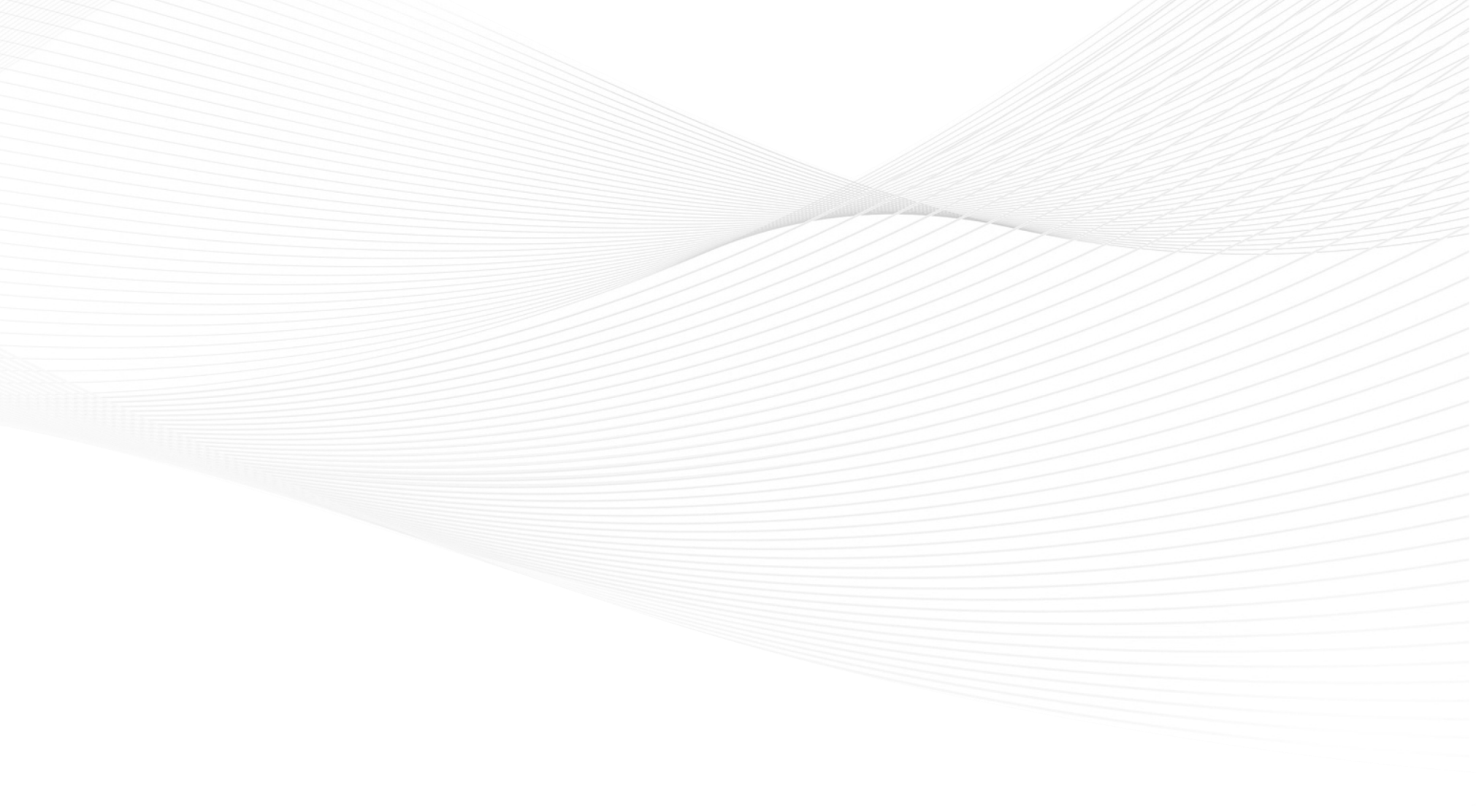
Ein Bild, das Screenshot, Schrift, Logo, Grafiken enthält.

KI-generierte Inhalte können fehlerhaft sein.

Interface Manual

Interferometric Displacement Sensor (IDS)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Manual version: | 1.9.0 | | | | |
| Manual modified: | 05.2025 | | | | |
|  |  |  |  |  |  |
| Firmware version: | 1.9.0 |  |  |  |  |
| API version: | 1.9.0 |  |  |  |  |
| WAVE version: | 1.4 |  |  |  |  |
| attoDiscovery | 1.4.0 |  |  |  |  |

© 2001-2025 attocube systems GmbH. Product and company names listed are trademarks or trade names of their respective companies. Any rights not expressly granted herein are reserved. ATTENTION: Specifications and technical data are subject to change without notice.

© 2001-2025 attocube systems GmbH. Product and company names listed are trademarks or trade names of their respective companies. Any rights not expressly granted herein are reserved. ATTENTION: Specifications and technical data are subject to change without notice.

© 2001-2025 attocube systems GmbH. Product and company names listed are trademarks or trade names of their respective companies. Any rights not expressly granted herein are reserved. ATTENTION: Specifications and technical data are subject to change without notice.

© 2001-2025 attocube systems GmbH. Product and company names listed are trademarks or trade names of their respective companies. Any rights not expressly granted herein are reserved. ATTENTION: Specifications and technical data are subject to change without notice.

Table of Contents

[1. Real-time interfaces 4](#_Toc201653000)

[1.1. Introduction 4](#_Toc201653001)

[1.2. Digital interfaces 5](#_Toc201653002)

[1.2.1. HSSL 5](#_Toc201653003)

[1.2.2. AquadB 7](#_Toc201653004)

[1.3. Analog interfaces 10](#_Toc201653005)

[1.3.1. Sin/Cos (only differential) 10](#_Toc201653006)

[1.4. Anti-Aliasing Filter 12](#_Toc201653007)

[1.5. 3 dB cut-off frequency (Average N filter) 12](#_Toc201653008)

[1.6. Error signal 14](#_Toc201653009)

[1.7. Optional interfaces 16](#_Toc201653010)

[1.7.1. Linear Analog Output (only differential) 16](#_Toc201653011)

[1.7.2. BiSS C 23](#_Toc201653012)

[1.7.2.1. Overview 23](#_Toc201653013)

[1.7.2.2. Interface connector 23](#_Toc201653014)

[1.7.2.3. Interface configuration 24](#_Toc201653015)

[1.7.2.4. Protocol description 26](#_Toc201653016)

[1.8. Real-time interface connector 26](#_Toc201653017)

# Real-time interfaces

## Introduction

|  |  |
| --- | --- |
| Overview  M00 | The IDS provides real-time position signals for each axis. The user can select five different output modes offering different output signal levels. Only one mode and signal level is available at a time. The setting applies to all outputs simultaneously.   * HSSL (binary signal, LVTTL or LVDS) * AquadB (binary signal, LVTTL or LVDS) * Sin/Cos (differential analog signal, LVTTL or LVDS error signal) * Linear Analog Output (differential analog signal, LVTTL or LVDS error signal) – note that this feature is available on request * Deactivated (all pins on GND)   **Note**: Using LVTTL (Low-Voltage Transistor-Transistor-Logic) every signal is transferred single-ended, whereas LVDS (Low-Voltage Differential Signal) twisted pairs are used to obtain the advantages of a differential signal transfer.  **Digital signals:**  LVTTL:   * High voltage level: 3.3 V   LVDS:   * Common voltage: 1.125 to 1.375 V * Differential output voltage: typ. 340 mV at 100 Ω differential termination (we recommend using the 100 Ω differential termination near the receiver) * Positive signals are denoted by X(+) and the negative signals by X(-)   **Analog signals (only differential):**   * Common voltage:   + 1.3 V at 50 Ω termination (single ended)   + 2 V at 1 MΩ termination (single ended) * Differential voltage:   + 1 Vpp +/- 10 % at 50 Ω termination (single ended)   + 1.4 Vpp +/- 10 % at 1MΩ termination (single ended) * Source impedance: 30 Ω * Load current: 7 mA - 20 mA * Positive signals are denoted by X(+) and the negative signals by X(-) |
| Web interface setting | The settings can be configured via the IDS web interface in the Interface tab (see *Figure 1*) or via the available software communication interfaces.  Ein Bild, das Text, Software, Computersymbol, Multimedia-Software enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 1: Setting the output mode in the IDS web interface* |

## Digital interfaces

### HSSL

|  |  |
| --- | --- |
| **Overview** | HSSL (High Speed Serial Link) supports two different measurement modes, the Displacement and the Absolute position mode. HSSL uses a binary serial format based on 1 pm resolution. The HSSL protocol is defined by its resolution, clock time and gap. The clock time represents the bit output rate, and resolution defines the binary bit configuration, which provides the position information. The low resolution tab indicates the starting bit (= least significant bit, LSB) and the high resolution tab sets the final bit (= most significant bit, MSB). An example is shown below. The signal itself starts with the low resolution bit.  The HSSL word is encoded using the two's complement system.  The required gap is a time value in terms of clock time, which separates the different position signals from each other. The maximum clock rate is limited to 25 MHz, equal to 40 ns clock time. For real-time outputs it is recommended to use a DIO or DAQ card, which is compatible with a multiple of 40 ns. The clock period, gap and number of bits can be user adjusted in the web interface or via software communication interfaces (API). Between two sets of position information, synchronization of reader/sender can be gained through a continuous stream of position information. The data is synchronized with the rising edge of the clock signal.  The signals are available with two different signaling standards (only one at a time, please refer to chapter *Introduction*):  • Single-ended with LVTTL levels  • Differential with LVDS levels  A third signal (POS\_ERROR) indicates an error condition e.g. when the position is lost due to laser beam interruption. |
| **Web interface setting** | The HSSL signal in Displacement mode starts with a zero displacement value as soon as the measurement initialization process is done, and the absolute position is shown in the Operation tab. *Figure 2* shows an exemplary HSSL setting in the Interface tab.  Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 2: Defining the HSSL settings in the IDS web interface*  The digital resolution of the HSSL interface is 1 pm. However, for high-speed measurements it might be necessary to limit the number of transmitted bits. |
| **Resolution HSSL Low** | Resolution HSSL Low defines the position resolution of the HSSL interface. It allows specifying the lowest bit of the 48-bit distance value. The position resolution is therefore given by  The lowest bit value for this setting is 0 and the highest is 46. |
| **Resolution HSSL High** | Resolution HSSL High specifies the highest bit to be used of the 48-bit distance value. The lowest bit value for this setting is 1 and the highest is 47. The HSSL word size is therefore given by equation |
| **Period HSSL Clock** | The clock defines the time period at which the serial word bits are outputted (inverse of the bit rate). The clock is programmable as integer multiples of 40 ns. The maximum setting is 10.2 µs. |
| **Period HSSL Gap** | The gap argument is given in HSSL clock periods that are omitted and specifies the gap between the end of a HSSL word and the beginning of the subsequent HSSL word. The lowest gap bit for this setting is 1 and the highest is 255. |
| **Total length of signal and position update rate** | The total HSSL length of the signal is defined by the equation  The achievable position update rate in Hz is the inverse of the HSSL length.  The total HSSL length of signal can be set maximum to 3.09 ms (324 Hz) by 48 bit word size, 255 bit gap and maximum clock of 10.2 µs. The minimum signal length for the 48 bit word size is 2 µs (500 kHz) for minimum gap and clock settings. For further acceleration of the position update rate the word size can be reduced.  *Figure 3* shows an example trace of the HSSL signal with full 48 bit word length and a gap of 3 bits. Here, the first and second line represent the time trace of the data channel and clock channel, respectively. Each word is separated by 3 omitted clock bits.    *Figure 3: Signal pattern of a HSSL transfer* |
| **Displacement mode** | In Displacement mode the displacement data is transferred according to the set HSSL parameters starting with zero displacement value. The maximum transferred position value, for positive and negative displacements, corresponds to the set MSB value (= Resolution HSSL High Bit). As mentioned above, the HSSL word is encoded using the two's complement system. |
| **Absolute position mode** | In Absolute position mode the absolute position data is transferred according to the set HSSL parameters. Please note that the word size should be large enough to fit the measured absolute position. Otherwise, the absolute position will be truncated. |

