 means Off, 1 means On
- **Lockin Run** (unsigned int16) indicates if the Lock-In to runs during the measurement.
  This option is ignored when Lock-In is already running.
  This option is disabled if the Sweep Mode is MLS and the flag to configure the Lock-In per segment in the Multiline segment editor is set. 0 means Off, 1 means On
- **Error** described in the Response message>Body section

BiasSpectr.LimitsSet

Sets the Bias spectroscopy limits.

Arguments:

- **Start value (V)** (float32) is the starting value of the sweep
- **End value (V)** (float32) is the ending value of the sweep

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.LimitsGet

Returns the Bias spectroscopy limits.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Start value (V)** (float32) is the starting value of the sweep
- **End value (V)** (float32) is the ending value of the sweep
- **Error** described in the Response message>Body section

BiasSpectr.TimingSet

Configures the Bias spectroscopy timing parameters.

Arguments:

- **Z averaging time (s)** (float32)
- **Z offset (m)** (float32)
- **Initial settling time (s)** (float32)
- **Maximum slew rate (V/s)** (float32)
- **Settling time (s)** (float32)
- **Integration time (s)** (float32)
- **End settling time (s)** (float32)
- **Z control time (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.TimingGet

Returns the Bias spectroscopy timing parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z averaging time (s)** (float32)
- **Z offset (m)** (float32)
- **Initial settling time (s)** (float32)
- **Maximum slew rate (V/s)** (float32)
- **Settling time (s)** (float32)
- **Integration time (s)** (float32)
- **End settling time (s)** (float32)
- **Z control time (s)** (float32)
- **Error** described in the Response message>Body section

BiasSpectr.TTLSyncSet

Sets the configuration of the TTL Synchronization feature in the Advanced section of the Bias Spectroscopy module.

TTL synchronization allows for controlling the high-speed digital outs according to the individual stages of the bias spectroscopy measurement.

Arguments:

- **Enable** (unsigned int16) selects whether the feature is active or not. 0 means no change, 1 means On, and 2 means Off
- **TTL line** (unsigned int16) sets which digital line should be controlled. 0 means no change, 1 means HS Line 1, 2 means HS Line 2, 3 means HS Line 3, 4 means HS Line 4
- **TTL polarity** (unsigned int16) sets the polarity of the switching action. 0 means no change, 1 means Low Active, and 2 means High Active
- **Time to on (s)** (float32) defines the time to wait before activating the TTL line
- **On duration (s)** (float32) defines how long the TTL line should be activated before resetting

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.TTLSyncGet

Returns the configuration of the TTL Synchronization feature in the Advanced section of the Bias Spectroscopy module.

TTL synchronization allows for controlling the high-speed digital outs according to the individual stages of the bias spectroscopy measurement.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Enable** (unsigned int16) indicates whether the feature is active or not. 0 means Off, 1 means On
- **TTL line** (unsigned int16) indicates which digital line should be controlled. 0 means HS Line 1, 1 means HS Line 2, 2 means HS Line 3, 3 means HS Line 4
- **TTL polarity** (unsigned int16) indicates the polarity of the switching action. 0 means Low Active, 1 means High Active
- **Time to on (s)** (float32) indicates the time to wait before activating the TTL line
- **On duration (s)** (float32) indicates how long the TTL line should be activated before resetting
- **Error** described in the Response message>Body section

BiasSpectr.AltZCtrlSet

Sets the configuration of the alternate Z-controller setpoint in the Advanced section of the Bias Spectroscopy module.

When switched on, the Z-controller setpoint is set to the setpoint right after starting the measurement. After changing the setpoint the settling time (s) will be waited for the Z-controller to adjust to the modified setpoint.

Then the Z averaging will start. The original Z-controller setpoint is restored at the end of the measurement, before restoring the Z-controller state.

Arguments:

- **Alternate Z-controller setpoint** (unsigned int16) where 0 means no change, 1 means On, and 2 means Off
- **Setpoint** (float32)
- **Settling time (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.AltZCtrlGet

Returns the configuration of the alternate Z-controller setpoint in the Advanced section of the Bias Spectroscopy module.

When switched on, the Z-controller setpoint is set to the setpoint right after starting the measurement. After changing the setpoint the settling time (s) will be waited for the Z-controller to adjust to the modified setpoint.

Then the Z averaging will start. The original Z-controller setpoint is restored at the end of the measurement, before restoring the Z-controller state.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Alternate Z-controller setpoint** (unsigned int16) where 0 means Off, 1 means On
- **Setpoint** (float32)
- **Settling time (s)** (float32)
- **Error** described in the Response message>Body section

BiasSpectr.MLSLockinPerSegSet

Sets the Lock-In per Segment flag in the Multi line segment editor.

When selected, the Lock-In can be defined per segment in the Multi line segment editor. Otherwise, the Lock-In is set globally according to the flag in the Advanced section of Bias spectroscopy.

Arguments:

- **Lock-In per segment** (unsigned int32) where 0 means Off, 1 means On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.MLSLockinPerSegGet

Returns the Lock-In per Segment flag in the Multi line segment editor.

When selected, the Lock-In can be defined per segment in the Multi line segment editor. Otherwise, the Lock-In is set globally according to the flag in the Advanced section of Bias spectroscopy.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Lock-In per segment** (unsigned int32) where 0 means Off, 1 means On
- **Error** described in the Response message>Body section

BiasSpectr.MLSModeSet

Sets the Bias Spectroscopy sweep mode.

Arguments:

- **Sweep mode** (int) is the number of characters of the sweep mode string.
  If the sweep mode is *Linear*, this value is 6. If the sweep mode is *MLS*, this value is 3
- **Sweep mode** (string) is *Linear* in Linear mode or *MLS* in MultiSegment mode

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.MLSModeGet

Returns the Bias Spectroscopy sweep mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Sweep mode** (int) is the number of characters of the sweep mode string.
  If the sweep mode is *Linear*, this value is 6. If the sweep mode is *MLS*, this value is 3
- **Sweep mode** (string) is *Linear* in Linear mode or *MLS* in MultiSegment mode
- **Error** described in the Response message>Body section

BiasSpectr.MLSValsSet

Sets the bias spectroscopy multiple line segment configuration for Multi Line Segment mode.

Up to 16 distinct line segments may be defined. Any segments beyond the maximum allowed amount will be ignored.

Arguments:

- **Number of segments** (int) indicates the number of segments configured in MLS mode. This value is also the size of the 1D arrays set afterwards
- **Bias start (V)** (1D array float32) is the Start Bias value (V) for each line segment
- **Bias end (V)** (1D array float32 is the End Bias value (V) for each line segment
- **Initial settling time (s)** (1D array float32) indicates the number of seconds to wait at the beginning of each segment after the Lock-In setting is applied
- **Settling time (s)** (1D array float32) indicates the number of seconds to wait before measuring each data point each the line segment
- **Integration time (s)** (1D array float32) indicates the time during which the data are acquired and averaged in each segment
- **Steps** (1D array int) indicates the number of steps to measure in each segment
- **Lock-In run** (1D array unsigned int32) indicates if the Lock-In will run during the segment. This is true only if the global Lock-In per Segment flag is enabled.
  Otherwise, the Lock-In is set globally according to the flag in the Advanced section of Bias spectroscopy

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BiasSpectr.MLSValsGet

Returns the bias spectroscopy multiple line segment configuration for Multi Line Segment mode.

Up to 16 distinct line segments may be defined.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of segments** (int) indicates the number of segments configured in MLS mode. This value is also the size of the 1D arrays set afterwards
- **Bias start (V)** (1D array float32) is the Start Bias value (V) for each line segment
- **Bias end (V)** (1D array float32 is the End Bias value (V) for each line segment
- **Initial settling time (s)** (1D array float32) indicates the number of seconds to wait at the beginning of each segment after the Lock-In setting is applied
- **Settling time (s)** (1D array float32) indicates the number of seconds to wait before measuring each data point each the line segment
- **Integration time (s)** (1D array float32) indicates the time during which the data are acquired and averaged in each segment
- **Steps** (1D array int) indicates the number of steps to measure in each segment
- **Lock-In run** (1D array unsigned int32) indicates if the Lock-In will run during the segment. This is true only if the global Lock-In per Segment flag is enabled.
  Otherwise, the Lock-In is set globally according to the flag in the Advanced section of Bias spectroscopy
- **Error** described in the Response message>Body section

Kelvin Controller

KelvinCtrl.CtrlOnOffSet

Switches the KelvinCtrl. Controller on or off.

Arguments:

- **Control On/Off** (unsigned int32) where 0=Off and 1=On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.CtrlOnOffGet

Returns the status of the KelvinCtrl. Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) where 0=Off and 1=On
- **Error** described in the Response message>Body section

KelvinCtrl.SetpntSet

Sets the KelvinCtrl. Controller setpoint.

Arguments:

- **Setpoint** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.SetpntGet

Returns the KelvinCtrl. Controller setpoint.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Setpoint** (float32)
- **Error** described in the Response message>Body section

KelvinCtrl.GainSet

Sets the regulation loop parameters of the KelvinCtrl. Controller.

Arguments:

- **P-gain** (float32)
- **Time constant (s)** (float32)
- **Slope** (unsigned int16) where 0=no change, 1=Positive, 2=Negative

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.GainGet

Returns the regulation loop parameters of the KelvinCtrl. Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **P-gain** (float32)
- **Time constant (s)** (float32)
- **Slope** (unsigned int16) where 0=Negative and 1=Positive
- **Error** described in the Response message>Body section

KelvinCtrl.ModParamsSet

Returns the modulation parameters of the KelvinCtrl. Controller.

Arguments:

- **Frequency (Hz)** (float32)
- **Amplitude** (float32)
- **Phase (deg)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.ModParamsGet

Returns the modulation parameters of the KelvinCtrl. Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Frequency (Hz)** (float32)
- **Amplitude** (float32)
- **Phase (deg)** (float32)
- **Error** described in the Response message>Body section

KelvinCtrl.ModOnOffSet

Switches the KelvinCtrl. Controller AC mode and modulation.

Arguments:

- **AC mode On/Off** (unsigned int16) where 0=no change, 1=On and 2=Off
- **Modulation On/Off** (unsigned int16) where 0=Off and 1=On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.ModOnOffGet

Returns the status of the KelvinCtrl. Controller AC mode and modulation.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **AC mode status** (unsigned int16) where 0=Off and 1=On
- **Modulation status** (unsigned int16) where 0=Off and 1=On
- **Error** described in the Response message>Body section

KelvinCtrl.CtrlSignalSet

Sets the demodulated/control signal index of the KelvinCtrl. Controller.

Arguments:

- **Demodulated/Control signal index** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.CtrlSignalGet

Returns the demodulated/control signal index of the KelvinCtrl. Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Demodulated/Control signal index** (int)
- **Error** described in the Response message>Body section

KelvinCtrl.AmpGet

Returns the amplitude of the demodulated/control signal of the KelvinCtrl. Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Amplitude** (float32)
- **Error** described in the Response message>Body section

KelvinCtrl.BiasLimitsSet

Sets the bias limits of the KelvinCtrl. Controller.

The bias voltage will be limited to these values as long as the KelvinCtrl. controller is on.

Arguments:

- **Bias high limit (V)** (float32)
- **Bias low limit (V)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

KelvinCtrl.BiasLimitsGet

Returns the bias limits of the KelvinCtrl. Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Bias high limit (V)** (float32)
- **Bias low limit (V)** (float32)
- **Error** described in the Response message>Body section

CPD Compensation

CPDComp.Open

Opens the CPD compensation module.

This module starts automatically bias wobbling and CPD estimating when it opens.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

CPDComp.Close

Closes the CPD compensation module.

This module stops automatically bias wobbling and CPD estimating when it closes.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

CPDComp.ParamsSet

Sets the speed (Hz), the range (V), and the averaging from the CPD compensation module.

Arguments:

- **Speed (Hz)** (float32) sets the frequency of the sawtooth signal used to modulate bias
- **Range (V)** (float32) sets the amplitude of the sawtooth signal used to modulate bias
- **Averaging** (int) is used to average the last specified number of results to calculate the CPD. 0 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

CPDComp.ParamsGet

Sets the speed (Hz), the range (V), and the averaging from the CPD compensation module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Speed (Hz)** (float32) indicates the frequency of the sawtooth signal used to modulate bias
- **Range (V)** (float32) indicates the amplitude of the sawtooth signal used to modulate bias
- **Averaging** (int) is used to average the last specified number of results to calculate the CPD. 0 means no change
- **Error** described in the Response message>Body section

CPDComp.DataGet

Returns the graph data, the CPD estimate (V), and the fit coefficients from the CPD compensation module.

The fit coefficients correspond with the polynomial coefficients in the following fit model:

$df = a(U-Uo)^2 + b(U-Uo) + c$ where Uo is bias voltage.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Size 1** (int) is the number of elements of Bias forward, Frequency shift forward and Frequency Shift fit 1D data arrays. They contain the same number of elements. Bias forward corresponds to the X axis in the graph of the CPD module
- **Bias forward data** (1D array float32)
- **Frequency Shift forward data** (1D array float32)
- **Frequency Shift forward fit data** (1D array float32)
- **Size 2** (int) is the number of elements of Bias backward, Frequency shift backward and Frequency Shift fit 1D data arrays. They contain the same number of elements. Bias backward corresponds to the X axis in the graph of the CPD module
- **Bias backward data** (1D array float32)
- **Frequency Shift backward data** (1D array float32)
- **Frequency Shift backward fit data** (1D array float32)
- **CPD estimate** (float32)
- **a coefficient** (float64)
- **b coefficient** (float64)
- **Error** described in the Response message>Body section

Current Module

Current.Get

Returns the tunneling current value.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Current value (A)** (float32)
- **Error** described in the Response message>Body section

Current.100Get

Returns the tunneling current value of the “Current 100” module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Current 100 value (A)** (float32)
- **Error** described in the Response message>Body section

Current.BEEMGet

Returns the BEEM current value of the corresponding module in a BEEM system.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Current BEEM value (A)** (float32)
- **Error** described in the Response message>Body section

Current.GainSet

Sets the gain of the current amplifier.

Arguments:

- **Gain index** (unsigned int16) is the index out of the list of gains which can be retrieved by the function *Current.GainsGet*.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Current.GainsGet

Returns the selectable gains of the current amplifier and the index of the selected one.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Gains size** (int) is the size in bytes of the Gains array
- **Number of gains** (int) is the number of elements of the Gains array
- **Gains** (1D array string) returns an array of selectable gains. Each element of the array is preceded by its size in bytes
- **Gain index** (unsigned int16) is the index out of the list of gains.
- **Error** described in the Response message>Body section

Current.CalibrSet

Sets the calibration and offset of the selected gain in the Current module.

Arguments:

- **Calibration** (float64)
- **Offset** (float64)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Current.CalibrGet

Gets the calibration and offset of the selected gain in the Current module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration** (float64)
- **Offset** (float64)
- **Error** described in the Response message>Body section

Z-Controller

ZCtrl.ZPosSet

Sets the Z position of the tip.

Note: to change the Z-position of the tip, the Z-controller must be switched OFF.

Arguments:

- **Z position (m)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.ZPosGet

Returns the current Z position of the tip.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z position (m)** (float32)
- **Error** described in the Response message>Body section

ZCtrl.OnOffSet

Switches the Z-Controller On or Off.

Arguments:

- **Z-Controller status** (unsigned int32) switches the controller Off (=0) or On (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.OnOffGet

Returns the status of the Z-Controller.

This function returns the status from the real-time controller (i.e. not from the Z-Controller module).

This function is useful to make sure that the Z-controller is really off before starting an experiment. Due to the communication delay, switch-off delay... sending the off command with the *ZCtrl.OnOffGet* function might take some time before the controller is off.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z-Controller status** (unsigned int32) indicates if the controller is Off (=0) or On (=1)
- **Error** described in the Response message>Body section

ZCtrl.SetpntSet

Sets the setpoint of the Z-Controller.

Arguments:

- **Z-Controller setpoint** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.SetpntGet

Returns the setpoint of the Z-Controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z-Controller setpoint** (float32)
- **Error** described in the Response message>Body section

ZCtrl.GainSet

Sets the Z-Controller gains (P, I) and time settings.

The integral gain is calculated based on the P-gain and the Time constant as follows: I=P/T.

Arguments:

- **P-gain** (float32) is the proportional gain of the regulation loop
- **Time constant (s)** (float32) is the time constant T
- **I-gain** (float32) is the integral gain of the regulation loop (I=P/T)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.GainGet

Returns the Z-Controller gains (P, I) and time settings.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **P-gain** (float32) is the proportional gain of the regulation loop
- **Time constant (s)** (float32) is the time constant T
- **I-gain** (float32) is the integral gain of the regulation loop (I=P/T)
- **Error** described in the Response message>Body section

ZCtrl.SwitchOffDelaySet

Sets the switch off delay in seconds of the Z-Controller.

Before turning off the controller, the Z position is averaged over this time delay. The tip is then positioned at the averaged value. This leads to reproducible Z positions when switching off the Z-controller.

Arguments:

- **Z-Controller switch off delay (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.SwitchOffDelayGet

Returns the switch off delay in seconds of the Z-Controller.

Before turning off the controller, the Z position is averaged over this time delay. The tip is then positioned at the averaged value. This leads to reproducible Z positions when switching off the Z-controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z-Controller switch off delay (s)** (float32)
- **Error** described in the Response message>Body section

ZCtrl.TipLiftSet

Sets the TipLift of the Z-Controller.

Retracts the tip by the specified amount when turning off the Z-controller.

Arguments:

- **TipLift (m)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.TipLiftGet

Returns the TipLift of the Z-Controller.

Retracts the tip by the specified amount when turning off the Z-controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **TipLift (m)** (float32)
- **Error** described in the Response message>Body section

ZCtrl.Home

Moves the tip to its home position.

This function moves the tip to the home position defined by the Home Absolute (m)/ Home Relative (m) value. (Absolute and relative can be switched in the controller configuration panel in the software).

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.HomePropsSet

Sets the current status of the Z-Controller Home switch and its corresponding position.

The Home position can be absolute (fixed position) or relative to the current tip position.

Arguments:

- **Relative or Absolute** (unsigned int16), where 0 means no change with respect to the current selection, 1 means that the home position is absolute to the current position, and 2 means that it is relative to the current position
- **Home position (m)** (float32) is the home position value in meters

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.HomePropsGet

Returns the current status of the Z-Controller Home switch and its corresponding position.

The Home position can be absolute (fixed position) or relative to the current tip position.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Relative or Absolute** (unsigned int16), where 0 means that the home position is absolute to the current position, and 1 means that it is relative to the current position
- **Home position (m)** (float32) is the home position value in meters
- **Error** described in the Response message>Body section

ZCtrl.ActiveCtrlSet

Sets the active Z-Controller.

Arguments:

- **Z-Controller index** (int) is the index out of the list of controllers which can be retrieved by the function *ZCtrl.ControllersListGet*.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.CtrlListGet

Returns the list of Z-Controllers and the index of the active controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **List of controllers size** (int) is the size in bytes of the List of controllers array
- **Number of controllers** (int) is the number of elements of the List of controllers array
- **List of controllers** (1D array string) returns an array of the available Z-Controllers. Each element of the array is preceded by its size in bytes
- **Active Z-Controller index** (int) is the index out of the list of gains.
- **Error** described in the Response message>Body section

ZCtrl.Withdraw

Withdraws the tip.

This function switches off the Z-Controller and then fully withdraws the tip (to the upper limit of the Z-piezo range).

Arguments:

- **Wait until finished** (unsigned int32) indicates if the function waits until the tip is fully withdrawn (=1) or it does not wait (=0)
- **Timeout (ms)** (int) is time in ms this function waits. Set it to -1 to wait indefinitely.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.WithdrawRateSet

Sets the Z-Controller withdraw slew rate in meters per second.

Arguments:

- **Withdraw slew rate (m/s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.WithdrawRateGet

Returns the Z-Controller withdraw slew rate in meters per second.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Withdraw slew rate (m/s)** (float32)
- **Error** described in the Response message>Body section

ZCtrl.LimitsEnabledSet

Enables or disables the Z position limits.

Arguments:

- **Limit Z status** (unsigned int32) enables the Z limits (=1) or disables them (=0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.LimitsEnabledGet

Returns if the Z limits are enabled or disabled.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Limit Z status** (unsigned int32) indicates if the Z limits are disabled (=0) or enabled (=1)
- **Error** described in the Response message>Body section

ZCtrl.LimitsSet

Sets the Z position high and low limits in meters.

When the Z position limits are not enabled, this function has no effect.

Arguments:

- **Z high limit (m)** (float32)
- **Z low limit (m)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZCtrl.LimitsGet

Returns the Z position high and low limits in meters.

When the Z position limits are not enabled, they correspond to the piezo range limits.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z high limit (m)** (float32)
- **Z low limit (m)** (float32)
- **Error** described in the Response message>Body section

ZCtrl.StatusGet

Returns the current status of the Z-Controller module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z -Controller Status** (unsigned int16) returns if the controller is Off (1), On (2), on Hold (3), switching Off (4), if a Safe Tip event occurred (5), or if the tip is currently withdrawing (6)
- **Error** described in the Response message>Body section

Safe Tip

SafeTip.OnOffSet

Switches the Safe Tip feature on or off.

Arguments:

- **Safe Tip status** (unsigned int16) sets if the Safe Tip is On (=1) or Off (=2), or if it does not change (=0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SafeTip.OnOffGet

Returns the on-off status of the Safe Tip.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Safe Tip status** (unsigned int16) indicates if the Safe Tip is Off (=0) or On (=1)
- **Error** described in the Response message>Body section

SafeTip.SignalGet

Returns the current Safe Tip signal value.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Signal value** (float32)
- **Error** described in the Response message>Body section

SafeTip.PropsSet

Sets the Safe Tip configuration.

