Nanonis TCP Protocol

TCP Programming Interface

October 2022 (R5.1.12136)

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).

It is also explained the encoded strings in the response message when the *Send response back* flag in the request message is set to True, i.e. received by the client (remote application) from the server (Nanonis software).

The commands are displayed in hexadecimal (one hexadecimal number pair corresponds to one byte):

Read limits of Step Channel 1 in 3D Sweeper

This example reads the limits of Step Channel 1 in the 3D Sweeper module. It uses the function *3DSwp.StpCh1LimitsGet*.

Arguments: None

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

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops
- **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 *3DSwp.StpCh1LimitsGet* padded with zeros to length 32:
  3364 7377 702E 7374 7063 6831 6C69 6D69 7473 6765 7400 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 (=0 because there are no arguments) padded with zeros to length 4:
  0000 0000
- **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 no arguments in the Request message.

Request message					
	Header				Body
	Command name	Body size in bytes	Send response back	Not Used	-
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (2)	Fixed (2)	-
Readable value representation	3DSwp.StpCh1LimitsGet	0	True	-	-
Hex representation of string to be sent over TCP	3364 7377 702E 7374 7063 6831 6C69 6D69 7473 6765 7400 0000 0000 0000 0000 0000	0000 0000	0001	0000	-

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 *3DSwp.StpCh1LimitsGet* padded with zeros to length 32 (like in the request message):
  3364 7377 702E 7374 7063 6831 6C69 6D69 7473 6765 7400 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 (=16 bytes of the arguments *Start*, *Stop*, *Error status*, and *Error size*) padded with zeros to length 4:
  0000 0010
- **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:

- **Start (float32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to -1mm:
  BA83 126F
- **Stop (float32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 2mm:
  3B03 126F
- **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	Start	Stop	Error status	Error size in bytes	Error description
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (4)	4	4	4	4	0
Readable value representation	3DSwp.StpCh1LimitsGet	16	-	-1m	2m	False	0	-
Hex representation of string to be sent over TCP	3364 7377 702E 7374 7063 6831 6C69 6D69 7473 6765 7400 0000 0000 0000 0000 0000	0000 0010	0000 0000	BA83 126F	3B03 126F	0000 0000	0000 0000	-

Change limits of Step Channel 1 in 3D Sweeper

This example changes the limits of Step Channel 1 in the 3D Sweeper module. It uses the function *3DSwp.StpCh1LimitsSet*.

Arguments:

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops

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 *3DSwp.StpCh1LimitsSet* padded with zeros to length 32:
  3364 7377 702E 7374 7063 6831 6C69 6D69 7473 7365 7400 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 *Start* and *Stop*) padded with zeros to length 4:
  0000 0008
- **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:

- **Start (float32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to -3mm:
  BB44 9BA6
- **Stop (float32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 4mm:
  3B83 126F

Request message						
	Header				Body	
	Command name	Body size in bytes	Send response back	Not Used	Start	Stop
Size (Bytes)	Fixed (32)	Fixed (4)	Fixed (2)	Fixed (2)	-3m	4m
Readable value representation	3DSwp.StpCh1LimitsSet	8	True	-	10n	15n
Hex representation of string to be sent over TCP	3364 7377 702E 7374 7063 6831 6C69 6D69 7473 7365 7400 0000 0000 0000 0000 0000	0000 0008	0001	0000	BB44 9BA6	3B83 126F

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 *3DSwp.StpCh1LimitsSet* padded with zeros to length 32 (like in the request message):
  3364 7377 702E 7374 7063 6831 6C69 6D69 7473 7365 7400 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:

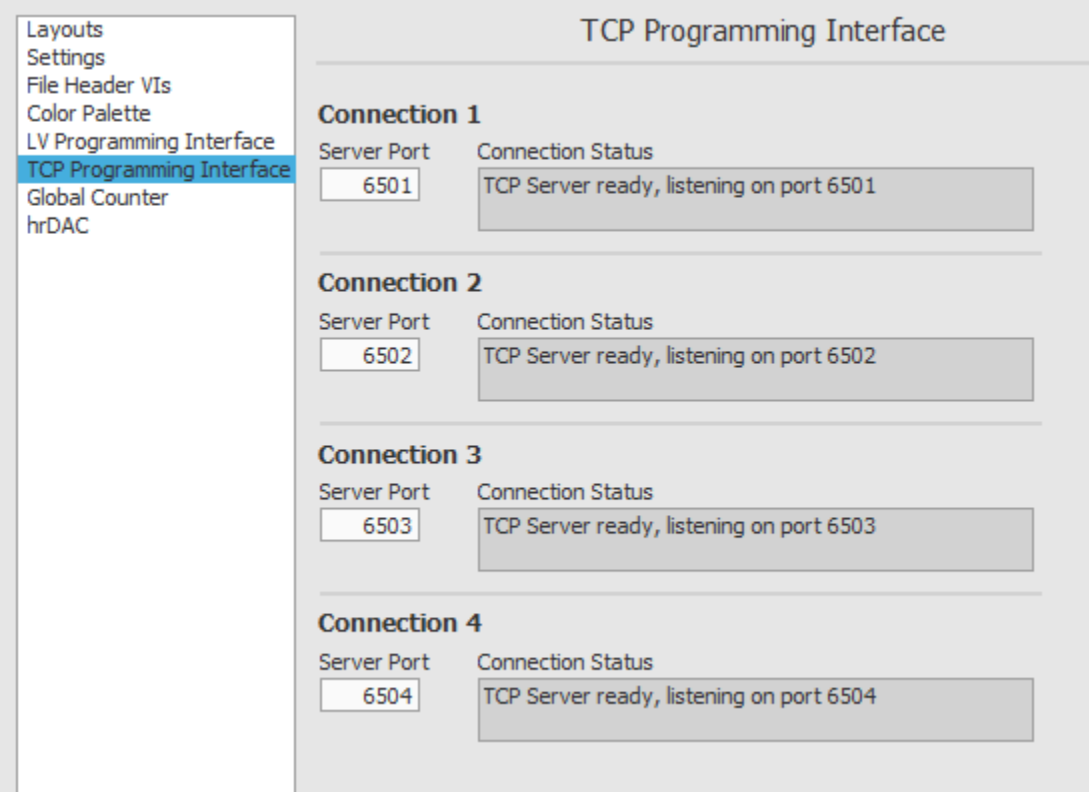
- **Error status (unsigned int32) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to 0 (no error):
  0000 0000
- **Error size (int) (4 bytes)** is the hexadecimal representation of the big-endian encoded string corresponding to the integer size of the error 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	3DSwp.StpCh1LimitsSet	8	-	False	0	-
Hex representation of string to be sent over TCP	3364 7377 702E 7374 7063 6831 6C69 6D69 7473 7365 7400 0000 0000 0000 0000 0000	0000 0008	0000 0000	0000 0000	0000 0000	-