### AquadB

|  |  |
| --- | --- |
| **Overview** | In contrast to HSSL, the AquadB protocol is an incremental protocol that only transfers information about the change of the measured position. The AquadB interface is used to transmit both the displacement and the direction of the target movement. Using this interface higher bandwidths are achievable compared to the HSSL interface. However, the user continuously needs to keep reading out the position as there is no information about the absolute value of the current position.  The AquadB protocol is defined by the parameters Resolution and Clock. The signal is transferred to channels A and B. The levels’ amplitude is either LVTTL or LVDS (one at a time, please refer to chapter *Introduction*):   * Single-ended with LVTTL levels * Differential with LVDS levels   The Resolution parameter defines the minimum displacement that can be detected and transferred via AqudB interface. This displacement results in a state change of either A or B, depending on the change direction. This is sometimes also referred to X4-Encoding.  *Figure 4* shows the signal pattern of AquadB signals A and B and the resulting current displacement. In an A-leads-B situation, the position has to be incremented and vice versa in a B-leads-A situation the position has to be decremented.    *Figure 4: Signal pattern of a AquadB transfer*  A third signal (POS\_ERROR) indicates an error condition e.g. when the position is lost due to laser beam interruption.  Another way of visualizing the AquadB protocol is to plot channels A and B against each other, which creates a square. In *Figure 5*, a signal change of 90° represents, depending on the direction and displacement of ± Resolution. Going clockwise represents an increment in position (A-leads-B) and going counterclockwise represents a decrement in position (B-leads-A). From this way of visualizing the AquadB protocol it becomes clear that only one channel can change its state at a time.  Ein Bild, das Dunkelheit, Schwarz, Screenshot, Nacht enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 5: Interpretating the AquadB signal lines as a square pattern* |
| **Web interface setting** | *Figure 6* shows an exemplary AquadB settings in the Interface tab. The digital resolution of the AquadB interface is 1 pm. The scaling and the update rate of the A/B outputs are defined by the Resolution and Clock parameters.  Ein Bild, das Text, Screenshot, Schrift, Software enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 6:* *Defining the AquadB settings in the IDS web interface* |
| **Resolution AquadB** | The Resolution AquadB parameter defines the position resolution of the AquadB interface and is user adjustable in the range from 1 pm to 64.93 nm (programmable as 2^*n* steps, where *n* is an integer). We recommend using a high resolution for the first test, afterwards you can decrease the resolution and adjust it to the application. |
| **AquadB clock** | The AquadB clock defines the period at which the quadrature signal increment can be outputted. The fastest possible clock setting is 40 ns (i.e. 25 MHz), equal to the update interval of the absolute distance accumulator. The slowest setting is 10.2 µs (i.e. 98 kHz). The clock is programmable as integer multiples of 40 ns. |
| **Maximum target velocity** | The maximum velocity outputted by the AquadB interface is limited by the interface itself. The maximum velocity supported by the AquadB interface can be calculated by following equation  As a result, when choosing the right Resolution, there is always a trade-off between the maximum velocity that AquadB protocol can transmit and the resolution of the transmitted position. Choosing the right clockis simpler, as there are no negative side-effects by choosing the fastest possible rate the receiving system can handle (e.g. a DAQ-Card or a Counter), other than resulting in larger datasets. |
| **M00** | **Note:** When the target displacement is above the maximum velocity that can be handled by the AquadB output, then the data needs more time to be transmitted, which can be seen as a low pass filtering of the data. |
| **M00** | **Note:** When using the AquadB interface, the signal must be recorded constantly, otherwise the position information is lost. |
| **M00** | **Note:** If the signal to noise ratio is not high enough during the target displacement or if the speed limit is exceeded, the AquadB interface error signal will be raised. |

## Analog interfaces

### Sin/Cos (only differential)

|  |  |
| --- | --- |
| **Overview** | Sin/Cos uses two analog signals in quadrature. For voltage levels please refer to the chapter *Introduction*. The maximum output bandwidth is 25 MHz.  The Sin/Cos interface is used to transmit both the displacement and the direction of the target movement (similar to the AquadB interface). The Resolution defines the smallest displacement which results in an increment or decrement of the signals A and B, depending on the direction. The clock parameter defines the maximum frequency at which the incremental change of the signal can be outputted.  A third signal (POS\_ERROR) indicates an error condition e.g. when the position is lost due to laser beam interruption. The levels’ amplitude is either LVTTL or LVDS (one at a time, please refer to chapter *Introduction*):   * Single-ended with LVTTL levels * Differential with LVDS levels   *Figure 7* shows the Sin/Cos signal pattern of the oscilloscope.    *Figure 7: Sin/Cos signal pattern oscilloscope screenshot: C1 (green) = SIN+ (A+), C2 (pink) = SIN- (A-), F1 (yellow) = Math-function C1 - C2 Signal F1 shows 1 Vpp (right scale bar) Signals C1 and C2 show the common mode voltage (CMV) of 2.05 V (left scale bar).* |
| **Web Interface setting** | *Figure 8* shows an exemplary AquadB settings in the Interface tab. The resolution of the Sin/Cos interface is 1 pm. The scaling and the update rate of the A/B outputs are defined by the Resolution and Clock parameters.  Ein Bild, das Text, Screenshot, Software, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 8: Defining the Sin/Cos settings in the IDS web interface* |
| **Resolution Sin/Cos** | The largest Sin/Cos interface resolution (see *Figure 9*) is 16.8 µm/90° (corresponding to a resolution range of 24 bits given in pm units), the smallest useful resolution is 1024 pm/90°.  Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 9: Sin/Cos signal resolution*  The Sin/Cos output is controlled by a 12-bit D/A converter. Therefore, the resolution for a whole sin/cos period (360°) contains 212 = 4096 steps.  For example, setting the position resolution to 5000 nm/90° of the sin/cos signal leads to 1024 steps with a resolution per step of 4.88 nm. |
| **Period Sin/Cos clock** | The clock period can be adjusted in multiples of 40 ns. The longest possible period is 10.2 µs, the shortest is 40 ns. |

## Anti-Aliasing Filter

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Overview** | Aliasing is a common problem when a signal is under sampled. Depending on the applied interface, the effective output rate is given by the particular parameters. To avoid Aliasing effects in the representing output signal an Anti-Aliasing filter (AAF) for all real-time interfaces (except Biss C) has been implemented. The AAF is based on a 256 elements FIR-Filter. The individual low pass filter cut-off frequencies regarding the Anti-Aliasing are given by the table below:   |  |  | | --- | --- | | *Output mode* |  | | Linear Analog |  | | HSSL |  | | Sin/Cos and AquadB |  |   *Table 1: Low pass filter cut-off frequencies for different rel-time interfaces regarding Anti-Aliasing*  **Note:** For individual setting parameters please refer to the corresponding interface chapter. |
| **Anti-Aliasing filter settings** | The AAF filter parameters can be set via software communication interfaces. The AAF filter window can be selected between:   * Rectangular * Cosine * Cosine^2 * Hamming * Raised Cosine (default)   The attenuation of the filter can be varied between 3 and 30 dB (default = 15 dB). A detailed function description can also be found in the related header file for C and Attocube.chm for C#. |

## 3 dB cut-off frequency (Average N filter)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Overview**  **M00** | The IDS has an integrated FIR low-pass filter, the Average N filter. This filter calculates the output samples (position value) using average from a finite number of input samples (measured position). The averaging time is given by following equation  The minimum setting for N is 0 (= no averaging) and the maximum setting is 24 which corresponds to the averaging time of 0.67 s.  **Note:** This filter will be applied independently on the real-time interface or other interfaces for data transmission.  The averaging time and the 3 dB cut-off frequency depending on the N setting is shown in the table below.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | ***N*** | ***Average time in ns*** | ***Bandwidth in Hz*** | ***Decimate Nd*** | ***3 dB cut-off frequency in Hz*** | | *0* | *40* | *25000000* | *1* | *4000000* | | *1* | *80* | *12500000* | *2* | *1900000* | | *2* | *160* | *6250000* | *4* | *730000* | | *3* | *320* | *3125000* | *8* | *340000* | | *4* | *640* | *1562500* | *16* | *160000* | | *5* | *1280* | *781250* | *32* | *80000* | | *6* | *2560* | *390625* | *64* | *40000* | | *7* | *5120* | *195312,5* | *128* | *20000* | | *8* | *10240* | *97656,25* | *256* | *10000* | | *9* | *20480* | *48828,125* | *512* | *5000* | | *10* | *40960* | *24414,0625* | *1024* | *2500* | | *11* | *81920* | *12207,0313* | *2048* | *1250* | | *12* | *163840* | *6103,51563* | *4096* | *625* | | *13* | *327680* | *3051,75781* | *8192* | *312* | | *14* | *655360* | *1525,87891* | *16384* | *160* | | *15* | *1310720* | *762,939453* | *32768* | *80* | | *16* | *2621440* | *381,469727* | *65536* | *40* | | *17* | *5242880* | *190,734863* | *131072* | *20* | | *18* | *10485760* | *95,3674316* | *262144* | *10* | | *19* | *20971520* | *47,6837158* | *524288* | *5* | | *20* | *41943040* | *23,8418579* | *1048576* | *2,5* | | *21* | *83886080* | *11,920929* | *2097152* | *1.25* | | *22* | *167772160* | *5,96046448* | *4194304* | *0.7* | | *23* | *335544320* | *2,98023224* | *8388608* | *0.35* | | *24* | *671088640* | *1,49011612* | *16777216* | *0.175* |   *Table 2: The averaging time and the 3 dB cut-off frequency depending on the N setting* |
| **Average N filter settings** | The average parameter N can be set via software communication interfaces with the function *setAverageN*.  The web interface allows directly defining the desired 3 dB cut-off frequency, which is more intuitive to use. |