Arguments:

- **Auto recovery** (unsigned int16) indicates if Z-controller automatically recovers from a SafeTip situation after a specified amount of time if Z-Controller was originally on. 0 means Off, 1 means On
- **Auto pause scan** (unsigned int16) indicates if the Z-controller automatically pauses/holds the scan on a SafeTip event. 0 means Off, 1 means On
- **Threshold** (float32) defines the condition to trigger the Safe Tip

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SafeTip.PropsGet

Returns the Safe Tip configuration.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Auto recovery** (unsigned int16) indicates if Z-controller automatically recovers from a SafeTip situation after a specified amount of time if Z-Controller was originally on. 0 means Off, 1 means On
- **Auto pause scan** (unsigned int16) indicates if the Z-controller automatically pauses/holds the scan on a SafeTip event. 0 means Off, 1 means On
- **Threshold** (float32) defines the condition to trigger the Safe Tip
- **Error** described in the Response message>Body section

Auto Approach

AutoApproach.Open

Opens the Auto-Approach module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

AutoApproach.OnOffSet

Starts or stops the Z auto- approach procedure.

Arguments:

- **On/Off** (unsigned int16) starts the auto-approach procedure (=1) or stops it (=0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

AutoApproach.OnOffGet

Returns the on-off status of the Z auto- approach procedure.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int16) indicates if the auto-approach procedure is Off (=0) or running (=1)
- **Error** described in the Response message>Body section

Z Spectroscopy

ZSpectr.Open

Opens the Z Spectroscopy module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.Start

Starts a Z spectroscopy in the Z Spectroscopy module.

Before using this function, select the channels to record in the Z Spectroscopy module.

Arguments:

- **Get data** (unsigned int32) defines if the function returns the spectroscopy data (1=True) or not (0=False)
- **Save base name string size** (int) defines the number of characters of the Save base name string
- **Save base name** (string) is the basename used by the saved files. If empty string, there is no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the spectroscopy data
- **Number of parameters** (int) is the number of elements of the Parameters array
- **Parameters** (1D array float32) returns the list of fixed parameters and parameters (in that order). To see the names of the returned parameters, use the Z*Spectr*.*PropsGet* function.
- **Error** described in the Response message>Body section

ZSpectr.Stop

Stops the current Z Spectroscopy measurement.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.StatusGet

Returns the status of the Z Spectroscopy measurement.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status (**unsigned int32**)** where 0=not running and 1=running
- **Error** described in the Response message>Body section

ZSpectr.ChsSet

Sets the list of recorded channels in Z Spectroscopy.

Arguments:

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.ChsGet

Returns the list of recorded channels in Z Spectroscopy.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.
- **Error** described in the Response message>Body section

ZSpectr.PropsSet

Configures the Z Spectroscopy parameters.

Arguments:

- **Backward sweep** (unsigned int16) selects whether to also acquire a backward sweep (forward is always measured) when it is 1. When it is 2 means that no backward sweep is performed, and 0 means no change.
- **Number of points** (int) defines the number of points to acquire over the sweep range, where 0 means no change
- **Number of sweeps** (unsigned int16) is the number of sweeps to measure and average. 0 means no change with respect to the current selection
- **Autosave** (unsigned int16) selects whether to automatically save the data to ASCII file once the sweep is done (=1). This flag is off when =2, and 0 means no change
- **Show save dialog** (unsigned int16) selects whether to show the save dialog box once the sweep is done (=1). This flag is off when =2, and 0 means no change
- **Save all** (unsigned int16) where 0 means no change, 1 means that the data from the individual sweeps is saved along with the average data of all of them, and 2 means that the individual sweeps are not saved in the file. This parameter only makes sense when multiple sweeps are configured

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.PropsGet

Returns the Z Spectroscopy parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Backward sweep** (unsigned int16), where 1 means that the backward sweep is performed (forward is always measured) and 0 means that there is no backward sweep
- **Number of points** (int) is the number of points to acquire over the sweep range
- **Channels size** (int) is the size in bytes of the Channels string array
- **Number of channels** (int) is the number of elements of the Channels string array
- **Channels** (1D array string) returns the names of the acquired channels in the sweep. The size of each string item comes right before it as integer 32
- **Parameters size** (int) is the size in bytes of the Parameters string array
- **Number of parameters** (int) is the number of elements of the Parameters string array
- **Parameters** (1D array string) returns the parameters of the sweep. The size of each string item comes right before it as integer 32
- **Fixed parameters size** (int) is the size in bytes of the Fixed parameters string array
- **Number of fixed parameters** (int) is the number of elements of the Fixed parameters string array
- **Fixed parameters** (1D array string) returns the fixed parameters of the sweep. The size of each string item comes right before it as integer 32
- **Number of sweeps** (unsigned int16) is the number of sweeps to measure and average
- **Save all** (unsigned int16) where 1 means that the data from the individual sweeps is saved along with the average data of all of them, and 0 means that the individual sweeps are not saved in the file. This parameter only makes sense when multiple sweeps are configured
- **Error** described in the Response message>Body section

ZSpectr.AdvPropsSet

Sets parameters from the Advanced configuration section of the Z spectroscopy module.

Arguments:

- **Time between forward and backward sweep (s)** (float32)
- **Record final Z** (unsigned int16) if on, the final Z position is averaged during Z averaging time after Z control time at the end of the sweep. 0 means no change, 1 means On, and 2 means Off
- **Lockin Run** (unsigned int16) sets the Lock-In to run during the measurement.
  When using this feature, make sure the Lock-In is configured correctly and settling times are set to twice the Lock-In period at least. This option is ignored when Lock-In is already running.
  This option is disabled if the Sweep Mode is MLS and the flag to configure the Lock-In per segment in the Multiline segment editor is set. 0 means no change, 1 means On, and 2 means Off
- **Reset Z** (unsigned int16) if on, the Z position is set back to the initial value at the end of the sweep. If off, the Z position stays at the last value at the end of the sweep.
  Be aware that if the Z-Controller is on and Z is not reset, the Z position will anyway be automatically controlled by the Z-Controller. 0 means no change, 1 means On, and 2 means Off

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.AdvPropsGet

Sets parameters from the Advanced configuration section of the Z spectroscopy module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Time between forward and backward sweep (s)** (float32)
- **Record final Z** (unsigned int16) if on, the final Z position is averaged during Z averaging time after Z control time at the end of the sweep. 0 means Off, 1 means On
- **Lockin Run** (unsigned int16) indicates if the Lock-In runs during the measurement.
  When using this feature, make sure the Lock-In is configured correctly and settling times are set to twice the Lock-In period at least. This option is ignored when Lock-In is already running.
  This option is disabled if the Sweep Mode is MLS and the flag to configure the Lock-In per segment in the Multiline segment editor is set. 0 means Off, 1 means On
- **Reset Z** (unsigned int16) if on, the Z position is set back to the initial value at the end of the sweep. If off, the Z position stays at the last value at the end of the sweep.
  Be aware that if the Z-Controller is on and Z is not reset, the Z position will anyway be automatically controlled by the Z-Controller. 0 means Off, 1 means On
- **Error** described in the Response message>Body section

ZSpectr.RangeSet

Sets the Z-spectroscopy range settings.

Arguments:

- **Z offset (m)** (float32) defines the offset to apply before starting the sweep
- **Z sweep distance (m)** (float32) defines the sweep span

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.RangeGet

Returns the Z-spectroscopy range settings.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z offset (m)** (float32) defines the offset to apply before starting the sweep
- **Z sweep distance (m)** (float32) defines the sweep span
- **Error** described in the Response message>Body section

ZSpectr.TimingSet

Configures the Z spectroscopy timing parameters.

Arguments:

- **Z averaging time (s)** (float32)
- **Initial settling time (s)** (float32)
- **Maximum slew rate (V/s)** (float32)
- **Settling time (s)** (float32)
- **Integration time (s)** (float32)
- **End settling time (s)** (float32)
- **Z control time (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.TimingGet

Returns the Z spectroscopy timing parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Z averaging time (s)** (float32)
- **Initial settling time (s)** (float32)
- **Maximum slew rate (V/s)** (float32)
- **Settling time (s)** (float32)
- **Integration time (s)** (float32)
- **End settling time (s)** (float32)
- **Z control time (s)** (float32)
- **Error** described in the Response message>Body section

ZSpectr.RetractDelaySet

Sets the Z-spectroscopy retract delay.

Arguments:

- **Retract delay (s)** (float32) defines the delay in seconds between forward sweep and backward sweep

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.RetractDelayGet

Returns the Z-spectroscopy retract delay.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Retract delay (s)** (float32) defines the delay in seconds between forward sweep and backward sweep
- **Error** described in the Response message>Body section

ZSpectr.RetractSet

Sets the configuration for the main condition of the Auto Retract in the Z-Spectroscopy module.

Arguments:

- **Enable** (unsigned int16) switches the Auto Retract on or off. 0 means no change, 1 means On, and 2 means Off
- **Threshold** (float32) combined with the comparison, sets which situation triggers the main condition to auto-retract the tip
- **Signal index** (int) sets the index between 0-127 of the signal used to check the main retract condition. Use -1 to leave the value unchanged in the Nanonis software
- **Comparison** (unsigned int16) sets which situation triggers the main condition to auto-retract the tip. 0 means >, 1 means <, and 2 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.RetractGet

Sets the configuration for the main condition of the Auto Retract in the Z-Spectroscopy module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Enable** (unsigned int16) indicates if the Auto Retract is on or off. 0 means Off, 1 means On
- **Threshold** (float32) combined with the comparison, defines which situation triggers the main condition to auto-retract the tip
- **Signal index** (int) is the index between 0-127 of the signal used to check the main retract condition. Use -1 to leave the value unchanged in the Nanonis software
- **Comparison** (unsigned int16) defines which situation triggers the main condition to auto-retract the tip. 0 means >, 1 means <
- **Error** described in the Response message>Body section

ZSpectr.Retract2ndSet

Sets the configuration for the 2nd condition of the Auto Retract in the Z-Spectroscopy module.

Arguments:

- **2nd condition** (int) configures the use of a second signal comparison in combination with the main Auto Retract signal comparison. Possible values are 0=no change, 1=-No-, 2=OR, 3=AND, 4=THEN, where:

  -No-: disables the use of a second signal comparison
  OR: Auto-Retract will execute if the 1st or the 2nd condition is met
  AND: Auto-Retract will execute if the 1st and the 2nd condition are met at the same time
  THEN: the 2nd condition is only checked once the 1st condition has been met

- **Threshold** (float32) combined with the comparison, sets which situation triggers the 2nd condition to auto-retract the tip
- **Signal index** (int) sets the index between 0-127 of the signal used to check the 2nd retract condition. Use -1 to leave the value unchanged in the Nanonis software
- **Comparison** (unsigned int16) sets which situation triggers the 2nd condition to auto-retract the tip. 0 means >, 1 means <, and 2 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

ZSpectr.Retract2ndGet

Returns the configuration for the 2nd condition of the Auto Retract in the Z-Spectroscopy module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **2nd condition** (int) indicates the use of a second signal comparison in combination with the main Auto Retract signal comparison. Possible values are 0=-No-, 1=OR, 2=AND, 3=THEN, where:

  -No-: disables the use of a second signal comparison
  OR: Auto-Retract will execute if the 1st or the 2nd condition is met
  AND: Auto-Retract will execute if the 1st and the 2nd condition are met at the same time
  THEN: the 2nd condition is only checked once the 1st condition has been met

- **Threshold** (float32) combined with the comparison, indicates which situation triggers the 2nd condition to auto-retract the tip
- **Signal index** (int) indicates the index between 0-127 of the signal used to check the 2nd retract condition
- **Comparison** (unsigned int16) indicates which situation triggers the 2nd condition to auto-retract the tip. 0 means >, 1 means <
- **Error** described in the Response message>Body section

Piezos

Piezo.TiltSet

Configures the tilt correction parameters.

Arguments:

- **Tilt X (deg)** (float32) sets by which angle to correct the tilt in the X direction
- **Tilt Y (deg)** (float32) sets by which angle to correct the tilt in the Y direction

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Piezo.TiltGet

Returns the tilt correction parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Tilt X (deg)** (float32) indicates by which angle to correct the tilt in the X direction
- **Tilt Y (deg)** (float32) indicates by which angle to correct the tilt in the Y direction
- **Error** described in the Response message>Body section

Piezo.RangeSet

Sets the piezo range (m) values for all 3 axes (X, Y, Z).

Changing the range will also change the sensitivity (HV gain will remain unchanged).

Arguments:

- **Range X (m)** (float32)
- **Range Y (m)** (float32)
- **Range Z (m)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Piezo.RangeGet

Returns the piezo range (m) values for all 3 axes (X, Y, Z).

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Range X (m)** (float32)
- **Range Y (m)** (float32)
- **Range Z (m)** (float32)
- **Error** described in the Response message>Body section

Piezo.SensSet

Sets the piezo sensitivity (m/V) values for all 3 axes (X, Y, Z).

Changing the sensitivity will also change the range (HV gain will remain unchanged).

Arguments:

- **Calibration X (m/V)** (float32)
- **Calibration Y (m/V)** (float32)
- **Calibration Z (m/V)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Piezo.SensGet

Returns the piezo sensitivity (m/V) values for all 3 axes (X, Y, Z).

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration X (m/V)** (float32)
- **Calibration Y (m/V)** (float32)
- **Calibration Z (m/V)** (float32)
- **Error** described in the Response message>Body section

Piezo.DriftCompSet

Configures the drift compensation parameters.

Arguments:

- **Compensation on/off** (int) activates or deactivates the drift compensation, where -1=no change, 0=Off, 1=On
- **Vx (m/s)** (float32) is the linear speed applied to the X piezo to compensate the drift
- **Vy (m/s)** (float32) is the linear speed applied to the Y piezo to compensate the drift
- **Vz (m/s)** (float32) is the linear speed applied to the Z piezo to compensate the drift
- **Saturation limit (%)** (float32) is the drift saturation limit in percent of the full piezo range and it applies to all axes

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Piezo.DriftCompGet

Returns the drift compensation settings and information.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Compensation status** (unsigned int32) indicates whether the drift compensation is On or Off
- **Vx (m/s)** (float32) is the linear speed applied to the X piezo to compensate the drift
- **Vy (m/s)** (float32) is the linear speed applied to the Y piezo to compensate the drift
- **Vz (m/s)** (float32) is the linear speed applied to the Z piezo to compensate the drift
- **X saturated status** (unsigned int32) indicates if the X drift correction reached 10% of the piezo range. When this happens, the drift compensation stops for this axis and its LED turns on. To reactivate the compensation, switch the drift compensation off and on
- **Y saturated status** (unsigned int32) indicates if the Y drift correction reached 10% of the piezo range. When this happens, the drift compensation stops for this axis and its LED turns on. To reactivate the compensation, switch the drift compensation off and on
- **Z saturated status** (unsigned int32) indicates if the Z drift correction reached 10% of the piezo range. When this happens, the drift compensation stops for this axis and its LED turns on. To reactivate the compensation, switch the drift compensation off and on
- **Saturation limit (%)** (float32) is the drift saturation limit in percent of the full piezo range and it applies to all axes
- **Error** described in the Response message>Body section

Piezo.CalibrGet

Returns the piezo calibration values for all 3 axes (X, Y, Z).

The calibration returned is for the low voltage signals, i.e. the +/-10V signals before the HV amplifier.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration X (m/V)** (float32)
- **Calibration Y (m/V)** (float32)
- **Calibration Z (m/V)** (float32)
- **Error** described in the Response message>Body section

Piezo.HVAInfoGet

Returns the HVA gain readout information.

If the HVA gain readout is not enabled, this function returns a warning.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Gain AUX** (float32)
- **Gain X** (float32)
- **Gain Y** (float32)
- **Gain Z** (float32)
- **X/Y enabled status**(unsigned int32)
- **Z enabled status** (unsigned int32)
- **AUX enabled status** (unsigned int32)
- **Error** described in the Response message>Body section

Piezo.HVAStatusLEDGet

Returns the HVA LED status readout information.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Overheated status**(unsigned int32)
- **HV supply status** (unsigned int32)
- **High temperature status** (unsigned int32)
- **Output connector status** (unsigned int32)
- **Error** described in the Response message>Body section

Scan

Scan.Action

Starts, stops, pauses or resumes a scan.

Arguments:

- **Scan action** (unsigned int16) sets which action to perform, where 0 means Start, 1 is Stop, 2 is Pause, and 3 is Resume
- **Scan direction** (unsigned int32) that if 1, scan direction is set to up. If 0, direction is down

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Scan.StatusGet

Returns if the scan is running or not.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Scan status** (unsigned int32) means that if it is 1, scan is running. If 0, scan is not running
- **Error** described in the Response message>Body section

Scan.WaitEndOfScan

Waits for the End-of-Scan.

This function returns only when an End-of-Scan or timeout occurs (whichever occurs first).

Arguments:

- **Timeout (ms)** (int) sets how many milliseconds this function waits for an End-of-Scan. If –1, it waits indefinitely

Return arguments (if Send response back flag is set to True when sending request message):

- **Timeout status status** (unsigned int32) means that if it is 1, the function timed-out. If 0, it didn't time-out
- **File path size** (unsigned int32) is the number of bytes corresponding to the File path string
- **File path** (string) returns the path where the data file was automatically saved (if auto-save was on). If no file was saved at the End-of-Scan, it returns an empty path
- **Error** described in the Response message>Body section

Scan.FrameSet

Configures the scan frame parameters.

Arguments:

- **Center X (m)** (float32) is the X position of the scan frame center
- **Center Y (m)** (float32) is the Y position of the scan frame center
- **Width (m)** (float32) is the width of the scan frame
- **Height (m)** (float32) is the height of the scan frame
- **Angle (deg)** (float32) is the angle of the scan frame (positive angle means clockwise rotation)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Scan.FrameGet

Returns the scan frame parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Center X (m)** (float32) is the X position of the scan frame center
- **Center Y (m)** (float32) is the Y position of the scan frame center
- **Width (m)** (float32) is the width of the scan frame
- **Height (m)** (float32) is the height of the scan frame
- **Angle (deg)** (float32) is the angle of the scan frame (positive angle means clockwise rotation)
- **Error** described in the Response message>Body section

Scan.BufferSet

Configures the scan buffer parameters.

Arguments:

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.
- **Pixels** (int) is the number of pixels per line.
  In the scan control module this value is coerced to the closest multiple of 16, because the scan data is sent from the RT to the host in packages of 16 pixels
- **Lines** (int) is the number of scan lines.
  Be aware that if the chain button to keep the scan resolution ratio in the scan control module is active and the number of lines is set to 0 or left unconnected, the number of lines will automatically coerce to keep the scan resolution ratio according to the new number of pixels.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Scan.BufferGet

Returns the scan buffer parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.
- **Pixels** (int) is the number of pixels per line
- **Lines** (int) is the number of scan lines
- **Error** described in the Response message>Body section

Scan.PropsSet

Configures some of the scan parameters.

Arguments:

- **Continuous scan** (unsigned int32) sets whether the scan continues or stops when a frame has been completed. 0 means no change, 1 is On, and 2 is Off
- **Bouncy scan** (unsigned int32) sets whether the scan direction changes when a frame has been completed. 0 means no change, 1 is On, and 2 is Off
- **Autosave** (unsigned int32) defines the save behavior when a frame has been completed. "All" saves all the future images. "Next" only saves the next frame. 0 means no change, 1 is All, 2 is Next, and 3 sets this feature Off
- **Series name size** (int) is the size in bytes of the Series name string
- **Series name** (string) is base name used for the saved images
- **Comment size** (int) is the size in bytes of the Comment string
- **Comment** (string) is comment saved in the file

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Scan.PropsGet

Returns some of the scan parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Continuous scan** (unsigned int32) indicates whether the scan continues or stops when a frame has been completed. 0 means Off, and 1 is On
- **Bouncy scan** (unsigned int32) indicates whether the scan direction changes when a frame has been completed. 0 means Off, and 1 is On
- **Autosave** (unsigned int32) defines the save behavior when a frame has been completed. "All" saves all the future images. "Next" only saves the next frame. 0 is All, 1 is Next, and 2 means Off
- **Series name size** (int) is the size in bytes of the Series name string
- **Series name** (string) is base name used for the saved images
- **Comment size** (int) is the size in bytes of the Comment string
- **Comment** (string) is comment saved in the file
- **Error** described in the Response message>Body section

Scan.SpeedSet

Configures the scan speed parameters.

Arguments:

- **Forward linear speed (m/s)** (float32)
- **Backward linear speed (m/s)** (float32)
- **Forward time per line (s)** (float32)
- **Backward time per line (s)** (float32)
- **Keep parameter constant** (unsigned int16) defines which speed parameter to keep constant, where 0 means no change, 1 keeps the linear speed constant, and 2 keeps the time per line constant
- **Speed ratio** (float32) defines the backward tip speed related to the forward speed

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Scan.SpeedGet

Returns the scan speed parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Forward linear speed (m/s)** (float32)
- **Backward linear speed (m/s)** (float32)
- **Forward time per line (s)** (float32)
- **Backward time per line (s)** (float32)
- **Keep parameter constant** (unsigned int16) defines which speed parameter to keep constant, where 0 keeps the linear speed constant, and 1 keeps the time per line constant
- **Speed ratio** (float32) is the backward tip speed related to the forward speed
- **Error** described in the Response message>Body section

Scan.FrameDataGrab

Returns the scan data of the selected frame.

Arguments:

- **Channel index** (unsigned int32) selects which channel to get the data from.
  The channel must be one of the acquired channels.
  The list of acquired channels while scanning can be configured by the function *Scan.BufferSet.*
  The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.
- **Data direction** (unsigned int32) selects the data direction, where 1 is forward, and 0 is backward

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels name size** (int) is the size in bytes of the Channel name string
- **Channel name** (string) is the name of the channel selected by Channel index
- **Scan data rows** (int) defines the number of rows of the Scan data array
- **Scan data columns** (int) defines the number of columns of the Scan data array
- **Scan data** (2D array float32) returns the scan frame data of the selected channel
- **Scan direction** (unsigned int32) is the scan direction, where 1 is up, and 0 is down
- **Error** described in the Response message>Body section

Follow Me

FolMe.XYPosSet

Moves the tip.

This function moves the tip to the specified X and Y target coordinates (in meters). It moves at the speed specified by the "Speed" parameter in the Follow Me mode of the Scan Control module.

This function will return when the tip reaches its destination or if the movement stops.