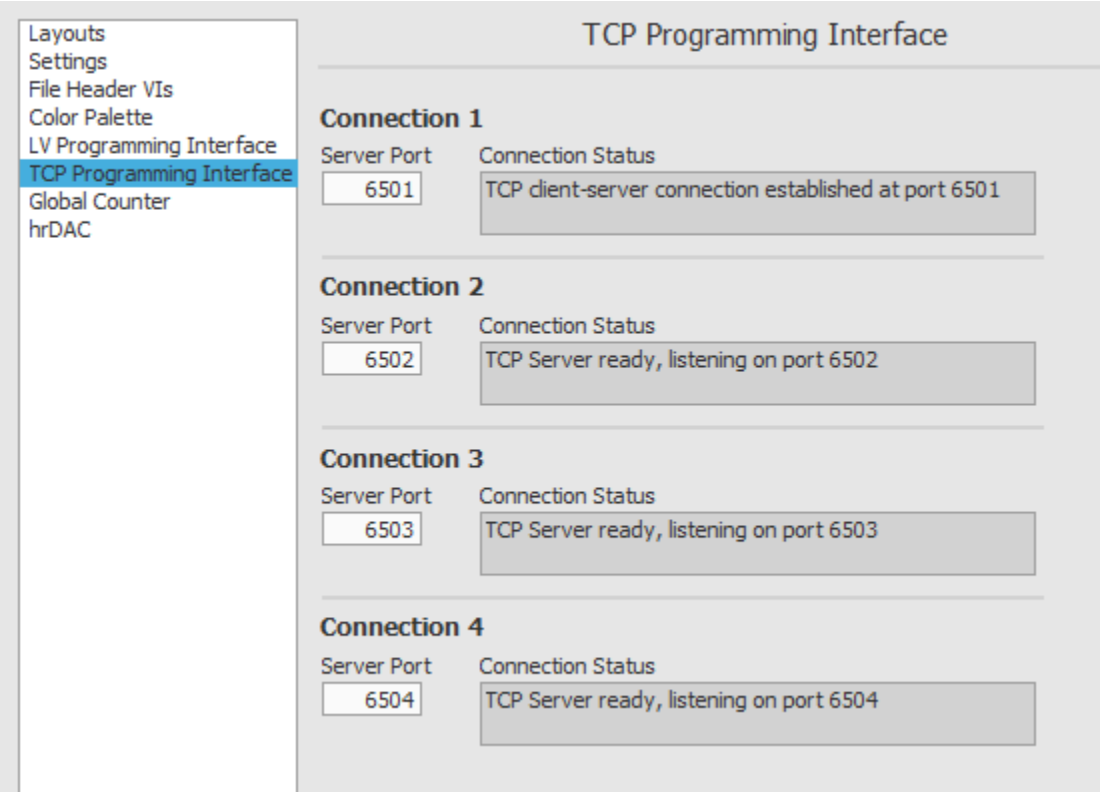
Troubleshooting

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 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

3D Sweeper

3DSwp.AcqChsSet

Sets the list of recorded channels of the 3D 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 are 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

3DSwp.AcqChsGet

Returns the list of recorded channels of the 3D 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 are comprised between 0 and 127, and it corresponds to the full list of signals available in the system
- **Channels names size** (int) is the size in bytes of the channels names array
- **Channels names number** (int) is the number of elements of the channels names array
- **Channels names** (1D array string) returns an array of channel names strings, where each string comes prepended by its size in bytes
- **Error** described in the Response message>Body section

3DSwp.SaveOptionsSet

Sets the saving options of the 3D Sweeper.

Arguments:

- **Series name size** (int) is the size (number of characters) of the series name string
- **Series name** (string) is the base name used for the saved sweeps. If empty string, there is no change
- **Create Date&Time Folder** (int) defines if this feature is active, where -1=no change, 0=Off, 1=On.
  If On, it creates a subfolder within the Session folder whose name is a combination of the basename and current date&time of the sweep, every time a sweep finishes.
- **Comment size** (int) is the size (number of characters) of the comment string
- **Comment** (string) is the comment saved in the header of the files. If empty string, there is no change
- **Modules names size** (int) is the size in bytes of the modules array. These are the modules whose parameters are saved in the header of the files
- **Modules names number** (int) is the number of elements of the modules names array
- **Modules names** (1D array string) is an array of modules names strings, where each string comes prepended by its size in bytes

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

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

3DSwp.SaveOptionsGet

Returns the saving options of the 3D Sweeper.

Arguments: None

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

- **Series name size** (int) is the size of the series name string
- **Series name** (string) is the base name used for the saved sweeps
- **Create Date&Time Folder** (unsigned int32) returns if this feature is active, where 0=Off, 1=On
- **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.
- **Comment size** (int) is the size (number of characters) of the comment string
- **Comment** (string) is the comment saved in the header of the files
- **Modules parameters size** (int) is the size in bytes of the modules parameters array. These are the modules parameters saved in the header of the files
- **Modules parameters number** (int) is the number of elements of the modules parameters array
- **Modules parameters** (1D array string) is an array of modules parameters strings, where each string comes prepended by its size in bytes.
  Each item displays the module name followed by the “>” character followed by the parameter name followed by the “=” character followed by the parameter value
- **Error** described in the Response message>Body section

3DSwp.Start

Starts a sweep in the 3D Sweeper module.