## Error signal

|  |  |
| --- | --- |
| **Overview** | The real-time interface provides an error signal for each individual interferometric axis. The voltage level of the error signal can be either differential LVDS or single ended LVTTL (see chapter *Introduction*). For digital interfaces the error signal has the same voltage level as the data lines. All analog interfaces are differential signals, whereas the error signals can be either LVTTL or LVDS (see *Figure 10*).  Ein Bild, das Text, Screenshot, Software, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 10: LVTTL and LVDS options are available*  The error signal is high (= 1) as long as no measurement is running. During the measurement the error signal is low (= 0) and changes to high (= 1) if an error condition is indicated e.g. when the position is lost due to laser beam interruption. |
| **Error example:**  **signal too high (overload)** | If the interference signal saturates the photo detector, a signal overload is recognized and indicated by the error signal (see *Figure 11*).  Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 11: Overload detection situation via the error signal* |
| **Error example:**  **signal too low** | The error signal indicates the case when the interference signal is less than the minimum threshold (see *Figure 12*).  Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 12: Signal loss detection situation via the error signal* |
| **Error example:**  **beam interruption** | The error signal indicates a beam interruption (see *Figure 13*).  Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 13: Beam interruption detected via the error signal* |

## Optional interfaces

### Linear Analog Output (only differential)

|  |  |
| --- | --- |
| **Overview** | The Linear Analog Output mode (see *Figure 14*) returns a voltage level, which is proportional to the displacement (please refer to chapter *Introduction*). This Linear Analog Output is an optional IDS feature on request, which can be activated with an attocube license file, for further information please refer to the IDS User Manual.  This output supports two different measurement modes, the Displacement and the Vibrometry mode. In Displacement mode, the displacement is directly represented by the given voltage. In Vibrometry mode, the high pass filtering can additionally be applied to the displacement to exclude low frequent disturbances. The maximum bandwidth of the signal is 10 MHz, so frequency analysis can be performed up to 5 MHz with respect to the Nyquist-theorem. The output is quantized by a 12-bit bipolar Digital Analog Converter (DAC). Basically, this interface takes 12 out of 48 HSSL position bits and is carried by a voltage offset. The offset arises from the handling and interfacing of the IDS itself. |
| **Web interface settings** | As well as the other interfaces, the Linear Analog Output appears as an additional option in the drop-down menu of the interface tab output modes.  Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 14: Using the Linear Analog Output mode* |
| **Linear analog resolution and range** | The resolution for the output is given by the equation  Since the outputed voltage level is converted by 12 bit, the minimum resolution of 1 pm can only be achieved for . It is recommended to read out the signal at least with 12 bit (ADC/DAC) to prevent a loss in resolution. For N = 0, the first possible 12 bits out of the whole 48 bit HSSL word are shifted into the DAC. Therefore, the Linear Analog range is given by the equation  The table below and the *Figure 15* indicates all possible results and parameters in terms of Linear Analog configuration.   |  |  |  |  | | --- | --- | --- | --- | | ***N*** | ***Max Range + in pm*** | ***Max Range – in pm*** | ***Resolution in pm*** | | *0* | *2047* | *-2048* | *1* | | *1* | *4095* | *-4096* | *2* | | *2* | *8191* | *-8192* | *4* | | *3* | *16383* | *-16384* | *8* | | *4* | *32767* | *-32768* | *16* | | *5* | *65535* | *-65536* | *32* | | *6* | *131071* | *-131072* | *64* | | *7* | *262143* | *-262144* | *128* | | *8* | *524287* | *-524288* | *256* | | *9* | *1048575* | *-1048576* | *512* | | *10* | *2097151* | *-2097152* | *1024* | | *11* | *4194303* | *-4194304* | *2048* | | *12* | *8388607* | *-8388608* | *4096* | | *13* | *16777215* | *-16777216* | *8192* | | *14* | *33554431* | *-33554432* | *16384* | | *15* | *67108863* | *-67108864* | *32768* | | *16* | *134217727* | *-134217728* | *65536* | | *17* | *268435455* | *-268435456* | *131072* | | *18* | *536870911* | *-536870912* | *262144* | | *19* | *1073741823* | *-1073741824* | *524288* | | *20* | *2147483647* | *-2147483648* | *1048576* | | *21* | *4294967295* | *-4294967296* | *2097152* | | *22* | *8589934591* | *-8589934592* | *4194304* | | *23* | *17179869183* | *-17179869184* | *8388608* | | *24* | *34359738367* | *-34359738368* | *16777216* | | *25* | *68719476735* | *-68719476736* | *33554432* | | *26* | *1.37439E+11* | *-1.37439E+11* | *67108864* | | *27* | *2.74878E+11* | *-2.74878E+11* | *134217728* | | *28* | *5.49756E+11* | *-5.49756E+11* | *268435456* | | *29* | *1.09951E+12* | *-1.09951E+12* | *536870912* | | *30* | *2.19902E+12* | *-2.19902E+12* | *1073741824* | | *31* | *4.39805E+12* | *-4.39805E+12* | *2147483648* | | *32* | *8.79609E+12* | *-8.79609E+12* | *4294967296* | | *33* | *1.75922E+13* | *-1.75922E+13* | *8589934592* | | *34* | *3.51844E+13* | *-3.51844E+13* | *17179869184* |   *Table 3: Signal range and resolution configurations for the Linear Analog Output in accordance with 35 different HSSL configurations*  Ein Bild, das Reihe, parallel, Diagramm, Farbigkeit enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 15: 35 different HSSL configurations that are transferred to the Linear Analog Output* |
| **Determining full range with Test Channel** | The Linear Analog Output feature provides an additional window in the Interface tab to activate a test signal channel (see *Figure 16*).    *Figure 16: Activating the test channel* |
| **M00M00** | By setting the check mark to the box, the differential test signal is transferred to pin 21 and 8 (see Chapter *Real-time interface connector*). The output is based on a rectangular signal to fill and determine the full output range.  **Note:** The test channel can be only activated on the measurement axis 1, as it is firmly connected to the output pin 21 and 8. Other measurement axes provide the measured signal, but have no test channel connection.  **Note**: The test channel on measurement axis 1 can only be activated in the idle state when no measurement or optics alignemnt is running.  Since linear analog is used in differential mode, the levels of pin 21 and 8 can be subtracted to evaluate the full range [V] (see *Figure 17*).  Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 17: Using the test channel when working with differential transfer*  The red line ist built by substracting the level of Pin 8 from Pin 21. The resulting signal reveals a full range of 1,54 V at 1 MΩ termination (single ended) in this example measurement. |
| **Exceeding full range** | If the full range is exceeded by means of displacement, the DAC will lead to a jump between the maximum and the minimum threshold of the analog output (see exemplary *Figure 18*), which is attributed to the characteristics of the DAC itself. Theoretically the full range can also be determined by the size of thoses jumps.  Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 18: Exceeding the Linear Analog Output range*  If the “reset axes” command is executed, the signal of the Linear Analog Output will be centered around the general signal offset (= zero position). From there on the range is limited by the positive and negative maximum from the table above, respectively a certain resolution parameter N. |
|  | Aufbau_disp  *Figure 19: Example application*  In an application example (see *Figure 19*) the target was actively excited by a waveform generator with a triangle voltage, according to the principal test setup in the picture above. The real-time data was acquired via a National Instruments Data Acquisition (DAQ)- Card and analysed with LabVIEW based software. The Linear Analog Resolution was set to N = 10. If the maximum range is not well aligned, the signal cosequently produces signal jumps (see *Figure 20*).  Ein Bild, das Screenshot enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 20: Signal jumps due to exceeding the Linear Analog Output range*  It appeals to execute a reset axes command to center the signal around the offset, when the voltage exceeds its boundaries on first glance. *Figure 21* shows a triangle oscillation of a piezo scanner within the predefined boundaries of the Linear Analog Output with no jumps distorting the displacement information. The range of the output should always be adapted to the expected target movement. In order to not exceed the boundaries it makes sense to apply higher ranges in the first step. Therefore, the measurement needs to be stopped and the range might be changed in terms of a higher maximum range.  Ein Bild, das Screenshot, Rechteck, Text enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 21: Range-conform displacement measurement*  In order to achieve good signal-to-noise ratio, it is recommended to use the whole dynamic output span of the Linear Analog Output, because the DAC only quantizes 4095 steps within the whole range. |
| **Vibrometry mode** | As already mentioned, the Vibrometry setting (see *Figure 22*) contains a high pass filter option to get rid of low frequent influences in signal analysis.  Ein Bild, das Text, Screenshot, Software, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 22: Vibrometry setting*  The filter function is built by a cascaded first order two filter block. The high pass cut-off frequency is given by the equation  All values are listed on the table below. The pass band decreases with a slope of -20 dB/decade.   |  |  | | --- | --- | | ***N*** | ***-3 dB high pass cut-off frequency in Hz*** | | *1* | *800000,000* | | *2* | *400000,000* | | *3* | *200000,000* | | *4* | *100000,00* | | *5* | *50000,000* | | *6* | *25000,000* | | *7* | *12500,000* | | *8* | *6250,000* | | *9* | *3125,000* | | *10* | *1562,500* | | *11* | *781,250* | | *12* | *390,625* | | *13* | *195,313* | | *14* | *97,656* | | *15* | *48,828* | | *16* | *24,414* | | *17* | *12,207* | | *18* | *6,104* | | *19* | *3,052* | | *20* | *1,526* | | *21* | *0,763* | | *22* | *0,381* | | *23* | *0,191* | | *24* | *0,095* |   *Table 4: High pass cut-off frequency in the Linear Analog Vibrometry setting* |
| **M00** | **Note:** The high pass filter uses the same register as the Average N filter (3 dB cut-off frequency). Therefore, in Vibrometry mode the high pass filter setting will be used for the measurement and the Average N filter setting has no impact on the measurement data. |
| **Vibrometry example** | With following example setup it is shown how the high pass filter in the vibrometry mode is working. A schematic of the principal setup can be found in *Figure 23*.  Aufbau_vib  *Figure 23: Vibrometry example setup*  The first screenshot indicates a measurement amplitude of 8.18 mV at an oscillation frequency of 194 Hz in the frequency spectrum. In this case the used filter setting was N = 24 to gain the full spectral information (see *Figure 24*).  Ein Bild, das Text, Screenshot, Display, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 24: HF2LI result for N=24*  In the next step the filter setting was changed to N = 13 which equals the -3 dB cut-off frequency of 195 Hz. Now the measured oscillation is supposed to be attenuated by – 3 dB, which corresponds to a factor of 0.707 (see *Figure 25*).  Ein Bild, das Text, Screenshot, Display, Reihe enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 25: HF2LI result for N=13*  After the cut-off frequency is set, the amplitude in the frequency spectrum reduces to 5.79 mV. |
| **Detector signal during Alignment** | Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 26: Detector signal routed to Linear Analog output during Optical Alignment*  The detector signal (see *Figure 26*) is routed out through the Linear Analog output while the optical alignment is running. The cursors indicate the maximum and minimum load of the output which can be determined by activating the test channel on the measurement axis 1 (pin 21 and 8), first. Comparing this signal to the indicator bar in the web interface shows that the Linear Analog output is much faster. The feedback from adjusting a target with Linear Analog is more precise and freer of any quantization, like it is used in the web interface’s alignment bar algorithm. |