Arguments:

- **X (m)** (float64) sets the target X position of the tip
- **Y (m)** (float64) sets the target Y position of the tip
- **Wait end of move** (unsigned int32) selects whether the function immediately (=0) or if it waits until the target is reached or the movement is stopped (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FolMe.XYPosGet

Returns the X,Y tip coordinates (oversampled during the Acquisition Period time, Tap).

Arguments:

- **Wait for newest data** (unsigned int32) selects whether the function returns the next available signal value or if it waits for a full period of new data.
  If 0, this function returns a value 0 to Tap seconds after being called.
  If 1, the function discards the first oversampled signal value received but returns the second value received. Thus, the function returns a value Tap to 2*Tap seconds after being called

Return arguments (if Send response back flag is set to True when sending request message):

- **X (m)** (float64) is the current X position of the tip
- **Y (m)** (float64) is the current Y position of the tip
- **Error** described in the Response message>Body section

FolMe.SpeedSet

Configures the tip speed when moving in Follow Me mode.

Arguments:

- **Speed (m/s)** (float32) sets the surface speed in Follow Me mode
- **Custom speed** (unsigned int32) sets whether custom speed setting is used for Follow Me mode (=1) or if scan speed is used (=0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FolMe.SpeedGet

Returns the tip speed when moving in Follow Me mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Speed (m/s)** (float32) is the surface speed in Follow Me mode
- **Custom speed** (unsigned int32) returns whether custom speed setting is used for Follow Me mode (=1) or if scan speed is used (=0)
- **Error** described in the Response message>Body section

FolMe.OversamplSet

Sets the oversampling of the acquired data when the tip is moving in Follow Me mode.

Arguments:

- **Oversampling** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FolMe.OversamplGet

Returns the oversampling and rate of the acquired data when the tip is moving in Follow Me mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Oversampling** (int)
- **Sampling rate (Samples/s)** (float32)
- **Error** described in the Response message>Body section

FolMe.Stop

Stops the tip movement in Follow Me mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FolMe.PSOnOffGet

Returns if Point & Shoot is enabled or disabled in Follow Me mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Point & Shoot status** (unsigned int32) returns whether Point & Shoot is enabled (=1) or disabled (=0)
- **Error** described in the Response message>Body section

FolMe.PSOnOffSet

Enables or disables Point & Shoot in Follow Me mode.

Arguments:

- **Point & Shoot status** (unsigned int32) enables (=1) or disables (=0) Point & Shoot

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FolMe.PSExpGet

Returns the Point & Shoot experiment selected in Follow Me mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Point & Shoot experiment** (unsigned int16) returns the selected Point & Shoot experiment
- **Size of the list of experiments** (int) is the full size in bytes of the List of experiments string array
- **Number of experiments** (int) is the number of elements of the List of experiments string array
- **List of experiments** (1D array string) returns the list of experiments available in the Pattern section. The size of each string item comes right before it as integer 32
- **Error** described in the Response message>Body section

FolMe.PSExpSet

Sets the Point & Shoot experiment selected in Follow Me mode.

Arguments:

- **Point & Shoot experiment** (unsigned int16) returns the selected Point & Shoot experiment

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FolMe.PSPropsGet

Returns the Point & Shoot configuration in Follow Me mode.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Auto resume** (unsigned int32) returns if the scan resumes after running the experiment (=1) or if it remains paused (=0)
- **Use own basename** (unsigned int32) returns if the file basename is the one defined in the experiment module (i.e. in Bias Spectroscopy) (=1) or if it uses the basename configured in Point & Shoot (=0)
- **Basename size** (int) is the size in bytes of the Basename string
- **Basename** (string) returns the basename defined in Point & Shoot
- **External VI path size** (int) is the size in bytes of the External VI path string
- **External VI path** (string) returns the path of the External VI selected in Point & Shoot
- **Pre-measure delay (s)** (float32) is the time to wait on each point before performing the experiment
- **Error** described in the Response message>Body section

FolMe.PSPropsSet

Sets the Point & Shoot configuration in Follow Me mode.

Arguments:

- **Auto resume** (unsigned int32) sets if the scan resumes after running the experiment (=1) or if it remains paused (=2). A value=0 means no change.
- **Use own basename** (unsigned int32) sets if the file basename is the one defined in the experiment module (i.e. in Bias Spectroscopy) (=1) or if it uses the basename configured in Point & Shoot (=2). A value=0 means no change.
- **Basename size** (int) is the size in bytes of the Basename string
- **Basename** (string) sets the basename in Point & Shoot
- **External VI path size** (int) is the size in bytes of the External VI path string
- **External VI path** (string) sets the path of the External VI selected in Point & Shoot
- **Pre-measure delay (s)** (float32) is the time to wait on each point before performing the experiment

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Tip Move Recorder

TipRec.BufferSizeSet

Sets the buffer size of the Tip Move Recorder. This function clears the graph.

Arguments:

- **Buffer size** (int) is the number of data elements in the Tip Move Recorder

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TipRec.BufferSizeGet

Returns the buffer size of the Tip Move Recorder.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Buffer size** (int) is the number of data elements in the Tip Move Recorder
- **Error** described in the Response message>Body section

TipRec.BufferClear

Clears the buffer of the Tip Move Recorder.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TipRec.DataGet

Returns the indexes and values of the channels acquired while the tip is moving in Follow Me mode (displayed in the Tip Move Recorder).

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.
- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the recorded data while moving the tip
- **Error** described in the Response message>Body section

TipRec.DataSave

Saves the data acquired in Follow Me mode (displayed in the Tip Move Recorder) to a file.

Arguments:

- **Clear buffer** (unsigned int32) clears the buffer after saving the data. 0 means Off, and 1 means On
- **Basename size** (int) is the number of bytes of the Basename string
- **Basename** (string) defines the basename of the file where the data are saved. If empty, the basename will be the one used in the last save operation

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern

Pattern.ExpOpen

Opens the selected grid experiment.

This is required to configure the experiment and be able to start it.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.ExpStart

Starts the selected grid experiment.

Before using this function, select the experiment through *Pattern.PropsSet*, and be sure to have it open in the software or through the function *Pattern.ExpOpen*. Otherwise it will give an error saying that the experiment has not been configured yet.

Arguments:

- **Pattern** (unsigned int16) switches the active pattern to this value before starting the grid experiment. 0 means no change, 1 means Grid, 2 means Line, and 3 means Cloud

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.ExpPause

Pauses or resumes the selected grid experiment.

Arguments:

- **Pause/Resume** (unsigned int32) where 1 means Pause and 0 means Resume

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.ExpStop

Stops the selected grid experiment.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.ExpStatusGet

Returns the status of the selected grid experiment.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) indicates if the experiment is running (=1) or not (=0)
- **Error** described in the Response message>Body section

Pattern.GridSet

Sets the grid size parameters.

Arguments:

- **Set active pattern** (unsigned int32) defines if the pattern switches to Grid, in case it was not Grid already. 0 means Off, and 1 means On
- **Number of points in X** (int) is the number of points in X that defines the grid
- **Number of points in Y** (int) is the number of points in Y that defines the grid
- **Grid=Scan frame** (unsigned int32) defines if the grid size should be set like the scan frame size. 0 means No, and 1 means Yes
- **Center X (m)** (float32) is the X coordinate of the center of the grid
- **Center Y (m)** (float32) is the Y coordinate of the center of the grid
- **Width (m)** (float32) is the width of the grid
- **Height (m)** (float32) is the height of the grid
- **Angle (deg)** (float32) is the rotation angle of the grid

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.GridGet

Returns the grid size parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of points in X** (int) is the number of points in X that defines the grid
- **Number of points in Y** (int) is the number of points in Y that defines the grid
- **Center X (m)** (float32) is the X coordinate of the center of the grid
- **Center Y (m)** (float32) is the Y coordinate of the center of the grid
- **Width (m)** (float32) is the width of the grid
- **Height (m)** (float32) is the height of the grid
- **Angle (deg)** (float32) is the rotation angle of the grid
- **Error** described in the Response message>Body section

Pattern.LineSet

Sets the line size parameters.

Arguments:

- **Set active pattern** (unsigned int32) defines if the pattern switches to Line, in case it was not Line already. 0 means Off, and 1 means On
- **Number of points** (int) is the number of points that defines the line
- **Line=Scan frame** (unsigned int32) defines if the line size should be set equal to the scan frame diagonal. 0 means No, and 1 means Yes
- **Line Point 1 X (m)** (float32) is the X coordinate of one of the two points that define the line
- **Line Point 1 Y (m)** (float32) is the Y coordinate of one of the two points that define the line
- **Line Point 2 X (m)** (float32) is the X coordinate of one of the two points that define the line
- **Line Point 2 Y (m)** (float32) is the Y coordinate of one of the two points that define the line

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.LineGet

Returns the line size parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of points** (int) is the number of points that defines the line
- **Line Point 1 X (m)** (float32) is the X coordinate of one of the two points that define the line
- **Line Point 1 Y (m)** (float32) is the Y coordinate of one of the two points that define the line
- **Line Point 2 X (m)** (float32) is the X coordinate of one of the two points that define the line
- **Line Point 2 Y (m)** (float32) is the Y coordinate of one of the two points that define the line
- **Error** described in the Response message>Body section

Pattern.CloudSet

Configures a cloud of points.

Arguments:

- **Set active pattern** (unsigned int32) defines if the pattern switches to Cloud, in case it was not Cloud already. 0 means Off, and 1 means On
- **Number of points** (int) is the number of points in the cloud, and it defines the size of the 1D arrays for X and Y coordinates
- **X coordinates (m)** (1D array float32) is a 1D array of the X coordinates of the points defining the cloud
- **Y coordinates (m)** (1D array float32) is a 1D array of the Y coordinates of the points defining the cloud

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.CloudGet

Returns the cloud configuration.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of points** (int) is the number of points in the cloud, and it defines the size of the 1D arrays for X and Y coordinates
- **X coordinates (m)** (1D array float32) is a 1D array of the X coordinates of the points defining the cloud
- **Y coordinates (m)** (1D array float32) is a 1D array of the Y coordinates of the points defining the cloud
- **Error** described in the Response message>Body section

Pattern.PropsSet

Sets the configuration of the Grid experiment section in the Scan module.

Arguments:

- **Selected experiment size** (int) is the number of bytes of the Selected experiment string
- **Selected experiment** (string) is the name of the selected experiment
- **Basename size** (int) is the number of bytes of the Basename string
- **Basename** (string) sets the basename of the saved files
- **External VI path size** (int) is the number of bytes of the External VI path string
- **External VI path** (string) sets the path of the External VI
- **Pre-measure delay (s)** (float32) is the time to wait on each point before performing the experiment
- **Save scan channels** (unsigned int32) sets if the scan channels are saved into the grid experiment file. 0 means Off, and 1 means On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Pattern.PropsGet

Gets the configuration of the Grid experiment section in the Scan module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Size of the list of experiments** (int) is the size in bytes of the List of experiments string array
- **Number of experiments** (int) is the number of elements of the List of experiments string array
- **List of experiments** (1D array string) returns the list of experiments available in the Pattern section. The size of each string item comes right before it as integer 32
- **Selected experiment size** (int) is the number of bytes of the Selected experiment string
- **Selected experiment** (string) returns the name of the selected experiment
- **External VI path size** (int) is the number of bytes of the External VI path string
- **External VI path** (string) returns the path of the External VI
- **Pre-measure delay (s)** (float32) is the time to wait on each point before performing the experiment
- **Save scan channels** (unsigned int32) indicates if the scan channels are saved into the grid experiment file. 0 means Off, and 1 means On
- **Error** described in the Response message>Body section

Marks in Scan

Marks.PointDraw

Draws text at the specified point of the scan frame.

This function can be very useful to mark an important location in the scan image (i.e. the position where the Tip Shaper executed).

Arguments:

- **X coordinate (m)** (float32) defines the X coordinate in meters of the center of the text
- **Y coordinate (m)** (float32) defines the Y coordinate in meters of the center of the text
- **Text size** (int) is the number of bytes corresponding to the Text to draw
- **Text** (string) sets the character/s to draw
- **Color** (unsigned int32) sets the RGB color of Text

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.PointsDraw

Draws text at the specified points of the scan frame.

This function can be very useful to mark important locations in the scan image (i.e. the position where the Tip Shaper executed).

Arguments:

- **Number of points** (int) indicates the number of points to draw. This value is also the size of the 1D arrays set afterwards
- **X coordinate (m)** (1D array float32) defines the X coordinates in meters of the center of the text for the points to draw
- **Y coordinate (m)** (1D array float32) defines the Y coordinates in meters of the center of the text for the points to draw
- **Text** (1D array string) sets the character/s to draw at each point.
  Each element of the array must be preceded by its size in bytes in Integer 32 (int) format
- **Color** (1D array unsigned int32) sets the RGB colors for the different points to draw

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.LineDraw

Draws a line in the scan frame.

Arguments:

- **Start point X coordinate (m)** (float32) defines the X coordinate in meters of the starting point of the line
- **Start point Y coordinate (m)** (float32) defines the Y coordinate in meters of the starting point of the line
- **End point X coordinate (m)** (float32) defines the X coordinate in meters of the end point of the line
- **End point Y coordinate (m)** (float32) defines the Y coordinate in meters of the end point of the line
- **Color** (unsigned int32) sets the RGB color of the line

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.LinesDraw

Draws multiple lines in the scan frame.

Arguments:

- **Number of lines** (int) indicates the number of lines to draw. This value is also the size of the 1D arrays set afterwards
- **Start point X coordinate (m)** (1D array float32) defines the X coordinate in meters of the starting point of each line
- **Start point Y coordinate (m)** (1D array float32) defines the Y coordinate in meters of the starting point of each line
- **End point X coordinate (m)** (1D array float32) defines the X coordinate in meters of the end point of each line
- **End point Y coordinate (m)** (1D array float32) defines the Y coordinate in meters of the end point of each line
- **Color** (1D array unsigned int32) sets the RGB color of each line

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.PointsErase

Erase the point specified by the index parameter from the scan frame.

Arguments:

- **Point index** (int) sets the index of the point to erase. The index is comprised between 0 and the total number of drawn points minus one. To see which point has which index, use the *Marks.PointsGet* function. Value -1 erases all points.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.LinesErase

Erase the line specified by the index parameter from the scan frame.

Arguments:

- **Line index** (int) sets the index of the line to erase. The index is comprised between 0 and the total number of drawn lines minus one. To see which line has which index, use the *Marks.LinesGet* function. Value -1 erases all lines.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.PointsVisibleSet

Show or hide the point specified by the index parameter.

Arguments:

- **Point index** (int) sets the index of the point to show or hide. The index is comprised between 0 and the total number of drawn points minus one. To see which point has which index, use the *Marks.PointsGet* function. Value -1 shows or hides all points.
- **Show/hide** (unsigned int16) defines if the point should be visible (=0) or invisible (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.LinesVisibleSet

Show or hide the line specified by the index parameter.

Arguments:

- **Line index** (int) sets the index of the line to show or hide. The index is comprised between 0 and the total number of drawn lines minus one. To see which line has which index, use the *Marks.LinesGet* function. Value -1 shows or hides all lines.
- **Show/hide** (unsigned int16) defines if the line should be visible (=0) or invisible (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Marks.PointsGet

Returns the information of the points drawn in the scan frame.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of points** (int) indicates the number of drawn points. This value is also the size of the 1D arrays returned afterwards
- **X coordinate (m)** (1D array float32) returns the X coordinates in meters of the center of the text for the drawn points
- **Y coordinate (m)** (1D array float32) returns the Y coordinates in meters of the center of the text for the drawn points
- **Text size** (int) is the number of bytes corresponding to the entire Text array
- **Text** (1D array string) returns the text drawn at each point.
  Each element of the array is preceded by its size in bytes in Integer 32 (int) format
- **Color** (1D array unsigned int32) returns the RGB colors for the different drawn points
- **Visible** (1D array unsigned int32) returns if each point is visible (=1) or invisible (=0) in the scan frame
- **Error** described in the Response message>Body section

Marks.LinesGet

Returns the information of the lines drawn in the scan frame.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of lines** (int) indicates the number of drawn lines. This value is also the size of the 1D arrays returned afterwards
- **Start point X coordinate (m)** (1D array float32) returns the X coordinate in meters of the starting point of each line
- **Start point Y coordinate (m)** (1D array float32) returns the Y coordinate in meters of the starting point of each line
- **End point X coordinate (m)** (1D array float32) returns the X coordinate in meters of the end point of each line
- **End point Y coordinate (m)** (1D array float32) returns the Y coordinate in meters of the end point of each line
- **Color** (1D array unsigned int32) returns the RGB colors for the different drawn lines
- **Visible** (1D array unsigned int32) returns if each line is visible (=1) or invisible (=0) in the scan frame
- **Error** described in the Response message>Body section

Multi-Pass

MPass.Activate

Activates Multi-Pass in the Scan Control module.

Arguments:

- **On/Off** (unsigned int32) defines if this function activates (1=On) Multi-Pass in the Scan Control module.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

MPass.Load

Loads a Multi-Pass configuration file (.mpas) in the Multi-Pass Configuration module.

Arguments:

- **File path size** (int) is the number of characters of the File path string
- **File path** (string) is the path of the .mpas file to load. When leaving it empty, Multi-Pass loads the configuration saved in the Session settings file, if there is any.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

MPass.Save

Saves the current configuration in the Multi-Pass Configuration module into a Multi-Pass configuration file (.mpas).

Arguments:

- **File path size** (int) is the number of characters of the File path string
- **File path** (string) is the path of the .mpas file to save. When leaving it empty, Multi-Pass saves the configuration into the Session settings file.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Tip Shaper

TipShaper.Start

Starts the tip shaper procedure.

Arguments:

- **Wait until finished** (unsigned int32) defines if this function waits (1=True) until the Tip Shaper procedure stops.
- **Timeout (ms)** (int) sets the number of milliseconds to wait if Wait until Finished is set to True.
  A value equal to -1 means waiting forever.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TipShaper.PropsSet

Sets the configuration of the tip shaper procedure.

Arguments:

- **Switch Off Delay** (float32) is the time during which the Z position is averaged right before switching the Z-Controller off.
- **Change Bias?** (unsigned int32) decides whether the Bias value is applied (0=no change, 1=True, 2=False) right before the first Z ramping.
- **Bias (V)** (float32) is the value applied to the Bias signal if Change Bias? is True.
- **Tip Lift (m)** (float32) defines the relative height the tip is going to ramp for the first time (from the current Z position).
- **Lift Time 1 (s)** (float32) defines the time to ramp Z from the current Z position by the Tip Lift amount.
- **Bias Lift (V)** (float32) is the Bias voltage applied just after the first Z ramping.
- **Bias Settling Time (s)** (float32) is the time to wait after applying the Bias Lift value, and it is also the time to wait after applying Bias (V) before ramping Z for the first time.
- **Lift Height (m)** (float32) defines the height the tip is going to ramp for the second time.
- **Lift Time 2 (s)** (float32) is the given time to ramp Z in the second ramping.
- **End Wait Time (s)** (float32) is the time to wait after restoring the initial Bias voltage (just after finishing the second ramping).
- **Restore Feedback?** (unsigned int32) defines whether the initial Z-Controller status is restored (0=no change, 1=True, 2=False) at the end of the tip shaper procedure.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TipShaper.PropsGet

Returns the configuration of the tip shaper procedure.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Switch Off Delay** (float32) is the time during which the Z position is averaged right before switching the Z-Controller off.
- **Change Bias?** (unsigned int32) returns whether the Bias value is applied (0=False, 1=True) right before the first Z ramping.
- **Bias (V)** (float32) is the value applied to the Bias signal if Change Bias? is True.
- **Tip Lift (m)** (float32) returns the relative height the tip is going to ramp for the first time (from the current Z position).
- **Lift Time 1 (s)** (float32) returns the time to ramp Z from the current Z position by the Tip Lift amount.
- **Bias Lift (V)** (float32) is the Bias voltage applied just after the first Z ramping.
- **Bias Settling Time (s)** (float32) is the time to wait after applying the Bias Lift value, and it is also the time to wait after applying Bias (V) before ramping Z for the first time.
- **Lift Height (m)** (float32) returns the height the tip is going to ramp for the second time.
- **Lift Time 2 (s)** (float32) is the given time to ramp Z in the second ramping.
- **End Wait Time (s)** (float32) is the time to wait after restoring the initial Bias voltage (just after finishing the second ramping).
- **Restore Feedback?** (unsigned int32) returns whether the initial Z-Controller status is restored (0=False, 1=True) at the end of the tip shaper procedure.
- **Error** described in the Response message>Body section

Coarse Motion

Motor.StartMove

Moves the coarse positioning device (motor, piezo actuator…).

Arguments:

- **Direction** (unsigned int32) selects in which direction to move. Note that depending on your motor controller and setup only the Z axis or even only Z- may work.
  Valid values are 0=X+, 1=X-, 2=Y+, 3=Y-, 4=Z+, 5=Z-
- **Number of steps** (unsigned int16) defines the number of steps to move in the specified direction
- **Group** (unsigned int32) is the selection of the groups defined in the motor control module. If the motor doesn't support the selection of groups, set it to 0.
  Valid values are 0=Group 1, 1=Group 2, 2=Group 3, 3=Group 4, 4=Group 5, 5=Group 6
- **Wait until finished** (unsigned int32) defines if this function only returns (1=True) when the motor reaches its destination or the movement stops

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Motor.StartClosedLoop

Moves the coarse positioning device (motor, piezo actuator…) in closed loop. This is not supported by all motor control modules.

Arguments:

- **Absolute/relative** (unsigned int32) selects if moving in relative (0=rel) or in absolute (1=abs) movement
- **Target X(m)** (float64) is the X target position to move in meters
- **Target Y(m)** (float64) is the Y target position to move in meters
- **Target Z(m)** (float64) is the Z target position to move in meters
- **Wait until finished** (unsigned int32) defines if this function only returns (1=True) when the motor reaches its destination or the movement stops

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Motor.StopMove

Stops the motor motion.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Motor.PosGet

Returns the positions of the motor control module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **X(m)** (float64) is the X position in meters
- **Y(m)** (float64) is the Y position in meters
- **Z(m)** (float64) is the Z position in meters
- **Error** described in the Response message>Body section

Motor.StepCounterGet

Returns the step counter values of X, Y, and Z.