When Send response back is set to True, it returns immediately afterwards.

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

3DSwp.Stop

Stops the sweep in the 3D 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

3DSwp.Open

Opens the 3D 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

3DSwp.StatusGet

Returns the status of the 3D Sweeper.

Arguments: None

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

- **Status** (unsigned int32) is status of the 3D Sweep, where 0=Stopped, 1=Running, 2=Paused
- **Error** described in the Response message>Body section

3DSwp.SwpChSignalSet

Sets the Sweep Channel signal in the 3D Sweeper.

Arguments:

- **Sweep channel index** (int) is the index of the Sweep Channel, where -1 sets the *Unused* option

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

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

3DSwp.SwpChSignalGet

Returns the selected Sweep Channel signal in the 3D Sweeper.

Arguments: None

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

- **Sweep channel index** (int) is the index of the Sweep Channel, where -1 is the *Unused* option
- **Channels names size** (int) is the size in bytes of the Channels names string array
- **Number of channels** (int) defines 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

3DSwp.SwpChLimitsSet

Sets the limits of the Sweep Channel in the 3D Sweeper.

Arguments:

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops

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

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

3DSwp.SwpChLimitsGet

Returns the limits of the Sweep Channel in the 3D Sweeper.

Arguments: None

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

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops
- **Error** described in the Response message>Body section

3DSwp.SwpChPropsSet

Sets the configuration of the Sweep Channel parameters in the 3D Sweeper.

Arguments:

- **Number of points** (int) sets the number of points of the sweep. 0 points means no change
- **Number of sweeps** (int) sets the total number of sweeps. 0 sweeps means no change
- **Backward sweep** (int) defines if the backward sweep is active, where -1=no change, 0=Off, 1=On
- **End of sweep action** (int) defines the behavior of the signal at the end of the sweep, where -1=no change, 0=no action, 1=reset signal to the original value, 2=go to arbitrary value
- **End of sweep arbitrary value** (float32) sets the arbitrary value to go at the end of the sweep if Go to arbitrary value is configured
- **Save all** (int) defines if all the configured sweeps are saved or only the averaged sweep, where -1=no change, 0=Off, 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

3DSwp.SwpChPropsGet

Returns the configuration of the Sweep Channel parameters in the 3D Sweeper.

Arguments: None

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

- **Number of points** (int) returns the number of points of the sweep
- **Number of sweeps** (int) returns the total number of sweeps
- **Backward sweep** (unsigned int32) returns if the backward sweep is active, where 0=Off, 1=On
- **End of sweep action** (unsigned int32) returns the behavior of the signal at the end of the sweep, where 0=no action, 1=reset signal to the original value, 2=go to arbitrary value
- **End of sweep arbitrary value** (float32) returns the arbitrary value to go at the end of the sweep if Go to arbitrary value is configured
- **Save all** (unsigned int32) returns if all the configured sweeps are saved or only the averaged sweep, where 0=Off, 1=On
- **Error** described in the Response message>Body section

3DSwp.SwpChTimingSet

Sets the timing parameters of the Sweep Channel in the 3D Sweeper.

Arguments:

- **Initial settling time (s)** (float32)
- **Settling time (s)** (float32)
- **Integration time (s)** (float32)
- **End settling time (s)** (float32)
- **Maximum slew rate (units/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

3DSwp.SwpChTimingGet

Returns the timing parameters of the Sweep Channel in the 3D Sweeper.

Arguments: None

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

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

3DSwp.SwpChModeSet

Sets the segments mode of the Sweep Channel signal in the 3D Sweeper.

Arguments:

- **Segments mode** (int) is the number of characters of the segments mode string.
  If the segments mode is *Linear*, this value is 6. If the segments mode is *MLS*, this value is 3
- **Segments 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

3DSwp.SwpChModeGet

Returns the segments mode of the Sweep Channel signal in the 3D Sweeper.

Arguments: None

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

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

3DSwp.SwpChMLSSet

Sets the MultiSegment values of the Sweep Channel in the 3D Sweeper.