### BiSS C

#### Overview

|  |  |
| --- | --- |
|  | The IDS can optionally be upgraded with the BiSS C interface (digital interface). BiSS C is an open standard introduced by the company iC-Haus. In contrast to the standard real-time interfaces, the BiSS signals are provided via the 14 pin GPIO connector (please refer to the IDS User Manual for the electrical interface). The BiSS C option requires a hardware update, in which the GPIO electronic board is replaced by the BiSS C interface module. All other real-time interfaces are disabled in this mode. The main difference to the HSSL mode is that the clock signal is provided by an external master. The master clock frequency is supported up to a frequency of 10 MHz. This is a local clock domain. There are three such clock domains in the FPGA because each BiSS C device gets its own clock from its master. The IDS uses the mc103z1 IP-core.  The BiSS C position data is encoded using the two's complement system. |

#### Interface connector

|  |  |
| --- | --- |
| **Overview** | The BiSS C signals are provided via the 14 pin GPIO connector. The matching connector is the Honda HDR-E14MAG1+ (see *Figure 27*).  Ein Bild, das Entwurf, Klavier, Diagramm enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 27: Honda HDR-E14MAG1+ connector*  The GPIO interface cable with 14 open ended wires (see *Figure 28*) is provided with the BiSS C feature. Each wire is numbered according to the pinout table in *Figure 29*.    *Figure 28: BiSS C connector cable as part of the BiSS-C IDS delivery scope* |
| **Pinout** | *Figure 29* shows the pinout of the BiSS C interface connector. The signal levels comply with the differential RS422 standard. The clock channel is denoted master (MA) and the data channels are denoted slave out (SLO).  Ein Bild, das Screenshot, Text, Design enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 29: BiSS C connector pinout* |
| **Factory reset** | A factory reset requires the short circuit of the following pins:  DataOut1\_p 🡪 ClockIn1\_p  DataOut1\_n 🡪 ClockIn1\_n |
| **M00** | During the boot procedure of the IDS, a test pattern is sent to the GPIO DataOut1 channel. If this pattern is received at the ClockIn1 channel, the IDS executes a factory reset. In case the IDS is connected to a BiSS C master, this pattern could be interpreted as a position word or an invalid position word by mistake! |
| **M00** | **Note:** depending on the IDS hardware version, the factory reset can be executed via the factory reset button. If there is a factory reset button, the option of the factory reset via pin short circuit doesn’t exist. For more information, please refer to the IDS User Manual. |

#### Interface configuration

|  |  |
| --- | --- |
| **Web interface settings** | The BiSS C interface can be configured in the Interface tab of the web interface (see *Figure 30*). BissC supports two different measurement modes Displacement and Absolute position.  Ein Bild, das Text, Screenshot, Schrift, Software enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 30: Configurating the BiSS C interface* |
| **Resolution BiSS C** | This parameter N defines the position resolution of the BiSS C interface. It sets the lowest bit of the 32 bit distance value out of the full 48 bit position word (similar to HSSL, please refer to chapter *HSSL*). The position resolution is therefore given by the equation  The given resolution, as well as the resulting maximum range can be found in the table below.   |  |  |  | | --- | --- | --- | | ***N*** | ***Max. Range + in pm*** | ***Resolution in pm*** | | *0* | *4294967296* | *1* | | *1* | *8589934592* | *2* | | *2* | *17179869184* | *4* | | *3* | *34359738368* | *8* | | *4* | *68719476736* | *16* | | *5* | *1.37439E+11* | *32* | | *6* | *2.74878E+11* | *34* | | *7* | *5.49756E+11* | *128* | | *8* | *1.09951E+12* | *256* | | *9* | *2.19902E+12* | *512* | | *10* | *4.39805E+12* | *1024* | | *11* | *8.79609E+12* | *2048* | | *12* | *1.75922E+13* | *4096* | | *13* | *3.51844E+13* | *8192* | | *14* | *7.03687E+13* | *16384* | | *15* | *1.40737E+14* | *32768* | | *16* | *2.81475E+14* | *65536* |   *Table 5: Biss C configuration table*  *Figure 31* shows 32 bit section of the full 48-bit position word depending on the setting for N.  Ein Bild, das Reihe, Diagramm, Screenshot, Electric Blue (Farbe) enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 31: 32 bit section depending on N* |
| **M00** | **Note**: Biss C interface has no Anti-Alising filter by itself. Therefore, it is recommended to set the 3 dB cut-off frequency as low pass filter. This setting can be set in the web interface in the Operation tab or via software communication interfaces (please refer to chapter *3 dB cut-off frequency (Average N filter)*. |
| **Displacement mode** | In Displacement mode the displacement data starts with zero displacement value. The maximum transferred positive displacement corresponds to the maximum range according to *Table 5*. The maximum transferred negative displacement is half of the positive range due to the two’s complement encoding. |
| **Absolute position mode** | In Absolute position mode the absolute position data is transferred according to the set BiSS C parameters. Please note that the word size should be large enough to fit the measured absolute position. Otherwise, the absolute position will be truncated. |

#### Protocol description

|  |  |
| --- | --- |
| M00 | The signal levels comply with the RS422 standard. The slaves are linear sensors using the BISS C profile BP3.  The data format of the single cycle data (SCD) is  POS(32) + nE(1) + nW(1) + CRC(6),  where POS(32) denotes the 32 bit position value (the resolution is user adjustable), nE(1) is an error bit, nW(1) is a warning bit, and CRC(6) is the 6 bit cyclic redundancy check, using the BP3 default polynomial 0x43 (start value 0).  **Note**: the warning bit nW (= “not warning”) is unused and is always set to 1. The error bit nE (= “not error”) is set to 1 and changes to 0 if an error condition is indicated e.g. when the position is lost due to laser beam interruption. |

## Real-time interface connector

|  |  |
| --- | --- |
| **Overview** | The IDS has a 26 pin connector for the real-time output signals from HSSL, AquadB, Sin/Cos and Linear Analog Output interfaces (please refer to the IDS User Manual for the electrical interface). *Figure 32* shows the front view of the real-time connector:  Ein Bild, das Text enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 32: Front view of the real-time connector*  A matching connector is the Honda HDR-E26 MAG1+ (see *Figure 33*).  Ein Bild, das Entwurf, Diagramm enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 33: HDR-E26 MAG1+ male connector that can be used to finish a cable*  The real-time interface cable with 26 open ended wires (see *Figure 34*) is provided in the accessories box. The table below shows the pinout of this cable.    *Figure 34: Real-time output cable with open ends (description using example of AquadB LVDS pinout)* |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Pin IDS top view** | **Pin 3M SDR Cable** | **Signal** | | **HSSL LVTTL** | | **HSSL LVDS** | | **AquadB LVTTL** | | **AquadB LVDS** | | **Sin/Cos (error: LVTTL)** | | **Sin/Cos (error: LVDS)** | | **Linear Analog**  **(error: LVTTL)** | **Linear**  **Analog**  **(error: LVDS)** | | **Axis 1** |  | |  | |  | |  | |  | |  | |  | |  |  |  | | 21 | blue (white) | POSITION 1 | | CLK1 | | CLK1(+) | | 1A | | 1A(+) | | 1A(+) | | 1A(+) | | 1(+) | 1(+) | | 8 | white (blue) | - | | CLK1(-) | | - | | 1A(-) | | 1A(-) | | 1A(-) | | 1(-) | 1(-) | | 22 | green (white) | DATA1 | | DATA1(+) | | 1B | | 1B(+) | | 1B(+) | | 1B(+) | | - | - | | 9 | white (green) | - | | DATA1(-) | | - | | 1B(-) | | 1B(-) | | 1B(-) | | - | - | | 23 | yellow (white) | ERROR 1 | | 1E | | 1E(+) | | 1E | | 1E(+) | | 1E | | 1E(+) | | 1E | 1E(+) | | 10 | white (yellow) | - | | 1E(-) | | - | | 1E(-) | | - | | 1E(-) | | - | 1E(-) | | **Axis 2** |  | |  | |  | |  | |  | |  | |  | |  |  |  | | 18 | 1 blue | POSITION 2 | | CLK2 | | CLK2(+) | | 2A | | 2A(+) | | 2A(+) | | 2A(+) | | 2(+) | 2(+) | | 5 | 1 green | - | | CLK2(-) | | - | | 2A(-) | | 2A(-) | | 2A(-) | | 2(-) | 2(-) | | 19 | 3 brown | DATA2 | | DATA2(+) | | 2B | | 2B(+) | | 2B(+) | | 2B(+) | | - | - | | 6 | 3 black | - | | DATA2(-) | | - | | 2B(-) | | 2B(-) | | 2B(-) | | - | - | | 20 | 3 yellow | ERROR 2 | | 2E | | 2E(+) | | 2E | | 2E(+) | | 2E | | 2E(+) | | 2E | 2E(+) | | 7 | 3 green | - | | 2E(-) | | - | | 2E(-) | | - | | 2E(-) | | - | 2E(-) | | **Axis 3** |  | |  | |  | |  | |  | |  | |  | |  |  |  | | 15 | 2 brown | POSITION 3 | | CLK3 | | CLK3(+) | | 3A | | 3A(+) | | 3A(+) | | 3A(+) | | 3(+) | 3(+) | | 2 | 2 black | - | | CLK3(-) | | - | | 3A(-) | | 3A(-) | | 3A(-) | | 3(-) | 3(-) | | 16 | 2 red | DATA3 | | DATA3(+) | | 3B | | 3B(+) | | 3B(+) | | 3B(+) | | - | - | | 3 | 2 green | - | | DATA3(-) | | - | | 3B(-) | | 3B(-) | | 3B(-) | | - | - | | 17 | 1 brown | ERROR 3 | | 3E | | 3E(+) | | 3E | | 3E(+) | | 3E | | 3E(+) | | 3E | 3E(+) | | 4 | 1 black | - | | 3E(-) | | - | | 3E(-) | | - | | 3E(-) | | - | 3E(-) | | **GND** |  | |  | |  | |  | |  | |  | |  | |  |  |  | | 12 | black (red) | | GND | |  | |  | |  | |  | |  | |  |  |  | | 26 | red (white) | |  | |  | |  | |  | |  | |  |  |  | | 1 | white (red) | |  | |  | |  | |  | |  | |  |  |  | | 24 | gold (white) | |  | |  | |  | |  | |  | |  |  |  | | 11 | white (gold) | |  | |  | |  | |  | |  | |  |  |  | | 13 + 14 | inner shield | |  | |  | |  | |  | |  | |  |  |  | |  |  | |  | |  | |  | |  | |  | |  | |  |  |  | | 25 | red (black) | |  | | Internal use only | | | | | | | | | | |  |  |   *Table 6: Pinout for different real-time interfaces* |