This function also allows to reset the step counters after reading their values through the inputs Reset X, Reset Y, and Reset Z.

Currently this function is only available in Attocube ANC150 devices.

Arguments:

- **Reset X** (unsigned int32) resets the Step Counter X after reading its value. 0 means False, and 1 means True
- **Reset Y** (unsigned int32) resets the Step Counter Y after reading its value. 0 means False, and 1 means True
- **Reset Z** (unsigned int32) resets the Step Counter Z after reading its value. 0 means False, and 1 means True

Return arguments (if Send response back flag is set to True when sending request message):

- **Step counter X** (int)
- **Step counter Y** (int)
- **Step counter Z** (int)
- **Error** described in the Response message>Body section

Motor.FreqAmpGet

Returns the frequency (Hz) and amplitude (V) of the motor control module.

This function is only available for PD5, PMD4, and Attocube ANC150 devices.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Frequency (Hz)** (float32)
- **Amplitude (V)** (float32)
- **Error** described in the Response message>Body section

Motor.FreqAmpSet

Sets the frequency (Hz) and amplitude (V) of the motor control module.

This function is only available for PD5, PMD4, and Attocube ANC150 devices.

Arguments:

- **Frequency (Hz)** (float32)
- **Amplitude (V)** (float32)
- **Axis** (unsigned int16) defines which axis these parameters will be applied to. 0 means All, 1 means X, 2 means Y, 3 means Z

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Generic Sweeper

GenSwp.AcqChsSet

Sets the list of recorded channels of the Generic Sweeper.

Arguments:

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The indexes correspond to the list of Measurement in the Nanonis software.
  To get the Measurements names use the *Signals.MeasNamesGet* function

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenSwp.AcqChsGet

Returns the list of recorded channels of the Generic Sweeper.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of the recorded channels. The indexes correspond to the list of Measurement in the Nanonis software.
  To get the Measurements names use the *Signals.MeasNamesGet* function
- **Error** described in the Response message>Body section

GenSwp.SwpSignalSet

Sets the Sweep signal in the Generic Sweeper.

Arguments:

- **Sweep channel name size** (int) is the number of characters of the sweep channel name string
- **Sweep channel name** (string) is the name of the signal selected for the sweep channel

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenSwp.SwpSignalGet

Returns the selected Sweep signal in the Generic Sweeper.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Sweep channel name size** (int) is the number of characters of the sweep channel name string
- **Sweep channel name** (string) is the name of the signal selected for the sweep channel
- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Error** described in the Response message>Body section

GenSwp.LimitsSet

Sets the limits of the Sweep signal in the Generic Sweeper.

Arguments:

- **Lower limit** (float32) defines the lower limit of the sweep range
- **Upper limit** (float32) defines the upper limit of the sweep range

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenSwp.LimitsGet

Returns the limits of the Sweep signal in the Generic Sweeper.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Lower limit** (float32) defines the lower limit of the sweep range
- **Upper limit** (float32) defines the upper limit of the sweep range
- **Error** described in the Response message>Body section

GenSwp.PropsSet

Sets the configuration of the parameters in the Generic Sweeper.

Arguments:

- **Initial Settling time (ms)** (float32)
- **Maximum slew rate (units/s)** (float32)
- **Number of steps** (int) defines the number of steps of the sweep. 0 points means no change
- **Period (ms)** (unsigned int16) where 0 means no change
- **Autosave** (int) defines if the sweep is automatically saved, where -1=no change, 0=Off, 1=On
- **Save dialog box** (int) defines if the save dialog box shows up or not, where -1=no change, 0=Off, 1=On
- **Settling time (ms)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenSwp.PropsGet

Returns the configuration of the parameters in the Generic Sweeper.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Initial Settling time (ms)** (float32)
- **Maximum slew rate (units/s)** (float32)
- **Number of steps** (int) defines the number of steps of the sweep
- **Period (ms)** (unsigned int16)
- **Autosave** (unsigned int32) defines if the sweep is automatically saved, where 0=Off, 1=On
- **Save dialog box** (unsigned int32) defines if the save dialog box shows up or not, where 0=Off, 1=On
- **Settling time (ms)** (float32)
- **Error** described in the Response message>Body section

GenSwp.Start

Starts the sweep in the Generic Sweeper.

Arguments:

- **Get data** (unsigned int32) defines if the function returns the sweep data (1=True) or not (0=False)
- **Sweep direction** (unsigned int32) defines if the sweep starts from the lower limit (=1) or from the upper limit (=0)
- **Save base name string size** (int) defines the number of characters of the Save base name string
- **Save base name** (string) is the basename used by the saved files. If empty string, there is no change
- **Reset signal** (unsigned int32) where 0=Off, 1=On

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the numer of rows of the Data array
- **Data columns** (int) defines the numer of columns of the Data array
- **Data** (2D array float32) returns the sweep data
- **Error** described in the Response message>Body section

GenSwp.Stop

Stops the sweep in the Generic Sweeper module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenSwp.Open

Opens the Generic Sweeper module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Generic PI Controller

GenPICtrl.OnOffSet

Switches the Generic PI Controller On or Off.

Arguments:

- **Controller status** (unsigned int32) switches the controller Off (=0) or On (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenPICtrl.OnOffGet

Returns the status of the Generic PI Controller..

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Controller status** (unsigned int32) indicates if the controller is Off (=0) or On (=1)
- **Error** described in the Response message>Body section

GenPICtrl.AOValSet

Sets the output signal value of the User Output controlled by the Generic PI controller.

Arguments:

- **Output value** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenPICtrl.AOValGet

Gets the output signal value of the User Output controlled by the Generic PI controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Output value** (float32)
- **Error** described in the Response message>Body section

GenPICtrl.AOPropsSet

Sets the properties of the User Output controlled by the Generic PI controller.

Arguments:

- **Signal name size** (int) is the number of characters of the Signal name string
- **Signal name** (string) is the name of the selected output
- **Units size** (int) is the number of characters of the Units string
- **Units** (string) sets the physical units of the selected output
- **Upper limit** (float32) defines the upper physical limit of the user output
- **Lower limit** (float32) defines the lower physical limit of the user output
- **Calibration per volt** (float32)
- **Offset in physical units** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenPICtrl.AOPropsGet

Gets the properties of the User Output controlled by the Generic PI controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Signal name size** (int) is the number of characters of the Signal name string
- **Signal name** (string) is the name of the selected output
- **Units size** (int) is the number of characters of the Units string
- **Units** (string) returns the physical units of the selected output
- **Upper limit** (float32) returns the upper physical limit of the user output
- **Lower limit** (float32) returns the lower physical limit of the user output
- **Calibration per volt** (float32)
- **Offset in physical units** (float32)
- **Error** described in the Response message>Body section

GenPICtrl.ModChSet

Sets the index of the User Output controlled by the Generic PI controller.

Arguments:

- **Output index** (int) sets the output index to be used, which could be any value from 1 to the number of available outputs

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenPICtrl.ModChGet

Gets the index of the User Output controlled by the Generic PI controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Output index** (int) returns the output index to be used, which could be any value from 1 to the number of available outputs. 0 means that no output is selected
- **Error** described in the Response message>Body section

GenPICtrl.DemodChSet

Sets the index of the signal demodulated by the Generic PI controller.

Arguments:

- **Input index** (int) is comprised between 0 and 127 for the physical inputs, physical outputs, and internal channels. To see which signal has which index, see *Signals.NamesGet* function. Value -1 means no change
- **AC mode** (unsigned int16) sets the AC parameter. 0 means no change, 1 means AC is On, and 2 means AC is Off

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenPICtrl.DemodChGet

Gets the index of the signal demodulated by the Generic PI controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Input index** (int) is comprised between 0 and 127 for the physical inputs, physical outputs, and internal channels. To see which signal has which index, see *Signals.NamesGet* function
- **Error** described in the Response message>Body section

GenPICtrl.PropsSet

Gets the properties of the Generic PI controller.

Arguments:

- **Setpoint** (float32)
- **P gain** (float32)
- **Time constant** (float32)
- **Slope** (unsigned int16) where 0 means no change, 1 means Positive, and 2 means Negative

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

GenPICtrl.PropsGet

Gets the properties of the Generic PI controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Setpoint** (float32)
- **P gain** (float32)
- **Time constant** (float32)
- **Slope** (unsigned int16) where 0 means Negative and 1 means Positive
- **Error** described in the Response message>Body section

Atom Tracking

AtomTrack.CtrlSet

Turns the selected Atom Tracking control (modulation, controller or drift measurement) On or Off.

Arguments:

- **AT control** (unsigned int16) sets which control to switch. 0 means Modulation, 1 means Controller, and 2 means Drift Measurement
- **Status** (unsigned int16) switches the selected control Off (=0) or On (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

AtomTrack.StatusGet

Returns the status of the selected Atom Tracking control (modulation, controller or drift measurement).

Arguments:

- **AT control** (unsigned int16) sets which control to read the status from. 0 means Modulation, 1 means Controller, and 2 means Drift Measurement

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int16) returns the status of the selected control, where 0 means Off and 1 means On
- **Error** described in the Response message>Body section

AtomTrack.PropsSet

Sets the Atom Tracking parameters.

Arguments:

- **Integral gain** (float32) is the gain of the Atom Tracking controller
- **Frequency (Hz)** (float32) is the frequency of the modulation
- **Amplitude (m)** (float32) is the amplitude of the modulation
- **Phase (deg)** (float32) is the phase of the modulation
- **Switch Off delay (s)** (float32) means that before turning off the controller, the position is averaged over this time delay. The averaged position is then applied. This leads to reproducible positions when switching off the Atom Tracking controller

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

AtomTrack.PropsGet

Returns the Atom Tracking parameters.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Integral gain** (float32) is the gain of the Atom Tracking controller
- **Frequency (Hz)** (float32) is the frequency of the modulation
- **Amplitude (m)** (float32) is the amplitude of the modulation
- **Phase (deg)** (float32) is the phase of the modulation
- **Switch Off delay (s)** (float32) means that before turning off the controller, the position is averaged over this time delay. The averaged position is then applied. This leads to reproducible positions when switching off the Atom Tracking controller
- **Error** described in the Response message>Body section

AtomTrack.QuickCompStart

Starts the Tilt or Drift compensation.

Arguments:

- **AT control** (unsigned int16) sets if Tilt (=0) or Drift (=1) compensations starts

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

AtomTrack.DriftComp

Applies the Drift measurement to the Drift compensation.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Lock-In

LockIn.ModOnOffSet

Turns the specified Lock-In modulator on or off.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)
- **Lock-In On/Off** (unsigned int32) turns the specified modulator on or off, where 0=Off and 1=On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModOnOffGet

Returns if the specified Lock-In modulator is turned on or off.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Lock-In On/Off** (unsigned int32) returns if the specified modulator is turned on or off, where 0=Off and 1=On
- **Error** described in the Response message>Body section

LockIn.ModSignalSet

Selects the modulated signal of the specified Lock-In modulator.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)
- **Modulator Signal Index** (int) is the signal index out of the list of 128 signals available in the software. To get a list of the available signals, use the *Signals.NamesGet* function.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModSignalGet

Returns the modulated signal of the specified Lock-In modulator.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Modulator Signal Index** (int) is the signal index out of the list of 128 signals available in the software. To get a list of the available signals, use the *Signals.NamesGet* function
- **Error** described in the Response message>Body section

LockIn.ModPhasRegSet

Sets the phase register index of the specified Lock-In modulator.

Each modulator can work on any phase register (frequency). Use this function to assign the modulator to one of the 8 available phase registers (index 1-8).

Use the *LockIn.ModPhaFreqSet* function to set the frequency of the phase registers.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)
- **Phase Register Index** (int) is the index of the phase register of the specified Lock-In modulator. Valid values are index 1 to 8.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModPhasRegGet

Returns the phase register index of the specified Lock-In modulator.

Each modulator can work on any phase register (frequency generator).

Use the *LockIn.ModPhaseRegFreqGet* function to get the frequency of the phase registers.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Phase Register Index** (int) is the index of the phase register of the specified Lock-In modulator. Valid values are index 1 to 8
- **Error** described in the Response message>Body section

LockIn.ModHarmonicSet

Sets the harmonic of the specified Lock-In modulator.

The modulator is bound to a phase register (frequency generator), but it can work on harmonics. Harmonic 1 is the base frequency (the frequency of the frequency generator).

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)
- **Harmonic** (int) is the harmonic of the specified Lock-In modulator. Valid values start from 1 (=base frequency)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModHarmonicGet

Returns the harmonic of the specified Lock-In modulator.

The modulator is bound to a phase register (frequency generator), but it can work on harmonics. Harmonic 1 is the base frequency (the frequency of the frequency generator).

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Harmonic** (int) is the harmonic of the specified Lock-In modulator. Valid values start from 1 (=base frequency)
- **Error** described in the Response message>Body section

LockIn.ModPhasSet

Sets the modulation phase offset of the specified Lock-In modulator.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)
- **Phase (deg)** (float32) is the modulation phase offset of the specified Lock-In modulator

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModPhasGet

Returns the modulation phase offset of the specified Lock-In modulator.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Phase (deg)** (float32) is the modulation phase offset of the specified Lock-In modulator
- **Error** described in the Response message>Body section

LockIn.ModAmpSet

Sets the modulation amplitude of the specified Lock-In modulator.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)
- **Amplitude** (float32) is the modulation amplitude of the specified Lock-In modulator

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModAmpGet

Returns the modulation amplitude of the specified Lock-In modulator.

Arguments:

- **Modulator number** (int) is the number that specifies which modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Amplitude** (float32) is the modulation amplitude of the specified Lock-In modulator
- **Error** described in the Response message>Body section

LockIn.ModPhasFreqSet

Sets the frequency of the specified Lock-In phase register/modulator.

The Lock-in module has a total of 8 frequency generators / phase registers. Each modulator and demodulator can be bound to one of the phase registers.

This function sets the frequency of one of the phase registers.

Arguments:

- **Modulator number** (int) is the number that specifies which phase register/modulator to use. It starts from number 1 (=Modulator 1)
- **Frequency (Hz)** (float64) is the frequency of the specified Lock-In phase register

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.ModPhasFreqGet

Returns the frequency of the specified Lock-In phase register/modulator.

The Lock-in module has a total of 8 frequency generators / phase registers. Each modulator and demodulator can be bound to one of the phase registers.

This function gets the frequency of one of the phase registers.

Arguments:

- **Modulator number** (int) is the number that specifies which phase register/modulator to use. It starts from number 1 (=Modulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Frequency (Hz)** (float64) is the frequency of the specified Lock-In phase register
- **Error** described in the Response message>Body section

LockIn.DemodSignalSet

Selects the demodulated signal of the specified Lock-In demodulator.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **Demodulator Signal Index** (int) is the signal index out of the list of 128 signals available in the software. To get a list of the available signals, use the *Signals.NamesGet* function.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodSignalGet

Returns the demodulated signal of the specified Lock-In demodulator.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Demodulator Signal Index** (int) is the signal index out of the list of 128 signals available in the software. To get a list of the available signals, use the *Signals.NamesGet* function

- **Error** described in the Response message>Body section

LockIn.DemodHarmonicSet

Sets the harmonic of the specified Lock-In demodulator.

The demodulator demodulates the input signal at the specified harmonic overtone of the frequency generator. Harmonic 1 is the base frequency (the frequency of the frequency generator).

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **Harmonic** (int) is the harmonic of the specified Lock-In demodulator. Valid values start from 1 (=base frequency)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodHarmonicGet

Returns the harmonic of the specified Lock-In demodulator.

The demodulator demodulates the input signal at the specified harmonic overtone of the frequency generator. Harmonic 1 is the base frequency (the frequency of the frequency generator).

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Harmonic** (int) is the harmonic of the specified Lock-In demodulator. Valid values start from 1 (=base frequency)
- **Error** described in the Response message>Body section

LockIn.DemodHPFilterSet

Sets the properties of the high-pass filter applied to the demodulated signal of the specified demodulator.

The high-pass filter is applied on the demodulated signal before the actual demodulation. It is used to get rid of DC or low-frequency components which could result in undesired components close to the modulation frequency on the demodulator output signals (X,Y).

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **HP Filter Order** (int) is the high-pass filter order. Valid values are from -1 to 8, where -1=no change, 0=filter off.
- **HP Filter Cutoff Frequency (Hz)** (float32) is the high-pass filter cutoff frequency in Hz, where 0 = no change.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodHPFilterGet

Returns the properties of the high-pass filter applied to the demodulated signal of the specified demodulator.

The high-pass filter is applied on the demodulated signal before the actual demodulation. It is used to get rid of DC or low-frequency components which could result in undesired components close to the modulation frequency on the demodulator output signals (X,Y).

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **HP Filter Order** (int) is the high-pass filter order. Valid values are from 0 to 8, where 0=filter off
- **HP Filter Cutoff Frequency (Hz)** (float32) is the high-pass filter cutoff frequency in Hz
- **Error** described in the Response message>Body section

LockIn.DemodLPFilterSet

Sets the properties of the low-pass filter applied to the demodulated signal of the specified demodulator.

The low-pass filter is applied on the demodulator output signals (X,Y) to remove undesired components. Lower cut-off frequency means better suppression of undesired frequency components, but longer response time (time constant) of the filter.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **LP Filter Order** (int) is the low-pass filter order. Valid values are from -1 to 8, where -1=no change, 0=filter off.
- **LP Filter Cutoff Frequency (Hz)** (float32) is the low-pass filter cutoff frequency in Hz, where 0 = no change.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodLPFilterGet

Returns the properties of the low-pass filter applied to the demodulated signal of the specified demodulator.

The low-pass filter is applied on the demodulator output signals (X,Y) to remove undesired components. Lower cut-off frequency means better suppression of undesired frequency components, but longer response time (time constant) of the filter.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **LP Filter Order** (int) is the low-pass filter order. Valid values are from -1 to 8, where -1=no change, 0=filter off.
- **LP Filter Cutoff Frequency (Hz)** (float32) is the low-pass filter cutoff frequency in Hz, where 0 = no change.
- **Error** described in the Response message>Body section

LockIn.DemodPhasRegSet

Sets the phase register index of the specified Lock-In demodulator.

Each demodulator can work on any phase register (frequency). Use this function to assign the demodulator to one of the 8 available phase registers (index 1-8).

Use the *LockIn.ModPhaFreqSet* function to set the frequency of the phase registers.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **Phase Register Index** (int) is the index of the phase register of the specified Lock-In demodulator. Valid values are index 1 to 8.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodPhasRegGet

Returns the phase register index of the specified Lock-In demodulator.

Each demodulator can work on any phase register (frequency). Use the *LockIn.ModPhaFreqSet* function to set the frequency of the phase registers.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Phase Register Index** (int) is the index of the phase register of the specified Lock-In demodulator. Valid values are index 1 to 8.
- **Error** described in the Response message>Body section

LockIn.DemodPhasSet

Sets the reference phase of the specified Lock-In demodulator.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **Phase (deg)** (float32) is the reference phase of the specified Lock-In demodulator

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodPhasGet

Returns the reference phase of the specified Lock-In demodulator.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Phase (deg)** (float32) is the reference phase of the specified Lock-In demodulator
- **Error** described in the Response message>Body section

LockIn.DemodSyncFilterSet

Switches the synchronous (Sync) filter of the specified demodulator On or Off.

The synchronous filter is applied on the demodulator output signals (X,Y) after the low-pass filter. It is very good in suppressing harmonic components (harmonics of the demodulation frequency), but does not suppress other frequencies.

The sync filter does not output a continuous signal, it only updates the value after each period of the demodulation frequency.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **Sync Filter** (unsigned int32) switches the synchronous filter of the specified demodulator on or off, where 0=Off and 1=On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodSyncFilterGet

Returns the status (on/off) of the synchronous (Sync) filter of the specified demodulator.

The synchronous filter is applied on the demodulator output signals (X,Y) after the low-pass filter. It is very good in suppressing harmonic components (harmonics of the demodulation frequency), but does not suppress other frequencies.

The sync filter does not output a continuous signal, it only updates the value after each period of the demodulation frequency.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Sync Filter** (unsigned int32) is the synchronous filter of the specified demodulator, where 0=Off and 1=On
- **Error** described in the Response message>Body section

LockIn.DemodRTSignalsSet

Sets the signals available for acquisition on the real-time system from the specified demodulator.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)
- **RT Signals** (unsigned int32) sets which signals from the specified demodulator should be available on the Real-time system. 0 sets the available RT Signals to X/Y, 1 sets them to R/phi.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockIn.DemodRTSignalsGet

Returns which the signals are available for acquisition on the real-time system from the specified demodulator.

Arguments:

- **Demodulator number** (int) is the number that specifies which demodulator to use. It starts from number 1 (=Demodulator 1)

Return arguments (if Send response back flag is set to True when sending request message):

- **RT Signals** (unsigned int32) returns which signals from the specified demodulator are available on the Real-time system. 0 means X/Y, and 1 means R/phi.

- **Error** described in the Response message>Body section

Lock-In Frequency Sweep

LockInFreqSwp.Open

Opens the Transfer function (Lock-In Frequency Sweep) module.

The transfer function does not run when its front panel is closed. To automate measurements it might be required to open the module first using this VI.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockInFreqSwp.Start

Starts a Lock-In frequency sweep.

Arguments:

- **Get Data** (unsigned int32) defines if the function returns the recorder channels and data
- **Direction** (unsigned int32) sets the direction of the frequency sweep. 0 means sweep down (from upper limit to lower limit) and 1 means sweep up (from lower limit to upper limit)

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the recorder channels names array
- **Channels names number** (int) is the number of elements of the recorded channels names array
- **Channels names** (1D array string) returns the array of recorded channel names (strings), where each string comes prepended by its size in bytes
- **Data rows** (int) is the number of rows of the returned data array (the first row is the swept frequency, and each additional row contains the data of each recorded channel)
- **Data columns** (int) is the number of recorded points (number of steps plus 1)
- **Data** (2D array float32) returns the recorded data. The number of rows is defined by *Data rows*, and the number of columns is defined by *Data columns*
- **Error** described in the Response message>Body section

LockInFreqSwp.SignalSet

Sets the sweep signal used in the Lock-In frequency sweep module.