Arguments:

- **Number of segments** (int) is the total number of segments. It defines the size of the following arrays
- **Segment Start values** (1D array float32) are the start values of the segments of the Sweep Channel
- **Segment Stop values** (1D array float32) are the stop values of the segments of the Sweep Channel
- **Segment Settling times** (1D array float32) are the settling times of the segments in seconds
- **Segment Integration times** (1D array float32) are the integration times of the segments in seconds
- **Segment Number of steps** (1D array int) are the number of steps of each segment
- **Last segment? array** (1D array unsigned int32) defines if the segments are the last one (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

3DSwp.SwpChMLSGet

Returns the MultiSegment values of the Sweep Channel in the 3D Sweeper.

Arguments: None

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

- **Number of segments** (int) is the total number of segments. It defines the size of the following arrays
- **Segment Start values** (1D array float32) are the start values of the segments of the Sweep Channel
- **Segment Stop values** (1D array float32) are the stop values of the segments of the Sweep Channel
- **Segment Settling times** (1D array float32) are the settling times of the segments in seconds
- **Segment Integration times** (1D array float32) are the integration times of the segments in seconds
- **Segment Number of steps** (1D array int) are the number of steps of each segment
- **Last segment? array** (1D array unsigned int32) defines if the segments are the last one (1) or not (0)
- **Error** described in the Response message>Body section

3DSwp.StpCh1SignalSet

Sets the Step Channel 1 signal in the 3D Sweeper.

Arguments:

- **Step channel 1 index** (int) is the index of the Step Channel 1, where -1 sets the *Unused* option

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

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

3DSwp.StpCh1SignalGet

Returns the selected Step Channel 1 signal in the 3D Sweeper.

Arguments: None

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

- **Step channel 1 index** (int) is index of the Step Channel 1, where -1 is the *Unused* option
- **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

3DSwp.StpCh1LimitsSet

Sets the limits of the Step Channel 1 in the 3D Sweeper.

Arguments:

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops

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

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

3DSwp.StpCh1LimitsGet

Returns the limits of the Step Channel 1 in the 3D Sweeper.

Arguments: None

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

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops
- **Error** described in the Response message>Body section

3DSwp.StpCh1PropsSet

Sets the configuration of the Step Channel 1 parameters in the 3D Sweeper.

Arguments:

- **Number of points** (int) sets the number of points of the sweep. 0 points means no change
- **Backward sweep** (int) defines if the backward sweep is active, where -1=no change, 0=Off, 1=On
- **End of sweep action** (int) defines the behavior of the signal at the end of the sweep, where -1=no change, 0=no action, 1=reset signal to the original value, 2=go to arbitrary value
- **End of sweep arbitrary value** (float32) sets the arbitrary value to go at the end of the sweep if Go to arbitrary value is configured

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

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

3DSwp.StpCh1PropsGet

Returns the configuration of the Step Channel 1 parameters in the 3D Sweeper.

Arguments: None

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

- **Number of points** (int) returns the number of points of the sweep
- **Backward sweep** (unsigned int32) returns if the backward sweep is active, where 0=Off, 1=On
- **End of sweep action** (unsigned int32) returns the behavior of the signal at the end of the sweep, where 0=no action, 1=reset signal to the original value, 2=go to arbitrary value
- **End of sweep arbitrary value** (float32) returns the arbitrary value to go at the end of the sweep if Go to arbitrary value is configured
- **Error** described in the Response message>Body section

3DSwp.StpCh1TimingSet

Sets the timing parameters of the Step Channel 1 in the 3D Sweeper.

Arguments:

- **Initial settling time (s)** (float32)
- **End settling time (s)** (float32)
- **Maximum slew rate (units/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

3DSwp.StpCh1TimingGet

Returns the timing parameters of the Step Channel 1 in the 3D Sweeper.

Arguments: None

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

- **Initial settling time (s)** (float32)
- **End settling time (s)** (float32)
- **Maximum slew rate (units/s)** (float32)
- **Error** described in the Response message>Body section

3DSwp.StpCh2SignalSet

Sets the Step Channel 2 signal in the 3D Sweeper.