# Software communication & interfaces

You can integrate your attocube device into complex automated processes via individual software interfaces. attocube provides APIs for the programming languages C, C#, LabVIEW, Python & Matlab as well as short programming examples to get you started.

The following sections provide information on methods, commands and parameters to be used for calling up device functions with the respective language.

## Introduction

The device provides a set of software communication interfaces offering a broad set of functions and options. These can be used to configure the device as well as to read out data. The following software communication interfaces exist:

• Web interface (please also refer to the User Manual)

• JSON-RPC

• C DLL

• C# DLL

• LabVIEW VIs

• Matlab library

• Python library

Most of the different functions are accessible within every interface. This is why we sort by functions not by interfaces. For every function, we show how the implementation in the particular interface is done. Anyway, first we give a short explanation of the different interfaces:

|  |  |
| --- | --- |
| M00 | **Note:** Part of the conventions mentioned below are specific for the handling of attocube devices and are not necessarily applicable in other contexts. |

### Web Interface

The device runs a built-in webserver. This means that a web interface can be accessed via a common web browser. How to set up the IP and a first connection is given in the User Manual. The web interface is the most straightforward way to communicate with the device and almost full functionality is implemented.

### JSON-RPC

The device allows platform-independent communication using JSON-RPC via TCP/IP and websocket. The JSON commands are the lower level that all other wrappers (e.g. Python or C) use.

### C/C# DLLs

Based on the JSON interface, C/C# libraries are available to implement the functions within C/C#-based coding environments.

### LabVIEW VIs

We offer ready-made VIs to have fast and easy implementation in National Instrument’s LabVIEW environment.

### Matlab library

Based on the JSON interface, we offer ready-made Matlab functions to have a fast and easy implementation in Mathwork’s Matlab environment.

### Python library

Based on the JSON interface, we offer ready-made Python functions to have a fast and easy implementation in Python environment.

## Overview and implementation of the APIs

### JSON-RPC (JRPC2.0)

Your attocube device allows platform-independent communication using JSON-RPC via TCP/IP. When using JSON-RPC, the following conventions apply.

|  |  |
| --- | --- |
| **Transport protocols** | *TCP*  Uses communication port 9090. |
| **Calling a JSON RPC 2.0 method** | A JSON RPC method is called by sending a message to the device.  { "jsonrpc": "2.0", "method": "<method>", "params": [<param [0]>, <param [1]>, …], "id": <call id>, “api”: <api version>}  *<method>*: String defined in chapter 2.2.  *<param x*>: Parameter for the method call. If PARAM is put between two “, it is a string. Without “ it is a number  *<call id>*: A unique id to find the corresponding answer  *<api version>*: A version identifier for backward compatibility, please set to 2 |
| **Example** | Example:  { "jsonrpc": "2.0", "method": "com.attocube.sen.displacement.getAxisDisplacement", "params": [1], "id": 1, “api”: 2} |
| **Receiving a JSON RPC 2.0 response** | The JSON RPC method answer is then sent back as payload to the OK message:  { "jsonrpc": "2.0", "result": [<return values [0]>, <return values [1]>, …],  "id": <call id>}  *<return values [x]*>: The return parameters  *<call id>*: The unique id of the method call |
| **Example** | Example:  { "jsonrpc": "2.0", "results": [0, 4], "id": 1} |
| **Example** | Example:  Communication via PuTTY  Open a Telnet connection with PuTTY.  Ein Bild, das Text, Elektronik, Screenshot, Display enthält.  KI-generierte Inhalte können fehlerhaft sein.  Sending JSON-RPC commands in the command line interface.  Ein Bild, das Text, Software, Multimedia-Software, Screenshot enthält.  KI-generierte Inhalte können fehlerhaft sein. |

### C Library

The C API is provided to integrate the device with all its functionality within your C programs.

|  |  |
| --- | --- |
| **Overview** | The C API contains the following files:  Standard C API:   * attocubeJSON.dll (x64 and x86 versions for a windows environment) * attocubeJSON.lib (x64 and x86 versions for a windows environment) * attocubeJSON.so (x64 and x86 versions for a linux environment) * attocubeJSONCall.h (header file for the general functions) * generatedAPI.h (header file for the device specific functions)   Disovery C API:   * attocube-discovery-dll.dll * attocube discovery-dll.lib * attocube-discovery.h (header file) |
| **Using the .dll’s with different systems** | Note that if you want to use the .dlls within x64 based systems outside the framework of Visual C, you might need to convert the library into a static .a format. |
| **Establishing a connection** | To connect to a device, please use (part of attocubeJSONCall.h):  int ATTOCUBE\_API **Connect**(const char *\*deviceAddress*, int\* *deviceHandle*)  The device handle is the reference to the connection and the device and is input to all other device functions that are following.  To close the connection, please use:  int ATTOCUBE\_API **Disconnect**(int \* *deviceHandle*)  Both functions are included in the API.  For a TCP/IP connection, the port 9090 is used. |
| **Discovering devices within the same network** | The discovery function can be used:  It searches your network for available attocube devices and returns a list of properties. This is done by a SSDP broadcast. If no devices are found, please check the device connection via TCP/IP (e.g. via the websever). The device must be in the same subnet than the requesting PC.  **IMPORTANT NOTE:** These functions are part of an additional discovery .dll – the “attocube discovery dll”, which is also part of the standard delivery content.  Therefore, following functions are available:  int DLL\_EXP AD\_GetDeviceInfos(int index, DeviceInfo\* info)  (Get informations about a discovered device)  void DLL\_EXP AD\_ReleaseInfo();  (Release memory allocated by AD\_Check)  int DLL\_EXP AD\_Check(deviceType)  (Checks discoverable devices on the network and retrieves informations)  **Special data types:**  typedef struct {  char ipAddress[32]; /\*\*< IP address of the device \*/  char modelName[32]; /\*\*< Type of the device \*/  char serialNumber[32]; /\*\*< Serial number of the device \*/  char deviceName[32]; /\*\*< Friendly name assigned to the device \*/  char macAddress[32]; /\*\*< MAC address of the device \*/  bool locked; /\*\*< Device locked by other program \*/  } DeviceInfo;  typedef enum {  IDS,  MOTION\_CTRLER,  BOTH  } deviceType; |

### C# Library

The C# API is provided to integrate the device with all its functionality within your C# programs.

|  |  |
| --- | --- |
| **Overview** | The C# API contains of following files:   * CSharpAPIDLL.dll (compiled as “any” version) * Newtonsoft.Json.dll (compiled as “any” version) * Attocube.chm (helpfile) |
| **Establishing a connection** | To connect to a device, please create an device object  public static Attocube<Device> client = new Attocube<Device>()  where <Device> is e.g. AMC or IDS  The connect function is a property of the Attocube<Device> class.  public void **Connect**(string *ipAddress*, int *port*)  The device handle is the reference to the connection to the device and is input to all other device functions that are following.  To close the connection, please use:  public void **Disconnect**()  Both functions are included in the API and part of the device class (so initialize a member of the class first).  For a TCP/IP connection, use the port 9090. |
| **Discovering devices within the same network** | The discovery function can be used:  It searches your network for available devices and returns a list of properties. This is done by an SSDP broadcast. If no devices are found, please check the device connection via TCP/IP (e.g. via the websever) or make sure that the device is in the same subnet than your PC  **IMPORTANT NOTE:** These functions are part of an additional discovery .dll – the “attocube discovery dll”, which is also part of the standard delivery content.  Therefore, following function is available:  public DiscoveryData[] **Check**()  **Special data type:**  Type: DiscoveryData  Class for handling the data of devices discovered using the discovery protocol |

### LabVIEW

The LabVIEW API is provided to integrate the device with all its functionality within your LabVIEW VIs.