Arguments:

- **Sweep signal index** (int) sets the sweep signal index out of the list of sweep signals to use, where -1 means no signal selected

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockInFreqSwp.SignalGet

Returns the sweep signal used in the Lock-In frequency sweep module.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Sweep signal index** (int) is the sweep signal index selected out of the list of sweep signals, where -1 means no signal selected
- **Error** described in the Response message>Body section

LockInFreqSwp.LimitsSet

Sets the frequency limits in the Lock-In frequency sweep module.

Arguments:

- **Lower limit (Hz)** (float32) sets the lower frequency limit in Hz
- **Upper limit (Hz)** (float32) sets the lower frequency limit in Hz

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockInFreqSwp.LimitsGet

Returns the frequency limits in the Lock-In frequency sweep module.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Lower limit (Hz)** (float32) is the lower frequency limit in Hz
- **Upper limit (Hz)** (float32) is the lower frequency limit in Hz
- **Error** described in the Response message>Body section

LockInFreqSwp.PropsSet

Sets the configuration of the Transfer Function (Lock-In frequency sweep) module.

Arguments:

- **Number of steps** (unsigned int16) is the number of frequency steps over the sweep range (logarithmic distribution). The number of data points = number of steps + 1. If set to 0, the number of steps is left unchanged
- **Integration periods** (unsigned int16) is the number of Lock in periods to average for one measurement.
- **Minimum integration time (s)** (float32) is the minimum integration time in seconds to average each measurement
- **Settling periods** (unsigned int16) is the number of Lock in periods to wait before acquiring data at each point of the sweep
- **Minimum Settling time (s)** (float32) is the minimum settling time in seconds to wait before acquiring data at each point of the sweep
- **Autosave** (unsigned int32) automatically saves the data at end of sweep
- **Save dialog** (unsigned int32) will open a dialog box when saving the data
- **Basename size** (int) is the size (number of characters) of the basename string
- **Basename** (string) is the basename of the saved files

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

LockInFreqSwp.PropsGet

Returns the configuration of the Transfer Function (Lock-In frequency sweep) module.

Arguments:

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of steps** (unsigned int16) is the number of frequency steps over the sweep range (logarithmic distribution). The number of data points = number of steps + 1
- **Integration periods** (unsigned int16) is the number of Lock in periods to average for one measurement.
- **Minimum integration time (s)** (float32) is the minimum integration time in seconds to average each measurement
- **Settling periods** (unsigned int16) is the number of Lock in periods to wait before acquiring data at each point of the sweep
- **Minimum Settling time (s)** (float32) is the minimum settling time in seconds to wait before acquiring data at each point of the sweep
- **Autosave** (unsigned int32) automatically saves the data at end of sweep
- **Save dialog** (unsigned int32) will open a dialog box when saving the data
- **Basename size** (int) is the size (number of characters) of the basename string
- **Basename** (string) is the basename of the saved files
- **Error** described in the Response message>Body section

PLL modules

PLL.InpCalibrSet

Sets the input calibration of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Calibration (m/V)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.InpCalibrGet

Returns the input calibration of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration (m/V)** (float32)
- **Error** described in the Response message>Body section

PLL.InpRangeSet

Sets the input range of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Input range (m)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.InpRangeGet

Returns the input range of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Input range (m)** (float32)
- **Error** described in the Response message>Body section

PLL.InpPropsSet

Sets the input parameters of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Differential input** (unsigned int16) where 0 is Off and 1 is On
- **1/10 divider** (unsigned int16) where 0 is Off and 1 is On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.InpPropsGet

Returns the input parameters of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Differential input** (unsigned int16) where 0 is Off and 1 is On
- **1/10 divider** (unsigned int16) where 0 is Off and 1 is On
- **Error** described in the Response message>Body section

PLL.AddOnOffSet

Sets the status of the Add external signal to the output.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Add** (unsigned int32) where 0 is Off and 1 is On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.AddOnOffGet

Returns the status of the Add external signal to the output.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Add** (unsigned int32) where 0 is Off and 1 is On
- **Error** described in the Response message>Body section

PLL.OutOnOffSet

Sets the status of the PLL output.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **PLL output** (unsigned int32) where 0 is Off and 1 is On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.OutOnOffGet

Returns the status of the PLL output.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **PLL output** (unsigned int32) where 0 is Off and 1 is On
- **Error** described in the Response message>Body section

PLL.ExcRangeSet

Sets the excitation range of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Output range** (unsigned int16) where 0 is 10V, 1 is 1V, 2 is 0.1V, 3 is 0.01V, and 4 is 0.001V

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.ExcRangeGet

Returns the excitation range of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Output range** (unsigned int16) where 0 is 10V, 1 is 1V, 2 is 0.1V, 3 is 0.01V, and 4 is 0.001V
- **Error** described in the Response message>Body section

PLL.ExcitationSet

Sets the current excitation value (i.e. the drive amplitude) of the oscillation control module.

This functions works only if the amplitude controller is switched Off.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Excitation value (V)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.ExcitationGet

Returns the current excitation value (i.e. the drive amplitude) of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Excitation value (V)** (float32)
- **Error** described in the Response message>Body section

PLL.AmpCtrlSetpntSet

Sets the amplitude controller setpoint.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Setpoint (m)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.AmpCtrlSetpntGet

Returns the amplitude controller setpoint.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Setpoint (m)** (float32)
- **Error** described in the Response message>Body section

PLL.AmpCtrlOnOffSet

Switches the amplitude controller On or Off.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Status** (unsigned int32) where 0 is Off and 1 is On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.AmpCtrlOnOffGet

Returns the status of the amplitude controller.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) where 0 is Off and 1 is On
- **Error** described in the Response message>Body section

PLL.AmpCtrlGainSet

Sets the amplitude controller gains and timing parameters.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **P gain (V/m)** (float32)
- **Time constant (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.AmpCtrlGainGet

Returns the amplitude controller gains and timing parameters.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **P gain (V/m)** (float32)
- **Time constant (s)** (float32)
- **Integral gain (V/m/s)** (float32)
- **Error** described in the Response message>Body section

PLL.AmpCtrlBandwidthSet

Sets the amplitude controller bandwidth of the oscillation control module.

This function uses the current Q factor and the amplitude to excitation ratio. These parameters can be identified through the Frequency Sweep module and they should be previously applied in the PLL Setup tool in order this function to get correctly the bandwidth.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Bandwidth (Hz)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.AmpCtrlBandwidthGet

Returns the amplitude controller bandwidth of the oscillation control module.

This function uses the current Q factor and the amplitude to excitation ratio. These parameters can be identified through the Frequency Sweep module and they should be previously applied in the PLL Setup tool in order this function to get correctly the bandwidth.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Bandwidth (Hz)** (float32)
- **Error** described in the Response message>Body section

PLL.PhasCtrlOnOffSet

Switches the phase controller On or Off.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Status** (unsigned int32) where 0 is Off and 1 is On

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.PhasCtrlOnOffGet

Returns the status of the phase controller.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) where 0 is Off and 1 is On
- **Error** described in the Response message>Body section

PLL.PhasCtrlGainSet

Sets the phase controller gains and timing parameters.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **P gain (Hz/deg)** (float32)
- **Time constant (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.PhasCtrlGainGet

Returns the phase controller gains and timing parameters.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **P gain (Hz/deg)** (float32)
- **Time constant (s)** (float32)
- **Integral gain (Hz/deg/s)** (float32)
- **Error** described in the Response message>Body section

PLL.PhasCtrlBandwidthSet

Sets the phase controller bandwidth of the oscillation control module.

This function uses the current Q factor. This parameter can be identified through the Frequency Sweep module and it should be previously applied in the PLL Setup tool in order this function to get correctly the bandwidth.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Bandwidth (Hz)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.PhasCtrlBandwidthGet

Returns the phase controller bandwidth of the oscillation control module.

This function uses the current Q factor. This parameter can be identified through the Frequency Sweep module and it should be previously applied in the PLL Setup tool in order this function to get correctly the bandwidth.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Bandwidth (Hz)** (float32)
- **Error** described in the Response message>Body section

PLL.FreqRangeSet

Sets the frequency range of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Frequency range (Hz)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.FreqRangeGet

Returns the frequency range of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Frequency range (Hz)** (float32)
- **Error** described in the Response message>Body section

PLL.CenterFreqSet

Sets the center frequency of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Center frequency (Hz)** (float64)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.CenterFreqGet

Returns the center frequency of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Center frequency (Hz)** (float64)
- **Error** described in the Response message>Body section

PLL.FreqShiftSet

Sets the frequency shift of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Frequency shift (Hz)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.FreqShiftGet

Returns the frequency shift of the oscillation control module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Frequency shift (Hz)** (float32)
- **Error** described in the Response message>Body section

PLL.FreqShiftAutoCenter

Auto-centers frequency shift of the oscillation control module.

It works like the corresponding button on the oscillation control module. It adds the current frequency shift to the center frequency and sets the frequency shift to zero.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.FreqExcOverwriteSet

Sets the signals to overwrite the Frequency Shift and/or Excitation signals of the oscillation control module.

It works when the corresponding controller (phase, amplitude) is not active.

To get a list of the available signals, see directly in the software or use the *Signals.NamesGet* function to get the full list of available signals.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Excitation overwrite signal index** (int), where value -2 means no change
- **Frequency overwrite signal index** (int), where value -2 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.FreqExcOverwriteGet

Returns the signals to overwrite the Frequency Shift and/or Excitation signals of the oscillation control module.

It works when the corresponding controller (phase, amplitude) is not active.

To get a list of the available signals, see directly in the software or use the *Signals.NamesGet* function to get the full list of available signals.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Excitation overwrite signal index** (int)
- **Frequency overwrite signal index** (int)
- **Error** described in the Response message>Body section

PLL.DemodInputSet

Sets the input and the frequency generator of the selected demodulator.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1
- **Input** (unsigned int16), where value 0 means no change
- **Frequency generator** (unsigned int16), where value 0 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.DemodInputGet

Returns the input and the frequency generator of the selected demodulator.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Input** (unsigned int16)
- **Frequency generator** (unsigned int16)
- **Error** described in the Response message>Body section

PLL.DemodHarmonicSet

Sets which harmonic of the input signal is selected in the PLL lock-in of the selected demodulator.

Harmonic 1 corresponds to the modulation frequency.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1
- **Harmonic** (unsigned int16)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.DemodHarmonicGet

Returns which harmonic of the input signal is selected in the PLL lock-in of the selected demodulator.

Harmonic 1 corresponds to the modulation frequency.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Harmonic** (unsigned int16)
- **Error** described in the Response message>Body section

PLL.DemodPhasRefSet

Sets the phase reference of the selected demodulator.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1
- **Phase reference (deg)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.DemodPhasRefGet

Returns the phase reference of the selected demodulator.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Phase reference (deg)** (float32)
- **Error** described in the Response message>Body section

PLL.DemodFilterSet

Returns the filter order of the low-pass filter after the PLL lock-in for the selected demodulator.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1
- **Filter order** (unsigned int16)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL.DemodFilterGet

Returns the filter order of the low-pass filter after the PLL lock-in for the selected demodulator.

Arguments:

- **Demodulator index** (unsigned int16) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Filter order** (unsigned int16)
- **Error** described in the Response message>Body section

PLL Q-Control

PLLQCtrl.AccessRequest

Requests/Releases access of Q-Control to PLL Configuration.

Q-Control needs access to the PLL Configuration to work. Depending on the system, this might be automatic, or must be controlled manually.

Arguments:

- **Request/Release** (unsigned int32) requests access (1) or releases access (0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLQCtrl.AccessGet

Returns the access state of Q-Control to PLL Configuration.

Q-Control needs access to the PLL Configuration to work. Depending on the system, this might be automatic, or must be controlled manually.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Request/Release** (unsigned int32) is 1 when access is granted and 0 otherwise
- **Error** described in the Response message>Body section

PLLQCtrl.OnOffSet

Switches Q-Control on or off.

Arguments:

- **On/Off** (unsigned int32) switches Q-Control on (1) or off (0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLQCtrl.OnOffGet

Returns the status of Q-Control.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **On/Off** (unsigned int32) is 1 when Q-Control is on and 0 otherwise
- **Error** described in the Response message>Body section

PLLQCtrl.QGainSet

Sets the Q-Control gain.

The higher the gain, the more Q is enhanced/decreased (depending on the Q-Control phase). A gain of 1 (with phase 0, i.e. Q enhance mode) theoretically doubles the Q factor, but depending on the system Q-Control might not be stable for such high gains.

Arguments:

- **Q Gain** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLQCtrl.QGainGet

Returns the Q-Control gain.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Q Gain** (float32)
- **Error** described in the Response message>Body section

PLLQCtrl.PhaseSet

Sets the Q-Control phase.

Arguments:

- **Phase Adjustment** (int) provides predefined phase for Q-Control. A value of 0 sets phase 0 (enhancing Q), a value of 1 set phase 180 deg (decreasing Q), a value of 2 activates custom phase (so the phase can be set using 'Custom Phase'.
- **Custom Phase** (float32) set the custom phase. This parameter is interpreted only when ‘Phase Adjustment’ is set to custom (value 2).

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLQCtrl.PhaseGet

Returns the Q-Control phase.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Phase Adjustment** (int) returns the phase adjustment mode (0 = enhance, 1 = decrease, 2 = custom)
- **Phase Custom** (float32) returns the custom phase (only relevant when 'Phase Adjustment' is set to Custom)
- **Error** described in the Response message>Body section

PLL Frequency Sweep

PLLFreqSwp.Open

Opens the PLL Frequency Sweep module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLFreqSwp.ParamsSet

Sets the parameters of a frequency sweep.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Number of points** (int) is the number of points for the frequency sweep
- **Period (s)** (float32) is the measurement time at each frequency value. The same value is also used as wait time at each value, so it is better to use higher values for high Q factors
- **Settling time (s)** (float32) is the time to wait after setting the frequency shift to the start position

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLFreqSwp.ParamsGet

Returns the parameters of a frequency sweep.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of points** (int) is the number of points for the frequency sweep
- **Period (s)** (float32) is the measurement time at each frequency value. The same value is also used as wait time at each value, so it is better to use higher values for high Q factors
- **Settling time (s)** (float32) is the time to wait after setting the frequency shift to the start position
- **Error** described in the Response message>Body section

PLLFreqSwp.Start

Starts a frequency sweep.

Before using this function, set the center frequency and frequency range in the Oscillation Control module. Also, set the other parameters (number of points...) in the frequency sweep module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Get data** (unsigned int32), where if true (=1), the function will return the recorded Channels, the Data and the Characteristic values
- **Sweep direction** (unsigned int32), where if true (=1), the sweep is done from lower to upper limit, and if false (=0) is done from upper to lower limit.

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the data
- **Resonance frequency (Hz)** (float64)
- **Q factor** (float64)
- **Phase (deg)** (float32) at the resonance frequency
- **Amplitude to excitation quotient (nm/mV)** (float32)
- **Fit length** (int) is the number of samples used to draw the fit line when the Parameter Estimation Method for Q is Phase Slope
- **Number of points** (int) is the number of points distributed over the frequency range
- **Error** described in the Response message>Body section

PLLFreqSwp.Stop

Stops the sweep in the PLL Frequency Sweep module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL Phase Sweep

PLLPhasSwp.Start

Starts a phase sweep.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1
- **Get data** (unsigned int32), where if true (=1), the function will return the recorded Channels and the Data

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the data
- **Error** described in the Response message>Body section

.

PLLPhasSwp.Stop

Stops the sweep in the PLL Phase Sweep module.

Arguments:

- **Modulator index** (int) specifies which modulator or PLL to control. The valid values start from 1

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLL Signal Analyzer

PLLSignalAnlzr.Open

Opens the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.ChSet

Sets the channel of the PLL Signal Analyzer.

Arguments:

- **Channel index** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.ChGet

Returns the channel of the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel index** (int)
- **Error** described in the Response message>Body section

PLLSignalAnlzr.TimebaseSet

Sets the Time Base and Update Rate of the PLL Signal Analyzer.

Arguments:

- **Timebase** (int) Base is the index out of the list of Time Base values. Use the *PLLSignalAnlzr.TimebaseGet* function to get a list of the available time bases. Value -1 means no change
- **Update rate** (int) is the graph update rate, where 1 corresponds to the fastest rate and higher values reduce update speed, TCP traffic and CPU load. Value -1 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.TimebaseGet

Returns the Time Base and Update Rate of the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Timebase** (int) Base is the index out of the list of Time Base values. Use the *PLLSignalAnlzr.TimebaseGet* function to get a list of the available time bases. Value -1 means no change
- **Update rate** (int) is the graph update rate, where 1 corresponds to the fastest rate and higher values reduce update speed, TCP traffic and CPU load. Value -1 means no change
- **Timebases size** (int) is the size in bytes of the timebases array
- **Timebases number** (int) is the number of elements of the timebases array
- **Timebases** (1D array string) returns an array of timebases strings, where each string comes prepended by its size in bytes
- **Error** described in the Response message>Body section

PLLSignalAnlzr.TrigAuto

Sets the trigger parameters to pre-defined values in the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.TrigRearm

Rearms the trigger in the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.TrigSet

Sets the trigger configuration in the PLL Signal Analyzer.

Arguments:

- **Trigger mode** (unsigned int16) sets the trigger mode, where 0=no change, 1=Immediate, and 2=Level
- **Trigger source** (int) set the signal index on which the trigger works.
  The list of available signals is returned by the *PLLSignalAnlzr.FFTPropsGet* function
- **Trigger slope** (unsigned int16) sets the triggering direction, where 0=no change, 1=Rising, and 2=Falling
- **Trigger level** (float64) sets the trigger level
- **Trigger position (s)** (float64) sets the trigger position
- **Arming mode** (unsigned int16) sets whether the trigger is automatically (=2) or manually rearmed (=1).
  Value 0 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.TrigGet

Returns the trigger configuration in the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Trigger mode** (unsigned int16) sets the trigger mode, where 0=no change, 1=Immediate, and 2=Level
- **Trigger source** (int) set the signal index on which the trigger works.
  The list of available signals is returned by the *PLLSignalAnlzr.FFTPropsGet* function
- **Trigger slope** (unsigned int16) sets the triggering direction, where 0=no change, 1=Rising, and 2=Falling
- **Trigger level** (float64) sets the trigger level
- **Trigger position (s)** (float64) sets the trigger position
- **Arming mode** (unsigned int16) sets whether the trigger is automatically (=2) or manually rearmed (=1). Value 0 means no change
- **Trigger source signals list size** (int) is the size in bytes of the trigger source signals array
- **Trigger source signals list number** (int) is the number of elements of the trigger source signals array
- **Trigger source signals list** (1D array string) returns an array of trigger source signals, where each string comes prepended by its size in bytes
- **Error** described in the Response message>Body section

PLLSignalAnlzr.OsciDataGet

Returns the oscilloscope graph data from the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Data t0** (float64) is the timestamp of the 1st acquired point
- **Data dt** (float64) is the time distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float64) is the data acquired in the oscilloscope
- **Error** described in the Response message>Body section

PLLSignalAnlzr.FFTPropsSet

Sets the configuration in the spectrum section of the PLL Signal Analyzer.

Arguments:

- **FFT window** (unsigned int16) is the window function applied to the timed signal before calculating the Power Spectral Density. The indexes of the possible FFT windows are as follows:
  0=no change, 1=None, 2=Hanning, 3=Hamming, 4=Blackman-Harris, 5=Exact Blackman, 6=Blackman, 7=Flat Top, 8=4 Term B-Harris, 9=7 Term B-Harris, and 10=Low Sidelobe
- **Averaging mode** (unsigned int16) where 0 is no change, 1 is None, 2 is Vector, 3 is RMS, and 4 is Peak Hold
- **Weighting mode** (unsigned int16) where 0 is no change, 1 is Linear, and 2 is Exponential
- **Count** (int) specifies the number of averages used for RMS and Vector averaging. 0 means no change.
  If weighting mode is Exponential, the averaging process is continuous and new spectral data have a higher weighting than older ones.
  If weighting mode is Linear, the averaging combines count spectral records with equal weighting and then stops

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.FFTPropsGet

Returns the configuration in the spectrum section of the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **FFT window** (unsigned int16) is the window function applied to the timed signal before calculating the Power Spectral Density. The indexes of the possible FFT windows are as follows:
  0=None, 1=Hanning, 2=Hamming, 3=Blackman-Harris, 4=Exact Blackman, 5=Blackman, 6=Flat Top, 7=4 Term B-Harris, 8=7 Term B-Harris, and 9=Low Sidelobe
- **Averaging mode** (unsigned int16) where 0 is None, 1 is Vector, 2 is RMS, and 3 is Peak Hold
- **Weighting mode** (unsigned int16) where 0 is Linear, and 1 is Exponential
- **Count** (int) indicates the number of averages used for RMS and Vector averaging.
  If weighting mode is Exponential, the averaging process is continuous and new spectral data have a higher weighting than older ones.
  If weighting mode is Linear, the averaging combines count spectral records with equal weighting and then stops
- **Error** described in the Response message>Body section

PLLSignalAnlzr.FFTAvgRestart

Restarts the averaging in the spectrum section of the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLSignalAnlzr.FFTDataGet

Returns the spectrum graph data from the PLL Signal Analyzer.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Data f0** (float64) is the frequency of the 1st acquired point
- **Data df** (float64) is the frequency distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float64) is the data acquired in the spectrum section
- **Error** described in the Response message>Body section

PLL Zoom FFT

PLLZoomFFT.Open

Opens the PLL Zoom FFT module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLZoomFFT.ChSet

Sets the channel of the PLL Zoom FFT module.

Selecting a channel (PLL1 or PLL2) in the Zoom FFT module is only available when the Oscillation Control 2 module is licensed.

Arguments:

- **Channel index** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLZoomFFT.ChGet

Returns the channel of the PLL Zoom FFT module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel index** (int)
- **Error** described in the Response message>Body section

PLLZoomFFT.AvgRestart

Restarts the averaging in the PLL Zoom FFT module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLZoomFFT.PropsSet

Sets the configuration of the PLL Zoom FFT module.