Arguments:

- **Step Channel 2 name size** (int) is the number of characters of the Step Channel 2 name string
- **Step Channel 2 name** (string) is the name of the signal selected for the Step Channel 2

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

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

3DSwp.StpCh2SignalGet

Returns the selected Step Channel 2 signal in the 3D Sweeper.

Arguments: None

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

- **Step Channel 2 name size** (int) is the number of characters of the Step Channel 2 name string
- **Step Channel 2 name** (string) is the name of the signal selected for the Step Channel 2
- **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

3DSwp.StpCh2LimitsSet

Sets the limits of the Step Channel 2 in the 3D Sweeper.

Arguments:

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops

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

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

3DSwp.StpCh2LimitsGet

Returns the limits of the Step Channel 2 in the 3D Sweeper.

Arguments: None

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

- **Start** (float32) defines the value where the sweep starts
- **Stop** (float32) defines the value where the sweep stops
- **Error** described in the Response message>Body section

3DSwp.StpCh2PropsSet

Sets the configuration of the Step Channel 2 parameters in the 3D Sweeper.

Arguments:

- **Number of points** (int) sets the number of points of the sweep. 0 points means no change
- **Backward sweep** (int) defines if the backward sweep is active, where -1=no change, 0=Off, 1=On
- **End of sweep action** (int) defines the behavior of the signal at the end of the sweep, where -1=no change, 0=no action, 1=reset signal to the original value, 2=go to arbitrary value
- **End of sweep arbitrary value** (float32) sets the arbitrary value to go at the end of the sweep if Go to arbitrary value is configured

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

Error described in the Response message>Body section

3DSwp.StpCh2PropsGet

Returns the configuration of the Step Channel 2 parameters in the 3D Sweeper.

Arguments: None

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

- **Number of points** (int) returns the number of points of the sweep
- **Backward sweep** (unsigned int32) returns if the backward sweep is active, where 0=Off, 1=On
- **End of sweep action** (unsigned int32) returns the behavior of the signal at the end of the sweep, where 0=no action, 1=reset signal to the original value, 2=go to arbitrary value
- **End of sweep arbitrary value** (float32) returns the arbitrary value to go at the end of the sweep if Go to arbitrary value is configured
- **Error** described in the Response message>Body section

3DSwp.StpCh2TimingSet

Sets the timing parameters of the Step Channel 2 in the 3D Sweeper.

Arguments:

- **Initial settling time (s)** (float32)
- **End settling time (s)** (float32)
- **Maximum slew rate (units/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

3DSwp.StpCh2TimingGet

Returns the timing parameters of the Step Channel 2 in the 3D Sweeper.

Arguments: None

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

- **Initial settling time (s)** (float32)
- **End settling time (s)** (float32)
- **Maximum slew rate (units/s)** (float32)
- **Error** described in the Response message>Body section

3DSwp.TimingRowLimitSet

Sets the maximum time (seconds) and channel of the selected row in the Advanced Timing section of the 3D Sweeper.

Arguments:

- **Row index** (int) starting from 0 index
- **Maximum time (seconds)** (float32) defines the ultimate stop condition which is required since certain signal types can result in a target SNR or StdDev never being reached (infinite integration).
  Setting it to the minimum essentially switches off that set (limits to a single RT Cycle). NaN means no change
- **Channel index** (int) defines the channel to which the advanced configuration of the selected row is applied.
  -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

3DSwp.TimingRowLimitGet

Returns the maximum time (seconds) and channel of the selected row in the Advanced Timing section of the 3D Sweeper.

Arguments:

- **Row index** (int) starting from 0 index

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

- **Maximum time (seconds)** (float32) defines the ultimate stop condition which is required since certain signal types can result in a target SNR or StdDev never being reached (infinite integration).
- Setting it to the minimum essentially switches off that set (limits to a single RT Cycle).
- **Channel index** (int) defines the channel to which the advanced configuration of the selected row is applied
- **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

3DSwp.TimingRowMethodsSet

Sets the methods of the selected row in the Advanced Timing section of the 3D Sweeper.