|  |  |
| --- | --- |
| **Overview** | The LabVIEW API contains a LabVIEW project which contains all single function VIs and a master example VI that uses almost all functions available and that mimics the web interfaces UI for easy navigation. |
| **Implementation** | To reduce complexity and external dependencies all TCP/IP calls have been implemented with native LabVIEW TCP/IP elements. For older LabVIEW Versions where there is no native TCP/IP support, DLL based VIs have been created taking care of the TCP/IP communication.  The folders “DLLHandler” or “TCPHandler” contain the respective SubVIs handling the messaging and communication with your attocube device, which are used within all low-level VIs. Those should not be modified or used directly. |
| **High-level Wrapper VIs** | For most functions that do have both a set and a get method a higher level “controlMethod” VI has been created to reduce the number of VIs and also be as backwards compatible as possible to the older motion controller series ECC100 and ANC350. Some additional high level VIs like the deviceInfo VI have been created where multiple low-level VIs are combined into one VI and all In- and Outputs are bundled into clusters.  In case you still want to use those low-level VIs instead, they can be found inside folders that contain the word “SubVIs”. For code cleanliness it is not recommended to use those. However to keep the documentation consistent over all programming languages, only the low-level methods are documented (see chapter 3). |
| **Establishing a connection** | To connect to an device, please use the connect VI  The output is the reference to the connection to the device and is needed as an input to all other device functions that are following.  To close the connection, please use the Close VI  Both VIs are included in the API. |
| **Discovering devices within the same network** | The discovery function can be used:  It searches your network for available devices and returns a list of properties. This is done by an SSDP broadcast. If no devices are found, please check the device connection via TCP/IP (e.g. via the websever) or make sure that your device is in the same subnet than your PC.  **IMPORTANT NOTE:** These functions are part of an external DLL – the “attocube discovery dll”, which is also part of the standard delivery content.  Therefore, the “Check.vi” is available. |

### Matlab

The Matlab API is provided to integrate the device with all its functionality within your Matlab scripts.

|  |  |
| --- | --- |
| **Establishing a connection** | To connect to an device, please use:  [success, DeviceHandle] = **connect**(*IPAddress*, *port*)  The device handle is the reference to the connection to the device and is input to all other device functions that are following.  To close the connection, please use:  [success] = **disconnect**(*DeviceHandle*)  Both functions are included in the API.  For a TCP/IP connection, use the port 9090 |

### Python

The Python API is provided to integrate the device with all its functionality within your Python programs.

|  |  |
| --- | --- |
| **Overview** | The Python API contains a folder with domain specific files.  To have access to the python functions, please import the Device within your python script:  import **<Device>**  **where <Device> is your DeviceType, e.g. AMC or IDS** |
| **Establishing a connection** | To connect to an device, please use:  device = <Device>.Device(*ipAdress*)  device.**connect()**  The device handle is the reference to the connection to the device and is input to all other device functions that are following.  To close the connection, please use:  device .**close()**  Both functions are included in the API and part of the device class (so initialize a member of the class first).  For a TCP/IP connection, the port 9090 is used per default. |
| **Discovering devices within the same network** | The discovery function can be used:  It searches your network for available devices and returns a list of properties. This is done by an SSDP broadcast. If no devices are found, please check the device connection via TCP/IP (e.g. via the websever) or make sure that the device is in the same subnet than your PC  Therefore, following module function is available:  <Device>.**discover()**  This returns a dictionary containing all found devices combined with their device information. |

## Error handling

### C error handling

|  |  |
| --- | --- |
| **Introduction** | The error handling in C is realized with return values, directly returned by each function. On success, the function yields zero. Negative error numbers indicate an error within the DLL itself and are specified in the header File ( attocubeJSONCall.h). Positive Error numbers indicate an error in the Device, and can be translated to readable strings with system\_errorNumberToString() |
| **Example** | int value;  int ret = Device\_Function(device, &value);  if( ret == ATTOCUBE\_Ok) {  //success  }  else if (res < ATTOCUBE\_Ok) {  //DLL Error, e.g. not connected  }  else if (res > ATTOCUBE\_Ok) {  //Device Error  char errorNameBuf[BUF\_SIZE];  system\_errorNumberToString(device, 0, ret, errorNameBuf, BUF\_SIZE);  printf("%s", errorNameBuf)  } |

### C# error handling

|  |  |
| --- | --- |
| **Introduction** | The error handling in C# is realized with exceptions not by error numbers. Errors can be caught using a try-catch statement. To include the device specific exceptions, the catch clause will need the AttocubeAPIException as argument. An example code is shown below. |
| **Example** | public class **AttocubeApiException:**  **ApplicationException**  Example: Exception handling  try  {  attoDevice.<Method>();  }  catch (AttocubeApiException e)  {  int err = e.ErrorCode; // passes the errorcode of type int to "err"  string errmsg = attoDevice .ErrorNumberToString(0, err);  // converts "err" into the corresponding error message and passes it to "errmsg" of type string  } |

### Python error handling

|  |  |
| --- | --- |
| **Introduction** | The error handling in Python is realized with exceptions not by error numbers. Errors can be caught using a try-except statement. To include the specific exceptions, the catch clause will need the AttoException as argument. An example code is shown below. |
| **Example** | #example for exception handling  from ACS import AttoException  try:  print(dev.<Method> ()) #OK  except AttoException as e:  print(e) |

### LabVIEW error handling

|  |  |
| --- | --- |
| **Introduction** | The error handling in LabVIEW is realized by using an error message variable which should be looped through all VIs. Therefore, every VI provides an error in and error out connector. Note that we divide the error variable in error messages and “real” errors, which are treated differently. Error messages have a positive error number value combined with the Boolean error status set on inactive (Boolean value on false – visualized by a green hook icon), whereas “real” errors have a negative error number values with an active (Boolean value on true – visualized by a red cross icon) error status. Error messages do not influence the execution of the following VIs, they are used to inform the user. “Real” errors hinder the execution of following VIs, they are meant to stop the program. Examples are shown below: |
| **Example** | Example for an error message. An indicator is used to visualize the error message in the front panel (see *Figure 35*). Facing a positive error code value, an error message is indicated. The status is still set on green meaning no real error available. Error messages are intended to inform the user or other functions about certain cases. If an error message is inputted in the following VI, this VI still is executed.  Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 35: LabVIEW example for an error message* |
|  | Example for a “real” error. Facing a negative error code value, a “real” error is indicated (see *Figure 36*). Therefore, also the status is set on red, which means that the error is active. When an active error is inputted in the next VI, this VI will not execute its function.  Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein.  *Figure 36: LabVIEW example for a "real" error* |

## Streaming

The streaming functionality is available as an optional feature and allows high-speed data access through a software interface. For additional information about feature activation please refer to the User Manual. With streaming, measurement data can be retrieved at defined sample rates of up to 1 MHz, without requiring a dedicated hardware connection. This makes it the simplest method for accessing high-speed measurement data. The data can be accessed either through the API, which is available in C, C#, and Python, or through the WAVE software, which provides a graphical user interface for the streaming feature.

### Streaming C

|  |  |
| --- | --- |
| **Overview** | The C streaming API contains following files:   * SEN.Stream.dll * SEN.stream.h * README.txt * streaming\_example\_console.cpp * streaming\_example\_recording.cpp |
| **Implementation** | From the README.txt:  The basic steps required to get the dll running with your C application  Step 1: Include the DLL's Header  In your C/C++ project, you need to include the header for the DLL. Ensure the header file can be found in the project file path.  e.g. include "SEN.Stream.h"  Step 2: Link the DLL  You need to link your C++ project with the SEN.Stream.dll. If you are using an import library (like SEN.Stream.lib), ensure it is linked correctly in your project settings.  - In Visual Studio:  Open the Project Properties.  Go to Configuration Properties > Linker > Input.  Add SEN.Stream.lib to Additional Dependencies.  - In GCC/MinGW:  g++ -o MyApp MyApp.cpp -L. -lSEN.Stream  Step 3: Ensure DLL Availability  Make sure that SEN.Stream.dll is accessible when running your application:  Place SEN.Stream.dll in the same directory as the executable.  Or add the directory containing SEN.Stream.dll to the system’s PATH environment variable. |
| **Examples** | Two examples are given to verify the correct implementation of the library and to help with the understanding of the functions.  **streaming\_example\_console.cpp**  The program streams and decodes 1023 position values with a streaming rate of 100 kHz from axis 1, 2 and 3. Note that error buffers should contain values of 0s or 1s after successful decoding. Error flag 1 indicates that error detected on the corresponding position value whereas 0 indicates no error detected.  **streaming\_example\_recording.cpp**  The program loops the data recording until at least 5000 (note that only packages of max. 1023 values are received) samples were streamed and decoded. The received data is written into a .txt file. Furthermore, the data gets recorded into an .aws file using the StartStreamRecording function. |

### Streaming C#

|  |  |
| --- | --- |
| **Overview** | The C# streaming API is wrapping the C streaming API and contains following files:   * SEN.Stream.dll (the actual C dll that gets wrapped) * SEN.StreamDLLWrapper.dll: the C# wrapper * README.txt * streaming\_example.cs |
| **Implementation** | From the README.txt:  Step 1: Placing the two DLLs:  The C# API Wrapper DLL (SEN.StreamDLLWrapper.dll): The C# API wrapper DLL is a managed assembly, which means it is part of your .NET project. You need to reference it in your project so that you can use its functionality to interact with the C DLL.  You don't need to place the C# wrapper DLL directly in the executable folder at build time. Instead, you should ensure that it's referenced by your .NET project. When you build the project, Visual Studio (or the .NET CLI) will output this DLL to your project’s output directory, typically either bin\Debug or bin\Release.  The C\_DLL (SEN.Stream.dll) is the original Streaming library that the C# API wrapper (SEN.StreamDLLWrapper.dll) interacts with using P/Invoke. This DLL needs to be located in the platform-specific directory where the application can find and load it during runtime.  The C DLL should be placed in the same directory as the application’s executable (or in a platform-specific folder within the application's directory). This ensures that the application can find the unmanaged library without issues. Missing or incorrectly placing the SEN.Stream.dll can result in runtime errors.  example target location: YourProject\bin\Debug\SEN.Stream.dll  Step 2: Referencing the C# API Wrapper DLL (SEN.StreamDLLWrapper):  When developing in Visual Studio (or using the .NET CLI), you need to reference the C# API wrapper DLL in your project. You’ll need to manually add a reference.  Right-click on your project in Solution Explorer > Add > Reference > Browse and select the C# API wrapper(SEN.StreamDLLWrapper.dll) file.  Step 3: Usage:  Once the DLL is referenced, you can use its methods just like any other .NET class. You’ll call the methods exposed in your wrapper, which in turn will call the unmanaged C functions from the C DLL.  Example of calling a method from the C# API wrapper:  using SEN.StreamDLLWrapper;  class Program  {  static void Main()  {  var wrapper = new StreamDLLWrapper();  wrapper.Open\_Stream("192.168.1.1", true, 10, channelMask); // Calling a function that interacts with the C DLL  }  } |
| **Example** | One example is given to verify the correct implementation of the library and to help with the understanding of the functions.  **streaming\_example.cs**  This is an exemplary implementation of the C# API Wrapper for the IDS streaming feature. The program uses the SEN.StreamDLLWrapper which interact with the SEN.Stream native C library to stream and decode atleast 5000 position values with a streaming rate of 100 kHz from axis 1, 2 and 3. Note that error buffers should contain values of 0s or 1s after successful decoding. Error flag 1 indicates that error detected on the corresponding position value whereas 0 indicates no error detected. |