Arguments:

- **FFT window** (unsigned int16) is the window function applied to the timed signal before calculating the Power Spectral Density. The indexes of the possible FFT windows are as follows:
  0=no change, 1=None, 2=Hanning, 3=Hamming, 4=Blackman-Harris, 5=Exact Blackman, 6=Blackman, 7=Flat Top, 8=4 Term B-Harris, 9=7 Term B-Harris, and 10=Low Sidelobe
- **Averaging mode** (unsigned int16) where 0 is no change, 1 is None, 2 is Vector, 3 is RMS, and 4 is Peak Hold
- **Weighting mode** (unsigned int16) where 0 is no change, 1 is Linear, and 2 is Exponential
- **Count** (int) specifies the number of averages used for RMS and Vector averaging. 0 means no change.
  If weighting mode is Exponential, the averaging process is continuous and new spectral data have a higher weighting than older ones.
  If weighting mode is Linear, the averaging combines count spectral records with equal weighting and then stops

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

PLLZoomFFT.PropsGet

Returns the configuration of the PLL Zoom FFT module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **FFT window** (unsigned int16) is the window function applied to the timed signal before calculating the Power Spectral Density. The indexes of the possible FFT windows are as follows:
  0=None, 1=Hanning, 2=Hamming, 3=Blackman-Harris, 4=Exact Blackman, 5=Blackman, 6=Flat Top, 7=4 Term B-Harris, 8=7 Term B-Harris, and 9=Low Sidelobe
- **Averaging mode** (unsigned int16) where 0 is None, 1 is Vector, 2 is RMS, and 3 is Peak Hold
- **Weighting mode** (unsigned int16) where 0 is Linear, and 1 is Exponential
- **Count** (int) indicates the number of averages used for RMS and Vector averaging.
  If weighting mode is Exponential, the averaging process is continuous and new spectral data have a higher weighting than older ones.
  If weighting mode is Linear, the averaging combines count spectral records with equal weighting and then stops
- **Error** described in the Response message>Body section

PLLZoomFFT.DataGet

Returns the data from the PLL Zoom FFT module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Data f0** (float64) is the frequency of the 1st acquired point
- **Data df** (float64) is the frequency distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float64) is the acquired data
- **Error** described in the Response message>Body section

OC Sync module

OCSync.AnglesSet

Sets the angle values used in the OC Sync module for digital channel 1 and 2.

The On angle is the angle of the Oscillation Control output (excitation) at which the corresponding digital channel is set to high.

The Off angle is the angle of the Oscillation Control output (excitation) at which the corresponding digital channel is set to low.

Arguments:

- **Channel 1 on angle (deg)** (float32)
- **Channel 1 off angle (deg)** (float32)
- **Channel 2 on angle (deg)** (float32)
- **Channel 3 off angle (deg)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OCSync.AnglesGet

Returns the angle values used in the OC Sync module for digital channel 1 and 2.

The On angle is the angle of the Oscillation Control output (excitation) at which the corresponding digital channel is set to high.

The Off angle is the angle of the Oscillation Control output (excitation) at which the corresponding digital channel is set to low.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel 1 on angle (deg)** (float32)
- **Channel 1 off angle (deg)** (float32)
- **Channel 2 on angle (deg)** (float32)
- **Channel 3 off angle (deg)** (float32)
- **Error** described in the Response message>Body section

OCSync.LinkAnglesSet

Sets the status of the Link functionality in the OC Sync module for digital channel 1 and 2.

When Link Angles is set, the difference between Off angle and On angle is kept constant and only On angle can be modified.

When Unlink Angles is set, both angles can be set independently.

When No Change is set, this function won't modify the status of the corresponding Link button in the OC Sync module.

Arguments:

- **Link angles Channel 1** (unsigned int32), where 0=no change, 1=Link Angles, and 2=Unlink angles
- **Link angles Channel 2** (unsigned int32), where 0=no change, 1=Link Angles, and 2=Unlink angles

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OCSync.LinkAnglesGet

Returns the status of the Link functionality in the OC Sync module for digital channel 1 and 2.

When Link Angles is used, the difference between Off angle and On angle is kept constant and only On angle can be modified.

When Unlink Angles is used, both angles can be set independently.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Link angles Channel 1** (unsigned int32), where 0= Unlink angles, and 1=Link Angles
- **Link angles Channel 2** (unsigned int32), where 0= Unlink angles, and 1=Link Angles
- **Error** described in the Response message>Body section

Script

Script.Load

Loads a script in the script module.

Arguments:

- **Script file path size** (int) is the number of characters of the script file path string
- **Script file path** (string) is the path of the script file to load
- **Load session** (unsigned int32) automatically loads the scripts from the session file bypassing the script file path argument, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.Save

Saves the current script in the specified .ini file.

Arguments:

- **Script file path size** (int) is the number of characters of the script file path string
- **Script file path** (string) is the path of the script file to save
- **Save session** (unsigned int32) automatically saves the current script into the session file bypassing the script file path argument, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.Deploy

Deploys a script in the script module.

Arguments:

- **Script index** (int) sets the script to deploy and covers a range from 0 (first script) to the total number of scripts minus one. A value of -1 sets the currently selected script to deploy.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.Undeploy

Undeploys a script in the script module.

Arguments:

- **Script index** (int) sets the script to undeploy and covers a range from 0 (first script) to the total number of scripts minus one. A value of -1 sets the currently selected script to undeploy.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.Run

Runs a script in the script module.

Arguments:

- **Script index** (int) sets the script to run and covers a range from 0 (first script) to the total number of scripts minus one. A value of -1 sets the currently selected script to run.
- **Wait until script finishes** (unsigned int32), where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.Stop

Stops the running script in the script module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.ChsGet

Returns the list of acquired channels in the Script module.

Arguments:

- **Acquire buffer** (unsigned int16) sets the Acquire Buffer number from which to read the list of channels. Valid values are 1 (=Acquire Buffer 1) and 2 (=Acquire Buffer 2).

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.
- **Error** described in the Response message>Body section

Script.ChsSet

Sets the list of acquired channels in the Script module.

Arguments:

- **Acquire buffer** (unsigned int16) sets the Acquire Buffer number from which to set the list of channels. Valid values are 1 (=Acquire Buffer 1) and 2 (=Acquire Buffer 2).
- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Script.DataGet

Returns the data acquired in the Script module.

Arguments:

- **Acquire buffer** (unsigned int16) sets the Acquire Buffer number from which to read the acquired data. Valid values are 1 (=Acquire Buffer 1) and 2 (=Acquire Buffer 2).
- **Sweep number** (int) selects the sweep this function will return the data from. Each sweep is configured as such in the script and it corresponds to each plot displayed in the graphs of the Script module. The sweep numbers start at 0.

Return arguments (if Send response back flag is set to True when sending request message):

- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the script data
- **Error** described in the Response message>Body section

Script.Autosave

Saves automatically to file the data stored in the Acquire Buffers after running a script in the Script module.

Arguments:

- **Acquire buffer** (unsigned int16) sets the Acquire Buffer number from which to save the data.
  Valid values are 0 (=Acquire Buffer 1 & Acquire Buffer 2), 1 (=Acquire Buffer 1), and 2 (=Acquire Buffer 2).
- **Sweep number** (int) selects the sweep this function will save the data for.
  Each sweep is configured as such in the script and it corresponds to each plot displayed in the graphs of the Script module.
  The sweep numbers start at 0. A value of -1 saves all acquired sweeps.
- **All sweeps to same file** (unsigned int32) decides if all sweeps defined by the Sweep number parameter are saved to the same file (=1) or not (=0).

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Interferometer

Interf.CtrlOnOffSet

Switches the interferometer controller On or Off.

Arguments:

- **Status** (unsigned int32) switches the interferometer controller Off (=0) or On (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Interf.CtrlOnOffGet

Returns the status of the interferometer controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) indicates whether the interferometer controller is Off (=0) or On (=1)
- **Error** described in the Response message>Body section

Interf.CtrlPropsSet

Sets the properties of the interferometer controller.

Arguments:

- **Integral** (float32) sets the integral gain of the controller
- **Proportional** (float32) sets the proportional gain of the controller
- **Sign** (unsigned int32) sets the sign of the controller. If 0, means negative, and if 1 means positive

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Interf.CtrlPropsGet

Returns the properties of the interferometer controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Integral** (float32) returns the integral gain of the controller
- **Proportional** (float32) returns the proportional gain of the controller
- **Sign** (unsigned int32) returns the sign of the controller. If 0, means negative, and if 1 means positive
- **Error** described in the Response message>Body section

Interf.WPiezoSet

Sets the position of the W-piezo.

To change the position of the W-piezo, the interferometer controller must be switched Off.

Arguments:

- **W-piezo** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Interf.WPiezoGet

Returns the position of the W-piezo.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **W-piezo** (float32)
- **Error** described in the Response message>Body section

Interf.ValGet

Returns the interferometer value.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Interferometer value** (float32)
- **Error** described in the Response message>Body section

Interf.CtrlCalibrOpen

Opens the calibration panel for the interferometer controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Interf.CtrlReset

Resets the interferometer controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Interf.CtrlNullDefl

Applies null deflection to the interferometer controller.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Laser module

Laser.OnOffSet

Switches the laser On or Off.

Arguments:

- **Status** (unsigned int32) switches the laser Off (=0) or On (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Laser.OnOffGet

Returns the status of the laser.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) indicates whether the laser is Off (=0) or On (=1)
- **Error** described in the Response message>Body section

Laser.PropsSet

Sets the laser properties.

Arguments:

- **Laser Setpoint** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Laser.PropsGet

Returns the laser properties.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Laser Setpoint** (float32)
- **Error** described in the Response message>Body section

Laser.PowerGet

Returns the laser power.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Laser power** (float32)
- **Error** described in the Response message>Body section

Beam Deflection

BeamDefl.HorConfigSet

Sets the configuration of the horizontal deflection.

Arguments:

- **Name size** (int) is the number of bytes of the Name string
- **Name** (string) is the name of the signal
- **Units size** (int) is the number of bytes of the Units string
- **Units** (string) is the physical units of the signal
- **Calibration** (float32)
- **Offset** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BeamDefl.HorConfigGet

Returns the configuration of the horizontal deflection.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Name size** (int) is the number of bytes of the Name string
- **Name** (string) is the name of the signal
- **Units size** (int) is the number of bytes of the Units string
- **Units** (string) is the physical units of the signal
- **Calibration** (float32)
- **Offset** (float32)
- **Error** described in the Response message>Body section

BeamDefl.VerConfigSet

Sets the configuration of the vertical deflection.

Arguments:

- **Name size** (int) is the number of bytes of the Name string
- **Name** (string) is the name of the signal
- **Units size** (int) is the number of bytes of the Units string
- **Units** (string) is the physical units of the signal
- **Calibration** (float32)
- **Offset** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BeamDefl.VerConfigGet

Returns the configuration of the vertical deflection.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Name size** (int) is the number of bytes of the Name string
- **Name** (string) is the name of the signal
- **Units size** (int) is the number of bytes of the Units string
- **Units** (string) is the physical units of the signal
- **Calibration** (float32)
- **Offset** (float32)
- **Error** described in the Response message>Body section

BeamDefl.IntConfigSet

Sets the configuration of the intensity signal.

Arguments:

- **Name size** (int) is the number of bytes of the Name string
- **Name** (string) is the name of the signal
- **Units size** (int) is the number of bytes of the Units string
- **Units** (string) is the physical units of the signal
- **Calibration** (float32)
- **Offset** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

BeamDefl.IntConfigGet

Returns the configuration of the intensity signal.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Name size** (int) is the number of bytes of the Name string
- **Name** (string) is the name of the signal
- **Units size** (int) is the number of bytes of the Units string
- **Units** (string) is the physical units of the signal
- **Calibration** (float32)
- **Offset** (float32)
- **Error** described in the Response message>Body section

BeamDefl.AutoOffset

Auto-offsets the Beam Deflection signal.

This function works like the corresponding buttons on the beam deflection module. It adds the current deflection value (vertical deflection, horizontal deflection or intensity) to the offset so the deflection signal is close to 0.

Arguments:

- **Deflection signal** (unsigned int16) selects the signal to correct the offset for. 0 means Horizontal, 1 means Vertical, and 2 means Intensity

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Signals

Signals.NamesGet

Returns the signals names list of the 128 signals available in the software.

The 128 signals are physical inputs, physical outputs and internal channels. By searching in the list the channel's name you are interested in, you can get its index (0-127).

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Signals names size** (int) is the size in bytes of the signals names array
- **Signals names number** (int) is the number of elements of the signals names array
- **Signals names** (1D array string) returns an array of signals names strings, where each string comes prepended by its size in bytes
- **Error** described in the Response message>Body sectionSignals.CalibrGet

Returns the calibration and offset of the selected signal.

Arguments:

- **Signal index** (int) is comprised between 0 and 127

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration per volt** (float32)
- **Offset in physical units** (float32)
- **Error** described in the Response message>Body section

Signals.RangeGet

Returns the range limits of the selected signal.

Arguments:

- **Signal index** (int) is comprised between 0 and 127

Return arguments (if Send response back flag is set to True when sending request message):

- **Maximum limit** (float32)
- **Minimum limit** (float32)
- **Error** described in the Response message>Body section

Signals.ValGet

Returns the current value of the selected signal (oversampled during the Acquisition Period time, Tap).

Signal measurement principle:

The signal is continuously oversampled with the Acquisition Period time, Tap, specified in the TCP receiver module. Every Tap second, the oversampled data is "published". This VI function waits for the next oversampled data to be published and returns its value. Calling this function does not trigger a signal measurement; it waits for data to be published! Thus, this function returns a value 0 to Tap second after being called.

An important consequence is that if you change a signal and immediately call this function to read a measurement you might get "old" data (i.e. signal data measured before you changed the signal). The solution to get only new data is to set Wait for newest data to True. In this case, the first published data is discarded and only the second one is returned.

Arguments:

- **Signal index** (int) is comprised between 0 and 127
- **Wait for newest data** (unsigned int32) selects whether the function returns the next available signal value or if it waits for a full period of new data. If False, this function returns a value 0 to Tap seconds after being called. If True, the function discard the first oversampled signal value received but returns the second value received. Thus, the function returns a value Tap to 2*Tap seconds after being called. It could be 0=False or 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Signal value** (float32) is the value of the selected signal in physical units
- **Error** described in the Response message>Body section

Signals.ValsGet

Returns the current values of the selected signals (oversampled during the Acquisition Period time, Tap).

Arguments:

- **Signals indexes size** (int) is the size of the Signals indexes array
- **Signals indexes** (1D array int) sets the selection of signals indexes, comprised between 0 and 127
- **Wait for newest data** (unsigned int32) selects whether the function returns the next available signal value or if it waits for a full period of new data. If False, this function returns a value 0 to Tap seconds after being called. If True, the function discard the first oversampled signal value received but returns the second value received. Thus, the function returns a value Tap to 2*Tap seconds after being called. It could be 0=False or 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Signals values size** (int) is the size of the Signals values array
- **Signals values** (1D array float32) returns the values of the selected signals in physical units
- **Error** described in the Response message>Body section

Signals.MeasNamesGet

Returns the list of measurement channels names available in the software.

Important Note: The Measurement channels don't correspond to the Signals. Measurement channels are used in sweepers whereas the Signals are used by the graphs and other modules.

By searching in the list the channels's names you are interested in, you can know its index. This index is then used e.g. to get/set the recorded channels in Sweepers, for example by using the *GenSwp.ChannelsGet* and *GenSwp.ChannelsSet* functions for the 1D Sweeper.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Measurement channels list size** (int) is the size in bytes of the Measurement channels list array
- **Number of Measurement channels** (int) is the number of elements of the Measurement channels list array
- **Measurement channels list** (1D array string) returns an array of names, where each array element is preceded by its size in bytes
- **Error** described in the Response message>Body section

Signals.AddRTGet

Returns the list of names of additional RT signals available, and the names of the signals currently assigned to the Internal 23 and 24 signals.

This can be found in the Signals Manager.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Additional RT signals names size** (int) is the size in bytes of the Additional RT signals names array
- **Number of Additional RT signals** (int) is the number of elements of the Additional RT signals names array
- **Additional RT signals names** (1D array string) returns the list of additional RT signals which can be assigned to Internal 23 and 24. Each array element is preceded by its size in bytes
- **Additional RT signal 1** (string) is the name of the RT signal assigned to the Internal 23 signal
- **Additional RT signal 2** (string) is the name of the RT signal assigned to the Internal 24 signal
- **Error** described in the Response message>Body section

Signals.AddRTSet

Assigns additional RT signals to the Internal 23 and 24 signals in the Signals Manager.

Arguments:

- **Additional RT signal 1** (int) is the index of the RT signal assigned to the Internal 23 signal
- **Additional RT signal 2** (int) is the index of the RT signal assigned to the Internal 24 signal

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

User Inputs

UserIn.CalibrSet

Sets the calibration of the selected user input.

Arguments:

- **Input index** (int) sets the input to be used, where index could be any value from 1 to the available inputs
- **Calibration per volt** (float32)
- **Offset in physical units** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

User Outputs

UserOut.ModeSet

Sets the mode (User Output, Monitor, Calculated signal) of the selected user output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Output mode** (unsigned int16) sets the output mode of the selected output, where 0=User Output, 1=Monitor, 2=Calc.Signal

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

UserOut.ModeGet

Returns the mode (User Output, Monitor, Calculated signal) of the selected user output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs

Return arguments (if Send response back flag is set to True when sending request message):

- **Output mode** (unsigned int16) returns the output mode of the selected output, where 0=User Output, 1=Monitor, 2=Calc.Signal, 3=Override
- **Error** described in the Response message>Body section

UserOut.MonitorChSet

Sets the monitor channel of the selected output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Monitor channel index** (int) sets the index of the channel to monitor. The index is comprised between 0 and 127 for the physical inputs, physical outputs, and internal channels. To see which signal has which index, see *Signals.NamesGet* function

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

UserOut.MonitorChGet

Returns the monitor channel of the selected output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs

Return arguments (if Send response back flag is set to True when sending request message):

- **Monitor channel index** (int) returns the index of the channel to monitor. The index is comprised between 0 and 127 for the physical inputs, physical outputs, and internal channels. To see which signal has which index, see *Signals.NamesGet* function
- **Error** described in the Response message>Body section

UserOut.ValSet

Sets the value of the selected user output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Output value** (float32) is the value applied to the selected user output in physical units

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

UserOut.CalibrSet

Sets the calibration of the selected user output or monitor channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Calibration per volt** (float32)
- **Offset in physical units** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

UserOut.CalcSignalNameSet

Sets the Calculated Signal name of the selected output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Calculated signal name size** (int) is the number of characters of the Calculated signal name string
- **Calculated signal name** (string) is the name of the calculated signal configured for the selected output

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

UserOut.CalcSignalNameGet

Returns the Calculated Signal name of the selected output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs

Return arguments (if Send response back flag is set to True when sending request message):

- **Calculated signal name size** (int) is the number of characters of the Calculated signal name string
- **Calculated signal name** (string) is the name of the calculated signal configured for the selected output
- **Error** described in the Response message>Body section

UserOut.CalcSignalConfigSet

Sets the configuration of the Calculated Signal for the selected output channel.

The configuration is a math operation between 2 signals, or the logarithmic value of one signal.

The possible values for the math operation are:
0=None, 1=Add, 1=Subtract, 3=Multiply, 4=Divide, 6=Log

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Signal 1** (unsigned int16) is the signal index (from 0 to 127) used as the first signal of the formula.
- **Operation** (unsigned int16) is the math operation.
- **Signal 2** (unsigned int16) is the signal index (from 0 to 127) used as the second signal of the formula.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section
-

UserOut.CalcSignalConfigGet

Returns the configuration of the Calculated Signal for the selected output channel.

The configuration is a math operation between 2 signals, or the logarithmic value of one signal.

The possible values for the math operation are:
0=None, 1=Add, 1=Subtract, 3=Multiply, 4=Divide, 6=Log

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs

Return arguments (if Send response back flag is set to True when sending request message):

- **Signal 1** (unsigned int16) is the signal index (from 0 to 127) used as the first signal of the formula.
- **Operation** (unsigned int16) is the math operation.
- **Signal 2** (unsigned int16) is the signal index (from 0 to 127) used as the second signal of the formula.
- **Error** described in the Response message>Body section

UserOut.LimitsSet

Sets the physical limits (in calibrated units) of the selected output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Upper limit** (float32) defines the upper physical limit of the user output
- **Lower limit** (float32) defines the lower physical limit of the user output

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

UserOut.LimitsGet

Returns the physical limits (in calibrated units) of the selected output channel.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs

Return arguments (if Send response back flag is set to True when sending request message):

- **Upper limit** (float32) defines the upper physical limit of the user output
- **Lower limit** (float32) defines the lower physical limit of the user output
- **Error** described in the Response message>Body section

Digital Lines

DigLines.PropsSet

Configures the properties of a digital line.

Arguments:

- **Digital line** (unsigned int32) defines the line to configure, from 1 to 8
- **Port** (unsigned int32) selects the digital port, where 0=Port A, 1=Port B, 2=Port C, 3=Port D
- **Direction** (unsigned int32) is the direction of the selected digital line, where 0=Input, 1=Output
- **Polarity** (unsigned int32), where 0=Low active, 1=High active

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DigLines.OutStatusSet

Sets the status of a digital output line.

Arguments:

- **Port** (unsigned int32) selects the digital port, where 0=Port A, 1=Port B, 2=Port C, 3=Port D
- **Digital line** (unsigned int32) defines the output line to configure, from 1 to 8
- **Status** (unsigned int32) sets whether the output is active or inactive, where 0=Inactive, 1=Active

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DigLines.TTLValGet

Reads the actual TTL voltages present at the pins of the selected port.

Arguments:

- **Port** (unsigned int16) selects the digital port, where 0=Port A, 1=Port B, 2=Port C, 3=Port D

Return arguments (if Send response back flag is set to True when sending request message):

- **TTL voltages size** (int) is the size of the TTL voltages array
- **TTL voltages** (1D array unsigned int32) sets whether the output is active or inactive, where 0=Inactive, 1=Active
- **Error** described in the Response message>Body section

DigLines.Pulse

Configures and starts the pulse generator on the selected digital outputs.