The possible values are -1=no change, 0=None, 1=Time, 2=Standard Deviation, 3=Signal to Noise Ratio.

Arguments:

- **Row index** (int) starting from 0 index
- **Method lower** (int) defines the method in the lower range of the selected row
- **Method middle** (int) defines the method in the middle range of the selected row
- **Method upper** (int) defines the method in the upper range of the selected row
- **Method alternative** (int) defines the method in the alternative range of the selected row

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

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

3DSwp.TimingRowMethodsGet

Returns the methods of the selected row in the Advanced Timing section of the 3D Sweeper.

The possible values are 0=None, 1=Time, 2=Standard Deviation, 3=Signal to Noise Ratio.

Arguments:

- **Row index** (int) starting from 0 index

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

- **Method lower** (int) gets the method in the lower range of the selected row
- **Method middle** (int) gets the method in the middle range of the selected row
- **Method upper** (int) gets the method in the upper range of the selected row
- **Method alternative** (int) gets the method in the alternative range of the selected row
- **Error** described in the Response message>Body section

3DSwp.TimingRowValsSet

Sets the ranges of the selected row in the Advanced Timing section of the 3D Sweeper.

Arguments:

- **Row index** (int) starting from 0 index
- **Middle range: from** (float64) is the upper limit of the lower range of the selected row
- **Lower range: value** (float64) is the value in the lower range of the selected row
- **Middle range: value** (float64) is the value in the middle range of the selected row
- **Middle range: to** (float64) is the lower limit of the upper range of the selected row
- **Upper range: value** (float64) is the value of the upper range of the selected row
- **Alternative range: value** (float64) is the value of the alternative range of the selected row

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

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

3DSwp.TimingRowValsGet

Returns the ranges of the selected row in the Advanced Timing section of the 3D Sweeper.

Arguments:

- **Row index** (int) starting from 0 index

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

- **Middle range: from** (float64) is the upper limit of the lower range of the selected row
- **Lower range: value** (float64) is the value in the lower range of the selected row
- **Middle range: value** (float64) is the value in the middle range of the selected row
- **Middle range: to** (float64) is the lower limit of the upper range of the selected row
- **Upper range: value** (float64) is the value of the upper range of the selected row
- **Alternative range: value** (float64) is the value of the alternative range of the selected row
- **Error** described in the Response message>Body section

3DSwp.TimingEnable

Enables/disables the Advanced Timing in the 3D Sweeper module.

Arguments:

- **Enable** (unsigned int32) where 0=Disable, 1=Enable

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

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

3DSwp.TimingSend

Sends the Advanced Timing configuration of the 3D Sweeper module to the real time 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

3DSwp.FilePathsGet

Returns the list of file paths for the data saved by one single measurement (i.e. 1D and 2D sweeps save one single file, whereas a 3D sweep saves as many files as points configured for Step Channel 2).

Arguments: None

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

- **File paths size** (int) is the size in bytes of the File paths string array
- **Number of file paths** (int) defines the number of elements of the File paths string array
- **File paths** (1D array string) returns the list of file paths. The size of each string item comes right before it as integer 32.
- **Error** described in the Response message>Body section

1D Sweeper

1DSwp.AcqChsSet

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

1DSwp.AcqChsGet

Returns the list of recorded channels of the 1D 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

1DSwp.SwpSignalSet

Sets the Sweep signal in the 1D 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

1DSwp.SwpSignalGet

Returns the selected Sweep signal in the 1D 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

1DSwp.LimitsSet

Sets the limits of the Sweep signal in the 1D 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

1DSwp.LimitsGet

Returns the limits of the Sweep signal in the 1D 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

1DSwp.PropsSet

Sets the configuration of the parameters in the 1D 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

1DSwp.PropsGet

Returns the configuration of the parameters in the 1D 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

1DSwp.Start

Starts the sweep in the 1D 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

1DSwp.Stop

Stops the sweep in the 1D 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

1DSwp.Open

Opens the 1D 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

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 section

Signals.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.

This function links advanced RT signals to Internal 23 and Internal 24.