### Streaming Python

|  |  |
| --- | --- |
| **Overview** | The Python streaming API is wrapping the C streaming API and contains the following files:   * SEN.Stream.dll * IDS.stream python module * README.txt * streaming\_example\_console.py * streaming\_example\_backgroundStreaming.py |
| **Implementation** | From the README.txt:  The SEN.Stream.dll provides functionality to read and decode streams of positional data from attocube interferometer devices.  The dll contains functions that take input buffers, establish the initial http connection to the attocube interferometer device for streaming, process streams, and write decoded data.  The Python wrapper contains modules to call and use functions from the SEN.Stream.dll. This is typically done using Python’s built-in ctypes libraries, which enable Python to load the DLL, define the C function signatures, and pass data between Python and C correctly.  Here's what to know when using SEN.Stream.dll through the python api wrapper  Main modules:  IDS.stream - the IDS.stream python module provides the interface to the core streaming functions of the SEN.Streaming.dll. It loads the DLL, sets up correct function signatures, wraps and exposes the C functions as simpler Python-callable methods.  available methods in the IDS.stream module:  open: Open Stream connection to device  close: Close stream and stop recording if necessary  read: Read and decode position data in buffer, uses the methods readRaw and decodeBuffer; requires 1 param - bufferSize(int) to hold each sample data of axes, returns the decoded buffer.  readRaw: Read raw position data in buffer(Undecoded data); requires 1 param - bufferSize(int), returns buffer of raw data  decodeBuffer: Decode the raw position data buffer to positions in pm; requires 1 param(buffer containing the raw read data), returns 6 values - decodedBytes(int), 3 buffers holdong positional data and 3 buffers holding error flags  startRecording: Starts stream recording to file(aws file)  stopRecording: Stop stream recording to file  These methods can be called on an instance of the Stream object. The Stream is instantiated with device  parameters as shown below:  e.g: Stream(deviceIp, isMaster, intervalInMicroseconds, axis0=ax0, axis1=ax1, axis2=ax2)  >> Stream("192.168.1.1", true, 10, axis0=true, axis1=true, axis2=true)  IDS.streaming - python's multiprocessiong module |
| **Examples** | Two examples are given to verify the correct implementation of the library and to help with the understanding of the functions.  **streaming\_example\_console.py**  This script connects to an IDS device via IP and streams high-frequency position data for up to three axes. It initializes the stream, reads raw measurement data into a buffer, decodes it, and prints the first 20 samples per axis along with associated error bits. The Stream class is utilized to manage the connection and handle the data flow efficiently.  **streaming\_example\_backgroundStreaming.py**  This script connects to an IDS device via IP, initializes it (if necessary), and records background streaming data for all three axes over 5 seconds. The data is saved to a .aws file, then loaded and visualized using Plotly. The script converts position data from picometers to nanometers and plots it over time with appropriate axis formatting. It provides a quick way to test streaming functionality and visualize recorded displacement data. |

### Streaming error handling

|  |  |
| --- | --- |
| **Introduction** | The streaming error handling can be revised using the return values of each function. Functions like ReadStream and DecodeStream that have **int** return types will return **0** when unsuccessful whereas functions like StartStreamRecording and StopStreamRecording that have **bool** return types may return **False** for failures. The OpenStream function in particular returns void\* or IntPtr(c#), an unsuccessful connection may return **null** or **0.** |
| **Example** | //example in C  auto stream = OpenStream("192.168.1.1", true, 10, channelMask);  if (!stream)  {  // something went wrong  return;  }  //example in c#  IntPtr stream = wrapper.Open\_Stream("192.168.1.1", true, 10, channelMask);  if (stream == IntPtr.Zero)  {  Debug.WriteLine("\*\*Failed to open stream\*\*");  return;  } |
| **Error bit streaming** | To indicate measurement errors an error bit is included in the data stream. The error bit indicates the presence or absence of a measurement error for each measurement axis and is included in the data packet. Errors can only be detected with the resolution of the data packet size, but for most applications, this will be sufficient. The approach is conservative: the error bit is set if even one invalid position is detected within the packet, ensuring more data is marked as invalid rather than less. The error bit can be obtained using the decodeStream function. To identify the specific error, it is recommended to use the getSystemError function, which receives an error number. With the errorNumberToString function this error number can then be mapped to the corresponding error description and recommendation for troubleshooting. Those two functions are part of the general device API. |

## Overview about state machine

In the following a state machine is presented. The pilot laser can be activated parallel to the alignment or the measurement mode.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Description of API-Functions

## About

|  |
| --- |
| **getInstalledPackages** Get list of packages installed on the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| value\_string1 | string: comma separated list of packages |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.about.getInstalledPackages |
| params: [] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_about\_getInstalledPackages**(int deviceHandle, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].about.getInstalledPackages**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_about\_getInstalledPackages**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**About\_GetInstalledPackages**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getInstalledPackages.vi |

|  |
| --- |
| **getPackageLicense** Get the license for a specific package. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | pckg | string: Package name |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| value\_string1 | string: license for this package |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.about.getPackageLicense |
| params: [pckg] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_about\_getPackageLicense**(int deviceHandle, const char\* pckg, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].about.getPackageLicense**(pckg) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_about\_getPackageLicense**(pckg) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**About\_GetPackageLicense**(string pckg) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPackageLicense.vi |

## Adjustment

|  |
| --- |
| **getAdjustmentEnabled** This function can be used to see if the adjustment is running. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.adjustment.getAdjustmentEnabled |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_adjustment\_getAdjustmentEnabled**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].adjustment.getAdjustmentEnabled**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **SEN\_adjustment\_getAdjustmentEnabled**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Adjustment\_GetAdjustmentEnabled**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAdjustmentEnabled.vi |

|  |
| --- |
| **getContrastInPermille** This function can be used to monitor the alignment contrast (peak-to-peak of the basic interference signal amplitude) and the baseline (its offset) during alignment mode. It is used to monitor and optimize the optical alignment such that it allows the successful initialization of a measurement. To monitor the optical contrast during a running measurement, please use the getAxisSignalQuality function.  Important: The baseline and contrast need to be added, otherwise an overload can occur without being recognized. Example: baseline 15 permille and contrast 850 permille. Just from the contrast it looks good. Adding the baseline the overload is visible. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| contrast | contrast of the base band signal in permille |
| baseline | offset of the contrast measurement in permille |
| mixcontrast | lower contrast measurement when measuring a mixed contrast (indicated by error code) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.adjustment.getContrastInPermille |
| params: [axis] |
| Result: [warningNo, contrast, baseline, mixcontrast] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_adjustment\_getContrastInPermille**(int deviceHandle, int axis, int\* warningNo, int\* contrast, int\* baseline, int\* mixcontrast) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, contrast, baseline, mixcontrast = **[dev].adjustment.getContrastInPermille**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, contrast, baseline, mixcontrast] = **SEN\_adjustment\_getContrastInPermille**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,int,int,int> value = [Device].**Adjustment\_GetContrastInPermille**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getContrastInPermille.vi |

## Axis

|  |
| --- |
| **apply** Applies new axis settings. Necessary after JSON set commands. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.apply |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_apply**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].axis.apply**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_axis\_apply**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Axis\_Apply**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| apply.vi |

|  |
| --- |
| **discard** Discards new axis settings. Necessary after JSON set commands instead of apply() in case of failure. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.discard |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_discard**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].axis.discard**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_axis\_discard**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Axis\_Discard**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| discard.vi |

|  |
| --- |
| **getAutoMasterAxis** Gets status of the automatic configuration of the master axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| automatic | true = automatic; false = manual (default: true) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.getAutoMasterAxis |
| params: [] |
| Result: [errNo, automatic] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_getAutoMasterAxis**(int deviceHandle, bool\* automatic) |

|  |
| --- |
| **Python** |

|  |
| --- |
| automatic = **[dev].axis.getAutoMasterAxis**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [automatic] = **SEN\_axis\_getAutoMasterAxis**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Axis\_GetAutoMasterAxis**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAutoMasterAxis.vi |

|  |
| --- |
| **getMasterAxis** Returns the master axis (for more information, please refer to the device user manual).  Please note that the master axis is not updated during the automatic configuration of the master axis. The getMasterAxis returns the last set value of the setMasterAxis function. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| axis | [0|1|2] |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.getMasterAxis |
| params: [] |
| Result: [errNo, axis] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_getMasterAxis**(int deviceHandle, int\* axis) |

|  |
| --- |
| **Python** |

|  |
| --- |
| axis = **[dev].axis.getMasterAxis**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [axis] = **SEN\_axis\_getMasterAxis**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Axis\_GetMasterAxis**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getMasterAxis.vi |

|  |
| --- |
| **getPassMode** Reads out the current pass mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| mode | 0 = single; pass 1 = dual pass |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.getPassMode |
| params: [] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_getPassMode**(int deviceHandle, int\* mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].axis.getPassMode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **SEN\_axis\_getPassMode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Axis\_GetPassMode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPassMode.vi |

|  |
| --- |
| **setAutoMasterAxis** Enables the automatic configuration of the master axis during measurement initialization. The master axis chosen is the axis furthest away. This function is only available in the "system idle" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | automatic | true = automatic; false = manual |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.setAutoMasterAxis |
| params: [automatic] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_setAutoMasterAxis**(int deviceHandle, bool automatic) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].axis.setAutoMasterAxis**(automatic) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_axis\_setAutoMasterAxis**(automatic) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Axis\_SetAutoMasterAxis**(bool automatic) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setAutoMasterAxis.vi |

|  |
| --- |
| **setMasterAxis** Sets the master axis (for more information, please refer to the device user manual).  This function is only available in the "system idle" state (please refer to the getCurrentMode function) and can be applied only if the automatic configuration of the master axis is deactivated (please refer to the getAutoMasterAxis function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.setMasterAxis |
| params: [axis] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_setMasterAxis**(int deviceHandle, int axis) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].axis.setMasterAxis**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_axis\_setMasterAxis**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Axis\_SetMasterAxis**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setMasterAxis.vi |

|  |
| --- |
| **setPassMode** Sets the desired pass mode. Effectively this mode defines, if the correction factor of two (necessary for measurements in optical dual pass configuration) is applied. This function is only available in the "system idle" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 0 = single pass; 1 = dual pass |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.axis.setPassMode |
| params: [mode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_axis\_setPassMode**(int deviceHandle, int mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].axis.setPassMode**(mode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_axis\_setPassMode**(mode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Axis\_SetPassMode**(int mode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setPassMode.vi |