Arguments:

- **Port** (unsigned int16) selects the digital port, where 0=Port A, 1=Port B, 2=Port C, 3=Port D
- **Digital lines size** (int) is the size of the Digital lines array
- **Digital lines** (1D array unsigned int8) defines the output lines to pulse, from 1 to 8
- **Pulse width (s)** (float32) defines how long the outputs are active
- **Pulse pause (s)** (float32) defines how long the outputs are inactive
- **Number of pulses** (int) defines how many pulses to generate, where valid values are from 1 to 32767
- **Wait until finished** (unsigned int32), if True this function waits until all pulses have been generated before the response is sent back, where 0=False, 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Data Logger

DataLog.Open

Opens the Data Logger module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DataLog.Start

Starts the acquisition in the Data Logger module.

Before using this function, select the channels to record in the Data Logger.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DataLog.Stop

Stops the acquisition in the Data Logger module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DataLog.StatusGet

Returns the status parameters from the Data Logger module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Start time size** (int) returns the number of bytes corresponding to the Start time string
- **Start time** (string) returns a timestamp of the moment when the acquisition started
- **Acquisition elapsed hours** (unsigned int16) returns the number of hours already passed since the acquisition started
- **Acquisition elapsed minutes** (unsigned int16) returns the number of minutes displayed on the Data Logger
- **Acquisition elapsed seconds** (float32) returns the number of seconds displayed on the Data Logger
- **Stop time size** (int) returns the number of bytes corresponding to the Stop time string
- **Stop time** (string) returns a timestamp of the moment when the acquisition Stopped
- **Saved file path size** (int) returns the number of bytes corresponding to the Saved file path string
- **Saved file path** (string) returns the path of the last saved file
- **Points counter** (int) returns the number of points (averaged samples) to save into file.
  This parameter updates while running the acquisition
- **Error** described in the Response message>Body section

DataLog.ChsSet

Sets the list of recorded channels in the Data Logger module.

Arguments:

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DataLog.ChsGet

Returns the list of recorded channels in the Data Logger module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Error** described in the Response message>Body section

DataLog.PropsSet

Sets the acquisition configuration and the save options in the Data Logger module.

Arguments:

- **Acquisition mode** (unsigned int16) means that if Timed (=2), the selected channels are acquired during the acquisition duration time or until the user presses the Stop button.
  If Continuous (=1), the selected channels are acquired continuously until the user presses the Stop button.
  If 0, the is no change in the acquisition mode.
  The acquired data are saved every time the averaged samples buffer reaches 25.000 samples and when the acquisition stops
- **Acquisition duration(hours)** (int) sets the number of hours the acquisition should last. Value -1 means no change
- **Acquisition duration (minutes)** (int) sets the number of minutes. Value -1 means no change
- **Acquisition duration (seconds)** (float32) sets the number of seconds. Value -1 means no change
- **Averaging** (int) sets how many data samples (received from the real-time system) are averaged for one data point saved into file. By increasing this value, the noise might decrease, and fewer points per seconds are recorded.
  Use 0 to skip changing this parameter
- **Basename size** (int) is the size in bytes of the Basename string
- **Basename** (string) is base name used for the saved images
- **Comment size** (int) is the size in bytes of the Comment string
- **Comment** (string) is comment saved in the file
- **Size of the list of moduless** (int) is the size in bytes of the List of modules string array
- **Number of modules** (int) is the number of elements of the List of modules string array
- **List of modules** (1D array string) sets the modules names whose parameters will be saved in the header of the files. The size of each string item should come right before it as integer 32

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

DataLog.PropsGet

Returns the acquisition configuration and the save options in the Data Logger module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Acquisition mode** (unsigned int16) means that if Timed (=1), the selected channels are acquired during the acquisition duration time or until the user presses the Stop button.
  If Continuous (=0), the selected channels are acquired continuously until the user presses the Stop button. The acquired data are saved every time the averaged samples buffer reaches 25.000 samples and when the acquisition stops
- **Acquisition duration(hours)** (int) returns the number of hours the acquisition lasts
- **Acquisition duration (minutes)** (int) returns the number of minutes
- **Acquisition duration (seconds)** (float32) returns the number of seconds
- **Averaging** (int) returns how many data samples (received from the real-time system) are averaged for one data point saved into file
- **Basename size** (int) returns the size in bytes of the Basename string
- **Basename** (string) returns the base name used for the saved images
- **Comment size** (int) returns the size in bytes of the Comment string
- **Comment** (string) returns the comment saved in the file
- **Error** described in the Response message>Body section

TCP Logger

TCPLog.Start

Starts the acquisition in the TCP Logger module.

Before using this function, select the channels to record in the TCP Logger.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TCPLog.Stop

Stops the acquisition in the TCP Logger module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TCPLog.ChsSet

Sets the list of recorded channels in the TCP Logger module.

Arguments:

- **Number of channels** (int) is the number of recorded channels. It defines the size of the Channel indexes array
- **Channel indexes** (1D array int) are the indexes of recorded channels. The index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TCPLog.OversamplSet

Sets the oversampling value in the TCP Logger.

Arguments:

- **Oversampling value** (int) sets the oversampling index, where index could be any value from 0 to 1000

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

TCPLog.StatusGet

Returns the current status of the TCP Logger.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (int) returns an index which corresponds to one of the following status: 0=disconnected, 1=idle, 2=start, 3=stop, 4=running, 5=TCP connect, 6=TCP disconnect, 7=buffer overflow

- **Error** described in the Response message>Body section

Oscilloscope High Resolution

OsciHR.ChSet

Sets the channel index of the Oscilloscope High Resolution.

Arguments:

- **Channel index** (int) sets the channel to be used, where index could be any value from 0 to 15

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.ChGet

Returns the channel index of the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel index** (int) returns the channel used in the Oscilloscope High Resolution
- **Error** described in the Response message>Body section

OsciHR.OversamplSet

Sets the oversampling index of the Oscilloscope High Resolution.

Choosing to acquire data at lower rate than the maximum 1MS/s allows for an improved S/N ratio and also increases the time window for the acquisition for a given number of samples.

Arguments:

- **Oversampling index** (int) sets the oversampling index, where index could be any value from 0 to 10

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.OversamplGet

Returns the oversampling index of the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Oversampling index** (int) gets the oversampling index, where index could be any value from 0 to 10
- **Error** described in the Response message>Body section

OsciHR.CalibrModeSet

Sets the calibration mode of the Oscilloscope High Resolution.

Select between Raw Values or Calibrated Values. This setting affects the data displayed in the graph, and trigger level and hysteresis values.

Arguments:

- **Calibration mode** (unsigned int16), where 0=Raw values and 1=Calibrated values

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.CalibrModeGet

Returns the calibration mode of the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Calibration mode** (unsigned int16), where 0=Raw values and 1=Calibrated values
- **Error** described in the Response message>Body section

OsciHR.SamplesSet

Sets the number of samples to acquire in the Oscilloscope High Resolution.

Arguments:

- **Number of samples** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.SamplesGet

Returns the number of samples to acquire in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Number of samples** (int)
- **Error** described in the Response message>Body section

OsciHR.PreTrigSet

Sets the Pre-Trigger Samples or Seconds in the Oscilloscope High Resolution.

If Pre-Trigger (s) is NaN or Inf or below 0, Pre-Trigger Samples is taken into account instead of seconds.

Arguments:

- **Pre-Trigger samples** (unsigned int32)
- **Pre-Trigger (s)** (float64)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PreTrigGet

Returns the Pre-Trigger Samples in the Oscilloscope High Resolution.

If Pre-Trigger (s) is NaN or Inf or below 0, Pre-Trigger Samples is taken into account instead of seconds.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Pre-Trigger samples** (int)
- **Error** described in the Response message>Body section

OsciHR.Run

Starts the Oscilloscope High Resolution module.

The Oscilloscope High Resolution module does not run when its front panel is closed. To automate measurements it might be required to run the module first using this function.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.OsciDataGet

Returns the graph data from the Oscilloscope High Resolution.

Arguments:

- **Data to get** (unsigned int16), where 0=Current returns the currently displayed data and 1=Next trigger waits for the next trigger to retrieve data
- **Timeout (s)** (float64), tip

Return arguments (if Send response back flag is set to True when sending request message):

- **Data t0 size** (int) is the number of characters of the Data t0 string
- **Data t0** (string) is the timestamp of the 1st acquired point
- **Data dt** (float64) is the time distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float32) is the data acquired in the oscilloscope
- **Timeout** (unsigned int32) is 0 when no timeout occurred, and 1 when a timeout occurred
- **Error** described in the Response message>Body section

OsciHR.TrigModeSet

Sets the trigger mode in the Oscilloscope High Resolution.

Arguments:

- **Trigger mode** (unsigned int16), 0=Immediate means triggering immediately whenever the current data set is received by the host software, 1=Level where the trigger detection is performed on the non-averaged raw channel data (1MS/s), and 2=Digital where the trigger detection on the LS-DIO channels is performed at 500kS/s. Trigger detection on the HS-DIO channels is performed at 10MS/s

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigModeGet

Returns the trigger mode in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Trigger mode** (unsigned int16), where 0=Immediate, 1=Level, and 2=Digital
- **Error** described in the Response message>Body section

OsciHR.TrigLevChSet

Sets the Level Trigger Channel index in the Oscilloscope High Resolution.

Trigger detection is performed on the non-averaged raw channel data.

Arguments:

- **Level trigger channel index** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigLevChGet

Returns the Level Trigger Channel index in the Oscilloscope High Resolution.

Trigger detection is performed on the non-averaged raw channel data.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Level trigger channel index** (int)
- **Error** described in the Response message>Body section

OsciHR.TrigLevValSet

Sets the Level Trigger value in the Oscilloscope High Resolution.

Arguments:

- **Level trigger value** (float64)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigLevValGet

Returns the Level Trigger value in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Level trigger value** (float64)
- **Error** described in the Response message>Body section

OsciHR.TrigLevHystSet

Sets the Level Trigger Hysteresis in the Oscilloscope High Resolution.

Arguments:

- **Level trigger Hysteresis** (float64)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigLevHystGet

Returns the Level Trigger Hysteresis in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Level trigger Hysteresis** (float64)
- **Error** described in the Response message>Body section

OsciHR.TrigLevSlopeSet

Sets the Level Trigger Slope in the Oscilloscope High Resolution.

Arguments:

- **Level trigger slope** (unsigned int16), where 0=Rising and 1=Falling

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigLevSlopeGet

Returns the Level Trigger Slope in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Level trigger slope** (unsigned int16), where 0=Rising and 1=Falling
- **Error** described in the Response message>Body section

OsciHR.TrigDigChSet

Sets the Digital Trigger Channel in the Oscilloscope High Resolution.

Trigger detection on the LS-DIO channels is performed at 500kS/s. Trigger detection on the HS-DIO channels is performed at 10MS/s.

Arguments:

- **Digital trigger channel index** (int), where index can be any value from 0 to 35. Low Speed Port A lines are indexes 0 to 7, Low Speed Port B lines are indexes 8 to 15, Low Speed Port C lines are indexes 16 to 23, Low Speed Port D lines are indexes 24 to 31, and High Speed Port lines are indexes 32 to 35

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigDigChGet

Returns the Digital Trigger Channel in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Digital trigger channel index** (int), where index can be any value from 0 to 35. Low Speed Port A lines are indexes 0 to 7, Low Speed Port B lines are indexes 8 to 15, Low Speed Port C lines are indexes 16 to 23, Low Speed Port D lines are indexes 24 to 31, and High Speed Port lines are indexes 32 to 35
- **Error** described in the Response message>Body section

OsciHR.TrigArmModeSet

Sets the Trigger Arming Mode in the Oscilloscope High Resolution.

Arguments:

- **Trigger arming mode** (unsigned int16), where 0=Single shot means recording the next available data and stopping acquisition. and 1=Continuous means recording every available data and automatically re-triggers the acquisition

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigArmModeGet

Returns the Trigger Arming Mode in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Trigger arming mode** (unsigned int16), where 0=Single shot means recording the next available data and stopping acquisition. and 1=Continuous means recording every available data and automatically re-triggers the acquisition
- **Error** described in the Response message>Body section

OsciHR.TrigDigSlopeSet

Sets the Digital Trigger Slope in the Oscilloscope High Resolution.

Arguments:

- **Digital trigger slope** (unsigned int16), where 0=Rising and 1=Falling

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.TrigDigSlopeGet

Returns the Digital Trigger Slope in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Digital trigger slope** (unsigned int16), where 0=Rising and 1=Falling
- **Error** described in the Response message>Body section

OsciHR.TrigRearm

Rearms the trigger in the Oscilloscope High Resolution module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDShow

Shows or hides the PSD section of the Oscilloscope High Resolution.

Arguments:

- **Show PSD section** (unsigned int32), where 0=Hide and 1=Show

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDWeightSet

Sets the PSD Weighting in the Oscilloscope High Resolution.

Arguments:

- **PSD Weighting** (unsigned int16), where 0=Linear means that the averaging combines Count spectral records with equal weighting and then stops, whereas 1=Exponential means that the averaging process is continuous and new spectral data have a higher weighting than older ones

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDWeightGet

Returns the PSD Weighting in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **PSD Weighting** (unsigned int16), where 0=Linear means that the averaging combines Count spectral records with equal weighting and then stops, whereas 1=Exponential means that the averaging process is continuous and new spectral data have a higher weighting than older ones
- **Error** described in the Response message>Body section

OsciHR.PSDWindowSet

Sets the PSD Window Type in the Oscilloscope High Resolution.

Arguments:

- **PSD window type** (unsigned int16) is the window function applied to the timed signal before calculating the power spectral density, where 0=None, 1=Hanning, 2=Hamming, etc

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDWindowGet

Returns the PSD Window Type in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **PSD window type** (unsigned int16) is the window function applied to the timed signal before calculating the power spectral density, where 0=None, 1=Hanning, 2=Hamming, etc
- **Error** described in the Response message>Body section

OsciHR.PSDAvrgTypeSet

Sets the PSD Averaging Type in the Oscilloscope High Resolution.

Arguments:

- **PSD averaging type** (unsigned int16), where 0=None, 1=Vector, 2=RMS, 3=Peak hold

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDAvrgTypeGet

Returns the PSD Averaging Type in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **PSD averaging type** (unsigned int16), where 0=None, 1=Vector, 2=RMS, 3=Peak hold
- **Error** described in the Response message>Body section

OsciHR.PSDAvrgCountSet

Sets the PSD Averaging Count used by the RMS and Vector averaging types in the Oscilloscope High Resolution.

Arguments:

- **PSD averaging count** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDAvrgCountGet

Returns the PSD Averaging Count used by the RMS and Vector averaging types in the Oscilloscope High Resolution.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **PSD averaging count** (int)
- **Error** described in the Response message>Body section

OsciHR.PSDAvrgRestart

Restarts the PSD averaging process in the Oscilloscope High Resolution module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

OsciHR.PSDDataGet

Returns the Power Spectral Density data from the Oscilloscope High Resolution.

Arguments:

- **Data to get** (unsigned int16), where 0=Current returns the currently displayed data and 1=Next trigger waits for the next trigger to retrieve data
- **Timeout (s)** (float64), where -1 means waiting forever

Return arguments (if Send response back flag is set to True when sending request message):

- **Data f0** (float64) is the x coordinate of the 1st acquired point
- **Data df** (float64) is the frequency distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float64) is the PSD data acquired in the oscilloscope
- **Timeout** (unsigned int32) is 0 when no timeout occurred, and 1 when a timeout occurred
- **Error** described in the Response message>Body section

Oscilloscope 1-Channel

Osci1T.ChSet

Sets the channel to display in the Oscilloscope 1-Channel.

Arguments:

- **Channel index** (int) sets the channel to be used, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci1T.ChGet

Returns the channel displayed in the Oscilloscope 1-Channel.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel index** (int) returns the channel used, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Error** described in the Response message>Body section

Osci1T.TimebaseSet

Sets the timebase in the Oscilloscope 1-Channel.

To set the timebase, use the *Osci1T.TimebaseGet* function first to obtain a list of available timebases. Then, use the index of the desired timebase as input to this function.

The available timebases depend on the RT frequency and the RT oversampling.

Arguments:

- **Timebase index** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci1T.TimebaseGet

Returns the timebase in the Oscilloscope 1-Channel.

The available timebases depend on the RT frequency and the RT oversampling.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Timebase index** (int) returns the index of the selected timebase
- **Number of timebases** (int) returns the number of elements in the Timebases array
- **Timebases (s)** (1D array float32) returns an array of the timebases values in seconds
- **Error** described in the Response message>Body section

Osci1T.TrigSet

Sets the trigger configuration in the Oscilloscope 1-Channel.

Arguments:

- **Trigger mode** (unsigned int16) sets the triggering mode. For Immediate mode (=0) no further configuration is required. 1 means Level, and 2 means Auto
- **Trigger slope** (unsigned int16) sets whether to trigger on rising (=1) or falling (=0) slope of the signal
- **Trigger level** (float64) sets the value the signal must cross (in the direction specified in slope) to trigger
- **Trigger hysteresis** (float64) is used to prevent noise from causing a false trigger.
  For a rising edge trigger slope, the signal must pass below (level – hysteresis) before a trigger level crossing is detected.
  For a falling edge trigger slope, the signal must pass above (level + hysteresis) before a trigger level crossing is detected

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci1T.TrigGet

Returns the trigger configuration in the Oscilloscope 1-Channel.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Trigger mode** (unsigned int16) returns the triggering mode. 0 means Immediate mode, 1 means Level, and 2 means Auto
- **Trigger slope** (unsigned int16) returns whether to trigger on rising (=1) or falling (=0) slope of the signal
- **Trigger level** (float64) returns the value the signal must cross (in the direction specified in slope) to trigger
- **Trigger hysteresis** (float64) is used to prevent noise from causing a false trigger.
  For a rising edge trigger slope, the signal must pass below (level – hysteresis) before a trigger level crossing is detected.
  For a falling edge trigger slope, the signal must pass above (level + hysteresis) before a trigger level crossing is detected
- **Error** described in the Response message>Body section

Osci1T.Run

Starts the Oscilloscope 1-Channel.

This module does not run when its front panel is closed. To automate measurements it is required to run the module first using this function.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci1T.DataGet

Returns the graph data from the Oscilloscope 1-Channel.

Arguments:

- **Data to get** (unsigned int16), where 0=Current returns the currently displayed data, 1=Next trigger waits for the next trigger to retrieve data, and 2=wait 2 triggers

Return arguments (if Send response back flag is set to True when sending request message):

- **Data t0** (float64) is the timestamp of the 1st acquired point
- **Data dt** (float64) is the time distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float64) is the data acquired in the oscilloscope
- **Error** described in the Response message>Body section

Oscilloscope 2-Channels

Osci2T.ChsSet

Sets the channels to display in the Oscilloscope 2-Channels.

Arguments:

- **Channel A index** (int) sets the channel A, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Channel B index** (int) sets the channel B, where the index is comprised between 0 and 127

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci2T.ChsGet

Returns the channels displayed in the Oscilloscope 2-Channels.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel A index** (int) returns the channel A, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Channel B index** (int) returns the channel B, where the index is comprised between 0 and 127
- **Error** described in the Response message>Body section

Osci2T.TimebaseSet

Sets the timebase in the Oscilloscope 2-Channels.

To set the timebase, use the *Osci2T.TimebaseGet* function first to obtain a list of available timebases. Then, use the index of the desired timebase as input to this function.

The available timebases depend on the RT frequency and the RT oversampling.

Arguments:

- **Timebase index** (unsigned int16)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci2T.TimebaseGet

Returns the timebase in the Oscilloscope 2-Channels.

The available timebases depend on the RT frequency and the RT oversampling.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Timebase index** (unsigned int16) returns the index of the selected timebase
- **Number of timebases** (int) returns the number of elements in the Timebases array
- **Timebases (s)** (1D array float32) returns an array of the timebases values in seconds
- **Error** described in the Response message>Body section

Osci2T.OversamplSet

Sets the oversampling in the Oscilloscope 2-Channels.

Arguments:

- **Oversampling index** (unsigned int16) defines how many integer number of samples each data point is averaged over. Index 0 means 50 samples, index 1 means 20, index 2 means 10, index 3 means 5, index 4 means 2, and index 5 means 1 sample (so no averaging)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci2T.OversamplGet

Returns the oversampling in the Oscilloscope 2-Channels.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Oversampling index** (unsigned int16) defines how many integer number of samples each data point is averaged over. Index 0 means 50 samples, index 1 means 20, index 2 means 10, index 3 means 5, index 4 means 2, and index 5 means 1 sample (so no averaging)
- **Error** described in the Response message>Body section

Osci2T.TrigSet

Sets the trigger configuration in the Oscilloscope 2-Channels.

Arguments:

- **Trigger mode** (unsigned int16) sets the triggering mode. For Immediate mode (=0) no further configuration is required. 1 means Level, and 2 means Auto
- **Trigger channel** (unsigned int16) sets the channel used to trigger. CHA is 0 and CHB is 1
- **Trigger slope** (unsigned int16) sets whether to trigger on rising (=1) or falling (=0) slope of the signal
- **Trigger level** (float64) sets the value the signal must cross (in the direction specified in slope) to trigger
- **Trigger hysteresis** (float64) is used to prevent noise from causing a false trigger.
  For a rising edge trigger slope, the signal must pass below (level – hysteresis) before a trigger level crossing is detected.
  For a falling edge trigger slope, the signal must pass above (level + hysteresis) before a trigger level crossing is detected
- **Trigger position (s)** (float64) sets the pre-sampling trigger position in seconds

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci2T.TrigGet

Returns the trigger configuration in the Oscilloscope 2-Channels.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Trigger mode** (unsigned int16) returns the triggering mode. 0 means Immediate, 1 means Level, and 2 means Auto
- **Trigger channel** (unsigned int16) returns the channel used to trigger. CHA is 0 and CHB is 1
- **Trigger slope** (unsigned int16) returns whether to trigger on rising (=1) or falling (=0) slope of the signal
- **Trigger level** (float64) returns the value the signal must cross (in the direction specified in slope) to trigger
- **Trigger hysteresis** (float64) is used to prevent noise from causing a false trigger.
  For a rising edge trigger slope, the signal must pass below (level – hysteresis) before a trigger level crossing is detected.
  For a falling edge trigger slope, the signal must pass above (level + hysteresis) before a trigger level crossing is detected
- **Trigger position (s)** (float64) returns the pre-sampling trigger position in seconds
- **Error** described in the Response message>Body section

Osci2T.Run

Starts the Oscilloscope 2-Channels.