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 combination of 4 parts that creates a formula. Each part of the formula is a parameter/math operation and a value (which depending on the parameter/math operation is applicable or not).

The possible values for the parameter/math operation of part 1 are:
0=None, 5=Constant, 10=Signal Index

The possible values for the parameter/math operation of parts 2, 3 and 4 are:
0=None, 1=Add Constant, 1=Subtract Constant, 3=Multiply Constant, 4=Divide Constant, 6=Add Signal, 7=Subtract Signal, 8=Multiply Signal, 9=Divide Signal, 11=Exponent, 12=Absolute, 13= Negate, 14= Log

There is no mathematical operator precedence in operation here; the equations are executed in a strict left-to-right (part 1 to part 4) fashion.
This is equivalent to defining the calculations as ((((Part 1) Part 2) Part 3) Part 4).

For example:
The average of Input 1 and Input 2 is defined as “Input 1 + Input 2 / 2”.
The sum of Input 1 plus half of Input 2 is defined as “Input 2 / 2 + Input 1”.
The reciprocal of Input 1 is defined as “1 / Input 1”.

Arguments:

- **Output index** (int) sets the output to be used, where index could be any value from 1 to the number of available outputs
- **Operation 1** (unsigned int16) is the parameter or math operation selected as the 1st part of the configuration formula.
- **Value 1** (float32) is a constant value or signal index, depending on the precedent operation
- **Operation 2** (unsigned int16) is the parameter or math operation selected as the 2nd part of the configuration formula.
- **Value 2** (float32) is a constant value or signal index, depending on the precedent operation
- **Operation 3** (unsigned int16) is the parameter or math operation selected as the 3rd part of the configuration formula.
- **Value 3** (float32) is a constant value or signal index, depending on the precedent operation
- **Operation 4** (unsigned int16) is the parameter or math operation selected as the 4th part of the configuration formula.
- **Value 4** (float32) is a constant value or signal index, depending on the precedent operation

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 combination of 4 parts that creates a formula. Each part of the formula is a parameter/math operation and a value (which depending on the parameter/math operation is applicable or not).

The possible values for the parameter/math operation of part 1 are:
0=None, 5=Constant, 10=Signal Index

The possible values for the parameter/math operation of parts 2, 3 and 4 are:
0=None, 1=Add Constant, 1=Subtract Constant, 3=Multiply Constant, 4=Divide Constant, 6=Add Signal, 7=Subtract Signal, 8=Multiply Signal, 9=Divide Signal, 11=Exponent, 12=Absolute, 13= Negate, 14= Log

There is no mathematical operator precedence in operation here; the equations are executed in a strict left-to-right (part 1 to part 4) fashion.
This is equivalent to defining the calculations as ((((Part 1) Part 2) Part 3) Part 4).

For example:
The average of Input 1 and Input 2 is defined as “Input 1 + Input 2 / 2”.
The sum of Input 1 plus half of Input 2 is defined as “Input 2 / 2 + Input 1”.
The reciprocal of Input 1 is defined as “1 / Input 1”.

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):

- **Operation 1** (unsigned int16) is the parameter or math operation selected as the 1st part of the configuration formula.
- **Value 1** (float32) is a constant value or signal index, depending on the precedent operation
- **Operation 2** (unsigned int16) is the parameter or math operation selected as the 2nd part of the configuration formula.
- **Value 2** (float32) is a constant value or signal index, depending on the precedent operation
- **Operation 3** (unsigned int16) is the parameter or math operation selected as the 3rd part of the configuration formula.
- **Value 3** (float32) is a constant value or signal index, depending on the precedent operation
- **Operation 4** (unsigned int16) is the parameter or math operation selected as the 4th part of the configuration formula.
- **Value 4** (float32) is a constant value or signal index, depending on the precedent operation
- **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

UserOut.SlewRateSet

Sets the slew rate (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
- **Slew Rate** (float64) defines the calibrated slew rate 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.SlewRateGet

Returns the slew rate (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):

- **Slew Rate** (float64) is the calibrated slew rate 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 are 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 indexes are 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 indexes are 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), where -1 means waiting forever

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

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 indexes are 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 indexes are 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

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

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