This module does not run when its front panel is closed. To automate measurements it is required to run the module first using this function.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Osci2T.DataGet

Returns the graph data from the Oscilloscope 2-Channels.

Arguments:

- **Data to get** (unsigned int16), where 0=Current returns the currently displayed data, 1=Next trigger waits for the next trigger to retrieve data, and 2=wait 2 triggers

Return arguments (if Send response back flag is set to True when sending request message):

- **Data t0** (float64) is the timestamp of the 1st acquired point
- **Data dt** (float64) is the time distance between two acquired points
- **Data Y size** (int) is the number of data points in Data CHA and Data CHB
- **Data CHA** (1D array float64) is the CHA data acquired in the oscilloscope
- **Data CHB** (1D array float64) is the CHB data acquired in the oscilloscope
- **Error** described in the Response message>Body section

Signal Chart

SignalChart.Open

Opens the Signal Chart module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SignalChart.ChsSet

Sets the channels to display in the Signal Chart module.

Arguments:

- **Channel A index** (int) sets the channel A, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Channel B index** (int) sets the channel B, where the index is comprised between 0 and 127

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SignalChart.ChsGet

Returns the channels to display in the Signal Chart module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel A index** (int) sets the channel A, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Channel B index** (int) sets the channel B, where the index is comprised between 0 and 127
- **Error** described in the Response message>Body section

Spectrum Analyzer

SpectrumAnlzr.ChSet

Sets the channel to display in the selected Spectrum Analyzer module.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **Channel index** (int) sets the channel to be used, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.ChGet

Returns the channel to display in the selected Spectrum Analyzer module.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Channel index** (int) returns the channel used, where the index is comprised between 0 and 127, and it corresponds to the full list of signals available in the system.
  To get the signal name and its corresponding index in the list of the 128 available signals in the Nanonis Controller, use the *Signal.NamesGet* function, or check the RT Idx value in the Signals Manager module
- **Error** described in the Response message>Body section

SpectrumAnlzr.FreqRangeSet

Sets the frequency range in the selected Spectrum Analyzer module.

To set the frequency range, use *SpectrumAnlzr.FreqRangeGet* first to obtain a list of available frequency ranges. Then, use the index of the desired range as input to this function.

The available frequency ranges depend on the RT frequency and the RT oversampling.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **Range index** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.FreqRangeGet

Returns the frequency range in the selected Spectrum Analyzer module.

The available frequency ranges depend on the RT frequency and the RT oversampling.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Range index** (int) is the index of the selected frequency range
- **Number of frequency ranges** (int) returns the number of elements in the Frequency ranges array
- **Frequency ranges (Hz)** (1D array float32) returns an array of the frequency ranges in Hz
- **Error** described in the Response message>Body section

SpectrumAnlzr.FreqResSet

Sets the frequency resolution in the selected Spectrum Analyzer module.

To set the frequency resolution, use *SpectrumAnlzr.FreqResGet* first to obtain a list of available frequency resolutions. Then, use the index of the desired resolution as input to this function.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **Resolution index** (unsigned int16)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.FreqResGet

Returns the frequency resolution in the selected Spectrum Analyzer module.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Resolution index** (unsigned int16) is the index of the selected frequency resolution
- **Number of frequency resolutions** (int) returns the number of elements in the Frequency resolutions array
- **Frequency resolutions (Hz)** (1D array float32) returns an array of the frequency resolutions in Hz
- **Error** described in the Response message>Body section

SpectrumAnlzr.FFTWindowSet

Sets the FFT window in the selected Spectrum Analyzer module.

The indexes of the possible FFT windows are as follows: 0=None, 1=Hanning, 2=Hamming, 3=Blackman-Harris, 4=Exact Blackman, 5=Blackman, 6=Flat Top, 7=4 Term B-Harris, 8=7 Term B-Harris, and 9=Low Sidelobe

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **FFT window index** (unsigned int16)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.FFTWindowGet

Returns the FFT window selected in the selected Spectrum Analyzer module.

The indexes of the possible FFT windows are as follows: 0=None, 1=Hanning, 2=Hamming, 3=Blackman-Harris, 4=Exact Blackman, 5=Blackman, 6=Flat Top, 7=4 Term B-Harris, 8=7 Term B-Harris, and 9=Low Sidelobe

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **FFT window index** (unsigned int16)
- **Error** described in the Response message>Body section

SpectrumAnlzr.AveragSet

Sets the averaging parameters in the selected Spectrum Analyzer module.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **Averaging mode** (unsigned int16) where 0 is None, 1 is Vector, 2 is RMS, and 3 is Peak Hold
- **Weighting mode** (unsigned int16) where 0 is Linear and 1 is Exponential
- **Count** (unsigned int32) specifies the number of averages used for RMS and Vector averaging.
  If weighting mode is Exponential, the averaging process is continuous and new spectral data have a higher weighting than older ones.
  If weighting mode is Linear, the averaging combines count spectral records with equal weighting and then stops

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.AveragGet

Returns the averaging parameters in the selected Spectrum Analyzer module.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Averaging mode** (unsigned int16) where 0 is None, 1 is Vector, 2 is RMS, and 3 is Peak Hold
- **Weighting mode** (unsigned int16) where 0 is Linear and 1 is Exponential
- **Count** (unsigned int32) specifies the number of averages used for RMS and Vector averaging.
  If weighting mode is Exponential, the averaging process is continuous and new spectral data have a higher weighting than older ones.
  If weighting mode is Linear, the averaging combines count spectral records with equal weighting and then stops
- **Error** described in the Response message>Body section

SpectrumAnlzr.ACCouplingSet

Sets the AC coupling mode in the selected Spectrum Analyzer module.

Use the associated *SpectrumAnlzr.DCGet* function to return the DC component when this method is activated.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **AC coupling** (unsigned int32) switches the AC coupling Off (=0) or On (=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.ACCouplingGet

Returns the AC coupling mode in the selected Spectrum Analyzer module.

Use the associated *SpectrumAnlzr.DCGet* function to return the DC component when this method is activated.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **AC coupling** (unsigned int32) returns whether the AC coupling is Off (=0) or On (=1)
- **Error** described in the Response message>Body section

SpectrumAnlzr.CursorPosSet

Sets the position of the cursors in the selected Spectrum Analyzer module.

Cursors 1 and 2 are used to define the frequency band.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function will apply changes to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **Cursor type** (unsigned int16) sets the type of cursor to display. 0 means x_y, 1 means dx_dy, 2 means x1_x2_dx, 3 means y1_y2_dy, 4 means RMS_df, and 5 means no change
- **Position X Cursor 1 (Hz)** (float64) sets the X position of cursor 1
- **Position X Cursor 2 (Hz)** (float64) sets the X position of cursor 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.CursorPosGet

Returns the position of the cursors in the selected Spectrum Analyzer module.

Cursors 1 and 2 are used to define the frequency band.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2
- **Cursor type** (unsigned int16) sets the type of cursor to display. 0 means x_y, 1 means dx_dy, 2 means x1_x2_dx, 3 means y1_y2_dy, 4 means RMS_df, and 5 means no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Position X Cursor 1 (Hz)** (float64) returns the X position of cursor 1
- **Position X Cursor 2 (Hz)** (float64) returns the X position of cursor 2
- **Position Y Cursor 1** (float64) returns the Y position of cursor 1
- **Error** described in the Response message>Body section

SpectrumAnlzr.BandRMSGet

Returns the RMS value in the frequency band from the Spectrum Analyzer modules.

By using this function the cursor type of the module is set to Band RMS if previously set to another type.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Band RMS** (float64)
- **Minimum frequency (Hz)** (float64)
- **Maximum frequency (Hz)** (float64)
- **Error** described in the Response message>Body section

SpectrumAnlzr.DCGet

Returns the DC value from the Spectrum Analyzer modules if the AC coupling mode is activated.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **DC value** (float64)
- **Error** described in the Response message>Body section

SpectrumAnlzr.Run

Starts the Spectrum Analyzer modules.

The Spectrum Analyzer does not run when its front panel is closed. To automate measurements it is required to run the module first using this function.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

SpectrumAnlzr.DataGet

Returns the data from the Spectrum Analyzer modules.

Arguments:

- **Spectrum Analyzer instance** (int) selects the Spectrum Analyzer instance this function refers to. Valid values are 1 for Spectrum Analyzer 1, and 2 for Spectrum Analyzer 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Data f0** (float64) is the x coordinate of the 1st acquired point
- **Data df** (float64) is the frequency distance between two acquired points
- **Data Y size** (int) is the number of data points in Data Y
- **Data Y** (1D array float64) is the data acquired in the Spectrum Analyzer
- **Error** described in the Response message>Body section

Function Generator 1-Channel

FunGen1Ch.Start

Starts the generation of the waveform on AO8 (SC4) or FAST AO (SC5) through the Function Generator module.

Arguments:

- **Periods** (int) is the number of periods to generate. Set -2 for continuous movement until Stop executes
- **Wait until finished** (unsigned int32) defines if this function waits for the generation to finish (=1) or not (=0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen1Ch.Stop

Stops the generation of the waveform through the Function Generator module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen1Ch.StatusGet

Gets the status of the waveform generation on AO8 (SC4) or FAST AO (SC5) through the Function Generator module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) indicates if the generation is running (=1) or not (=0)
- **Periods left** (int) is the number of periods left to generate
- **Error** described in the Response message>Body section

FunGen1Ch.PropsSet

Sets the amplitude, frequency, polarity, and direction of the waveform generated on AO8 (SC4) or FAST AO (SC5) through the Function Generator module.

Arguments:

- **Amplitude (%)** (float32) is the amplitude of the generated waveform in %, where 100% corresponds to +/- 10V at the output of the SC4/5
- **Frequency (Hz)** (float32) is the frequency of the generated waveform in Hz
- **Polarity** (unsigned int16) switches between positive and negative sign of the generated waveform. This will invert the waveform along the y axis. 0 means no change, 1 means Negative, and 2 means Positive
- **Direction** (unsigned int16) switches between forward and backward waveform. This will invert the waveform along the x axis (reverse the time). 0 means no change, 1 means Backward, and 2 means Forward

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen1Ch.PropsGet

Returns the amplitude, frequency, polarity, and direction of the waveform generated on AO8 (SC4) or FAST AO (SC5) through the Function Generator module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Amplitude (%)** (float32) is the amplitude of the generated waveform in %, where 100% corresponds to +/- 10V at the output of the SC4/5
- **Frequency (Hz)** (float32) is the frequency of the generated waveform in Hz
- **Polarity** (unsigned int16) is the positive (=0) or negative (=1) sign of the generated waveform
- **Direction** (unsigned int16) is the forward (=0) or backward (=1) direction of the generated waveform
- **Error** described in the Response message>Body section

FunGen1Ch.IdleSet

Sets the idle value in the Function Generator module.

The idle value is the value of AO8 (SC4) or FAST AO (SC5) when the function generator is not running.

Arguments:

- **Idle value** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen1Ch.IdleGet

Returns the idle value in the Function Generator module.

The idle value is the value of AO8 (SC4) or FAST AO (SC5) when the function generator is not running.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Idle value** (float32)
- **Error** described in the Response message>Body section

Function Generator 2-Channels

FunGen2Ch.Start

Starts the generation of the waveforms on the FAST AO or any available analog output through the Function Generator 2-Channels module.

Arguments:

- **Periods** (int) is the number of periods to generate. Set -2 for continuous movement until Stop executes
- **Wait until finished** (unsigned int32) defines if this function waits for the generation to finish (=1) or not (=0)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.Stop

Stops the generation of the waveform through the Function Generator 2-Channels module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.StatusGet

Returns the status of the waveforms generation in the Function Generator 2-Channels module.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) indicates if the generation is running (=1) or not (=0)
- **Periods left** (int) is the number of periods left to generate
- **Error** described in the Response message>Body section

FunGen2Ch.IdleSet

Sets the idle value in the Function Generator 2-Channels module.

The idle value is the value of the FAST AO (for the selected SC5) when the function generator is not running.

Arguments:

- **Device** (int) selects the SC5 where the idle value will be applied. 0 corresponds to SC5 1, 1 corresponds to SC5 2, and so on
- **Idle value** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.IdleGet

Returns the idle value in the Function Generator 2-Channels module.

The idle value is the value of the FAST AO (for the selected SC5) when the function generator is not running.

Arguments:

- **Device** (int) selects the SC5 the idle value will be read from. 0 corresponds to SC5 1, 1 corresponds to SC5 2, and so on

Return arguments (if Send response back flag is set to True when sending request message):

- **Idle value** (float32)
- **Error** described in the Response message>Body section

FunGen2Ch.OnOffSet

Sets the status of the On/Off switch of the selected channel in the Function Generator 2-Channels module.

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2
- **Status** (unsigned int32) switches the channel Off (=0) or On(=1)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.OnOffGet

Returns the status of the On/Off switch of Channel 1 in the Function Generator 2-Channels module.

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Status** (unsigned int32) returns whether the channel is Off (=0) or On(=1)
- **Error** described in the Response message>Body section

FunGen2Ch.SignalSet

Sets the signal assigned to the selected channel in the Function Generator 2-Channels module.

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2
- **Signal index** (int) where the possible values are: 0 for the Fast AO, 1 for AO1, 2 for AO2, 3 for AO3 and so on

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.SignalGet

Returns the signal assigned to the selected channel in the Function Generator 2-Channels module.

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Signal index** (int) where the values are: 0 for the Fast AO, 1 for AO1, 2 for AO2, 3 for AO3 and so on
- **Error** described in the Response message>Body section

FunGen2Ch.PropsSet

Sets the amplitude, time, polarity, direction, and state of the Add/Zero switch of the waveform generated on the selected channel in the Function Generator 2-Channels module.

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2
- **Amplitude (%)** (float32) is the amplitude in % of the waveform generated on Channel 1, where 100% corresponds to +/- 10V at the output of the SC5
- **Time (s)** (float32) is the time for each period of the generated waveforms in seconds.
  The time is such, so that the frequency can be set in multiples of 0.4768 Hz up to 15.625 kHz
- **Polarity** (unsigned int16) switches between positive (=2) and negative (=1) sign of the waveform generated on Channel 1. If value=0, there is no change.
  This will invert the waveform along the y axis
- **Direction** (unsigned int16) switches between forward (=2) and backward (=1) waveforms.
  If value=0, there is no change.
  This will invert the waveforms along the x axis (reverse the time). This is applied to both channels
- **Add/Zero** (unsigned int16) means that when Zero (=2), the waveform is generated on Channel 1 around zero. When Add (=1), adds the waveform to the current Idle/DC value. If value=0, there is no change

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.PropsGet

Returns the amplitude, time, polarity, direction, and state of the Add/Zero switch of the waveform generated on the selected channel in the Function Generator 2-Channels module.

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Amplitude (%)** (float32) is the amplitude in % of the waveform generated on Channel 1, where 100% corresponds to +/- 10V at the output of the SC5
- **Time (s)** (float32) is the time for each period of the generated waveforms in seconds.
  The time is such, so that the frequency can be set in multiples of 0.4768 Hz up to 15.625 kHz
- **Polarity** (unsigned int16) switches between positive (=0) and negative (=1) sign of the waveform generated on Channel 1
  This will invert the waveform along the y axis
- **Direction** (unsigned int16) switches between forward (=0) and backward (=1) waveforms.
  This will invert the waveforms along the x axis (reverse the time). This is applied to both channels
- **Add/Zero** (unsigned int16) means that when Zero (=0), the waveform is generated on Channel 1 around zero. When Add (=1), adds the waveform to the current Idle/DC value
- **Error** described in the Response message>Body section

FunGen2Ch.WaveformSet

Sets the shape of the waveform to generate on the selected channel in the Function Generator 2-Channels module.

Possible values for the shape are: 0=linear bipolar, 1=linear unipolar, 2=quadratic bipolar, 3=quadratic unipolar, 4=triangle bipolar, 5=triangle unipolar, 6=square bipolar, 7=square unipolar, 8=sine bipolar, 9=sine unipolar

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2
- **Shape** (unsigned int16)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

FunGen2Ch.WaveformGet

Returns the shape of the waveform generated on the selected channel in the Function Generator 2-Channels module.

Possible values for the shape are: 0=linear bipolar, 1=linear unipolar, 2=quadratic bipolar, 3=quadratic unipolar, 4=triangle bipolar, 5=triangle unipolar, 6=square bipolar, 7=square unipolar, 8=sine bipolar, 9=sine unipolar, 10=custom

Arguments:

- **Channel index** (int) selects the channel. Valid values are 1 for Channel 1 and 2 for Channel 2

Return arguments (if Send response back flag is set to True when sending request message):

- **Shape** (unsigned int16)
- **Error** described in the Response message>Body section

Utilities

Util.SessionPathGet

Returns the session path.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Session path size** (int) is the number of characters of the Session path string
- **Session path** (string)
- **Error** described in the Response message>Body section

Util.SessionPathSet

Sets the session folder path.

Arguments:

- **Session path size** (int) is the number of characters of the Session path string
- **Session path** (string)
- **Save settings to previous** (unsigned int32) determines if the settings are saved to the previous session file before changing it, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.SettingsLoad

Loads the settings from the specified .ini file.

Arguments:

- **Settings file path size** (int) is the number of characters of the Settings file path string
- **Settings file path** (string) is the path of the settings file to load
- **Load session settings** (unsigned int32) automatically loads the current settings from the session file bypassing the settings file path argument, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.SettingsSave

Saves the current settings in the specified .ini file.

Arguments:

- **Settings file path size** (int) is the number of characters of the Settings file path string
- **Settings file path** (string) is the path of the settings file to save
- **Save session settings** (unsigned int32) automatically saves the current settings into the session file bypassing the settings file path argument, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.LayoutLoad

Loads a layout from the specified .ini file.

Arguments:

- **Layout file path size** (int) is the number of characters of the layout file path string
- **Layout file path** (string) is the path of the layout file to load
- **Load session layout** (unsigned int32) automatically loads the layout from the session file bypassing the layout file path argument, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.LayoutSave

Saves the current layout in the specified .ini file.

Arguments:

- **Layout file path size** (int) is the number of characters of the layout file path string
- **Layout file path** (string) is the path of the layout file to save
- **Save session layout** (unsigned int32) automatically saves the current layout into the session file bypassing the layout file path argument, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.Lock

Locks the Nanonis software.

Launches the Lock modal window, preventing the user to interact with the Nanonis software until unlocking it manually or through the *Util.UnLock* function.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.UnLock

Unlocks the Nanonis software.

Closes the Lock modal window which prevents the user to interact with the Nanonis software.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

Error described in the Response message>Body section

Util.RTFreqSet

Sets the Real Time controller frequency.

Arguments:

- **RT frequency** (float32) is the Real Time frequency in Hz

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.RTFreqGet

Gets the Real Time controller frequency.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **RT frequency** (float32) is the Real Time frequency in Hz
- **Error** described in the Response message>Body section

Util.AcqPeriodSet

Sets the Acquisition Period (s) in the TCP Receiver.

Arguments:

- **Acquisition Period (s)** (float32)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.AcqPeriodGet

Gets the Acquisition Period (s) in the TCP Receiver.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **Acquisition Period (s)** (float32)
- **Error** described in the Response message>Body section

Util.RTOversamplSet

Sets the Real-time oversampling in the TCP Receiver.

The 24 signals are oversampled on the RT engine before they are sent to the host. The oversampling affects the maximum Spectrum Analyzer frequency and other displays.

Arguments:

- **RT oversampling** (int)

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

Util.RTOversamplGet

Returns the Real-time oversampling in the TCP Receiver.

Arguments: None

Return arguments (if Send response back flag is set to True when sending request message):

- **RT oversampling** (int)
- **Error** described in the Response message>Body section

Util.Quit

Quits the Nanonis software with the option to save settings, layout and signals (same functionality provided by the dialog window that pops-up when quitting the software through the File menu).

Arguments:

- **Use the stored values** (unsigned int32) automatically ignores the rest of the arguments (0=False and 1=True) and saves settings, layout, and signals according to the last time the software quit.
  This configuration is stored in the Main-Options settings.ini file located in the Certificate folder.
- **Settings name size** (int) is the number of characters of the Settings name string
- **Settings name** (string) is the name of the settings file to save when quitting. The list of settings can be found in the Settings section of the Main Options under the File menu.
  If left empty, no settings are saved (unless the argument "Use the stored values" is 1).
- **Layout name size** (int) is the number of characters of the Layout name string
- **Layout name** (string) is the name of the layout file to save when quitting. The list of layouts can be found in the Layouts section of the Main Options under the File menu.
  If left empty, no layout is saved (unless the argument "Use the stored values" is 1).
- **Save signals** (unsigned int32) automatically saves (0=False and 1=True) the signal configuration currently set in the Signals Manager if it has been changed.
  The signals configuration is stored in the Signals settings.ini file located in the Certificate folder.

Return arguments (if Send response back flag is set to True when sending request message):

- **Error** described in the Response message>Body section

File

File.datLoad

Returns the contents (Channel names, Data, and Header) of a .dat file saved with the Nanonis software.

Arguments:

- **File path size** (int) is the number of characters of the File path string
- **File path** (string) is the path of the .dat file to load
- **Read only header** (unsigned int32) defines if only the Header is returned or if also the Channel names & Data are returned, where 0=False and 1=True

Return arguments (if Send response back flag is set to True when sending request message):

- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) is the number of elements of the Channels names string array
- **Channels names** (1D array string) returns the list of channels names. The size of each string item comes right before it as integer 32
- **Data rows** (int) defines the number of rows of the Data array
- **Data columns** (int) defines the number of columns of the Data array
- **Data** (2D array float32) returns the data stored in the file
- **Header rows** (int) defines the number of rows of the Header array
- **Header columns** (int) defines the number of columns of the Header array
- **Header** (2D array string) returns the header line by line and the attribute followed by its value (i.e. Attribute 1->Value->Attribute 2->Value…).
  The size of each string item comes right before it as integer 32
- **Error** described in the Response message>Body section