## Displacement

|  |
| --- |
| **getAbsolutePosition** This function outputs the sum of the static absolute position register value and the continuously updated displacement register value for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| position | absolute position of the axis in pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getAbsolutePosition |
| params: [axis] |
| Result: [warningNo, position] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getAbsolutePosition**(int deviceHandle, int axis, int\* warningNo, double\* position) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, position = **[dev].displacement.getAbsolutePosition**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, position] = **SEN\_displacement\_getAbsolutePosition**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,double> value = [Device].**Displacement\_GetAbsolutePosition**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAbsolutePosition.vi |

|  |
| --- |
| **getAbsolutePositions** This function outputs the sum of the static absolute position register value and the continuously updated displacement register value for all three measurement axes. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| position0 | absolute position of the axis 0 in pm |
| position1 | absolute position of the axis 1 in pm |
| position2 | absolute position of the axis 2 in pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getAbsolutePositions |
| params: [] |
| Result: [warningNo, position0, position1, position2] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getAbsolutePositions**(int deviceHandle, int\* warningNo, double\* position0, double\* position1, double\* position2) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, position0, position1, position2 = **[dev].displacement.getAbsolutePositions**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, position0, position1, position2] = **SEN\_displacement\_getAbsolutePositions**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,double,double,double> value = [Device].**Displacement\_GetAbsolutePositions**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAbsolutePositions.vi |

|  |
| --- |
| **getAverageN** Reads out the averaging (lowpass) parameter N. The averaging time is calculated by (2^N)\*40ns, where N is the averaging value. Please refer to the manual for a table with stopband and 3dB cut-off frequency. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| averageN | value from 0 to 24 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getAverageN |
| params: [] |
| Result: [errNo, averageN] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getAverageN**(int deviceHandle, int\* averageN) |

|  |
| --- |
| **Python** |

|  |
| --- |
| averageN = **[dev].displacement.getAverageN**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [averageN] = **SEN\_displacement\_getAverageN**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Displacement\_GetAverageN**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAverageN.vi |

|  |
| --- |
| **getAxesDisplacement** Reads out the displacement values of all three measurement axes. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| displacement0 | displacement of the axis 0 in pm |
| displacement1 | displacement of the axis 1 in pm |
| displacement2 | displacement of the axis 2 in pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getAxesDisplacement |
| params: [] |
| Result: [warningNo, displacement0, displacement1, displacement2] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getAxesDisplacement**(int deviceHandle, int\* warningNo, double\* displacement0, double\* displacement1, double\* displacement2) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, displacement0, displacement1, displacement2 = **[dev].displacement.getAxesDisplacement**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, displacement0, displacement1, displacement2] = **SEN\_displacement\_getAxesDisplacement**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,double,double,double> value = [Device].**Displacement\_GetAxesDisplacement**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAxesDisplacement.vi |

|  |
| --- |
| **getAxisDisplacement** Reads out the displacement value of a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| displacement | displacement of the axis in pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getAxisDisplacement |
| params: [axis] |
| Result: [warningNo, displacement] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getAxisDisplacement**(int deviceHandle, int axis, int\* warningNo, double\* displacement) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, displacement = **[dev].displacement.getAxisDisplacement**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, displacement] = **SEN\_displacement\_getAxisDisplacement**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,double> value = [Device].**Displacement\_GetAxisDisplacement**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAxisDisplacement.vi |

|  |
| --- |
| **getAxisSignalQuality** This function can be used to monitor the signal contrast (peak-to-peak of the basic interference signal amplitude) and the baseline (its offset) during a running measurement. In comparison to the getContrastInPermille function, the contrast is additionally corrected by a normalization factor.  Please note that the getAxisSignalQuality function output is only updated when a displacement is measured. This means that angular misalignments without displacement changes on the measurement axes cannot be detected.  Furthermore, we recommend using the high accuracy initialization to obtain correct values directly after measurement initialization. When using the quick initialization, the initial value might be fcompromised at first but correct after some detected measurement (because it is updated with the correct value then). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| contrast | contrast of the base band signal in permille |
| baseline | offset of the contrast measurement in permille |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getAxisSignalQuality |
| params: [axis] |
| Result: [warningNo, contrast, baseline] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getAxisSignalQuality**(int deviceHandle, int axis, int\* warningNo, int\* contrast, int\* baseline) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, contrast, baseline = **[dev].displacement.getAxisSignalQuality**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, contrast, baseline] = **SEN\_displacement\_getAxisSignalQuality**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,int,int> value = [Device].**Displacement\_GetAxisSignalQuality**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAxisSignalQuality.vi |

|  |
| --- |
| **getMeasurementEnabled** This function can be used to see if the measurement is running. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getMeasurementEnabled |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getMeasurementEnabled**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].displacement.getMeasurementEnabled**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **SEN\_displacement\_getMeasurementEnabled**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Displacement\_GetMeasurementEnabled**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getMeasurementEnabled.vi |

|  |
| --- |
| **getReferencePosition** Reads out the reference position value of the specific measurement axis. The reference position information is estimated at the measurement initialization procedure or on reset position. This initial absolute position information is not updated during system operation, whereas the device continuously measures the relative displacement from that point. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| position | reference position of the axis in pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getReferencePosition |
| params: [axis] |
| Result: [warningNo, position] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getReferencePosition**(int deviceHandle, int axis, int\* warningNo, double\* position) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, position = **[dev].displacement.getReferencePosition**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, position] = **SEN\_displacement\_getReferencePosition**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,double> value = [Device].**Displacement\_GetReferencePosition**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getReferencePosition.vi |

|  |
| --- |
| **getReferencePositions** Reads out the reference position value of all three measurement axes.  The reference position information is estimated at the measurement initialization procedure or on reset position. This initial absolute position information is not updated during system operation, whereas the device continuously measures the relative displacement from that point. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | warningNo | warning code, can be converted into a string using the errorNumberToString function |
| position0 | reference position of the axis 0 in pm |
| position1 | reference position of the axis 1 in pm |
| position2 | reference position of the axis 2 in pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.getReferencePositions |
| params: [] |
| Result: [warningNo, position0, position1, position2] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_getReferencePositions**(int deviceHandle, int\* warningNo, double\* position0, double\* position1, double\* position2) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, position0, position1, position2 = **[dev].displacement.getReferencePositions**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, position0, position1, position2] = **SEN\_displacement\_getReferencePositions**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,double,double,double> value = [Device].**Displacement\_GetReferencePositions**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getReferencePositions.vi |

|  |
| --- |
| **setAverageN** Sets the averaging (lowpass) parameter N. The averaging time is calculated by (2^N)\*40ns, where N is the averaging value. Please refer to the manual for a table with stopband and 3dB cut-off frequency. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | averageN | value from 0 to 24 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.displacement.setAverageN |
| params: [averageN] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_displacement\_setAverageN**(int deviceHandle, int averageN) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].displacement.setAverageN**(averageN) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_displacement\_setAverageN**(averageN) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Displacement\_SetAverageN**(int averageN) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setAverageN.vi |

## Ecu

|  |
| --- |
| **disable** Disables the ECU interface. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.disable |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_disable**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].ecu.disable**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_ecu\_disable**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Ecu\_Disable**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| disable.vi |

|  |
| --- |
| **enable** Enables the ECU interface. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.enable |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_enable**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].ecu.enable**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_ecu\_enable**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Ecu\_Enable**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| enable.vi |

|  |
| --- |
| **getConnected** Reads out whether the ECU interface is physically connected or not. Checking if the ECU is connected can only be done on an enabled ECU interface. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| connected | boolean true = connected; false = disconnected |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getConnected |
| params: [] |
| Result: [errNo, connected] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getConnected**(int deviceHandle, bool\* connected) |

|  |
| --- |
| **Python** |

|  |
| --- |
| connected = **[dev].ecu.getConnected**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [connected] = **SEN\_ecu\_getConnected**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Ecu\_GetConnected**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getConnected.vi |

|  |
| --- |
| **getEnabled** Reads out whether the ECU interface is enabled or not. Enabling the ECU interface is crucial for working with the ECU. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| enabled | boolean true = enabled; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getEnabled |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getEnabled**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].ecu.getEnabled**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **SEN\_ecu\_getEnabled**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Ecu\_GetEnabled**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getEnabled.vi |

|  |
| --- |
| **getHumidityInPercent** Reads out the ECU measured air humidity in percent. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| humidity | double value in percent |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getHumidityInPercent |
| params: [] |
| Result: [errNo, humidity] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getHumidityInPercent**(int deviceHandle, double\* humidity) |

|  |
| --- |
| **Python** |

|  |
| --- |
| humidity = **[dev].ecu.getHumidityInPercent**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [humidity] = **SEN\_ecu\_getHumidityInPercent**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Ecu\_GetHumidityInPercent**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getHumidityInPercent.vi |

|  |
| --- |
| **getPressureInHPa** Reads out the ECU measured air pressure in hPa. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| pressure | double value in hPa |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getPressureInHPa |
| params: [] |
| Result: [errNo, pressure] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getPressureInHPa**(int deviceHandle, double\* pressure) |

|  |
| --- |
| **Python** |

|  |
| --- |
| pressure = **[dev].ecu.getPressureInHPa**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [pressure] = **SEN\_ecu\_getPressureInHPa**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Ecu\_GetPressureInHPa**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPressureInHPa.vi |

|  |
| --- |
| **getRefractiveIndex** Reads out the ECU estimated refractive index from the current ECU values. To get the refractive index for other compensation modes, please refer to the getRefractiveIndexForCompensation function. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| rIndex | double refractive index |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getRefractiveIndex |
| params: [] |
| Result: [errNo, rIndex] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getRefractiveIndex**(int deviceHandle, double\* rIndex) |

|  |
| --- |
| **Python** |

|  |
| --- |
| rIndex = **[dev].ecu.getRefractiveIndex**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [rIndex] = **SEN\_ecu\_getRefractiveIndex**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Ecu\_GetRefractiveIndex**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRefractiveIndex.vi |

|  |
| --- |
| **getRefractiveIndexCompensationMode** Reads out the compensation mode which is currently used for the environmental compensation. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| mode | int32 0 = direct ECU mode  1 = uses refractive index calculated from the manually set values 2 = uses manually set refrative index |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getRefractiveIndexCompensationMode |
| params: [axis] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getRefractiveIndexCompensationMode**(int deviceHandle, int axis, int\* mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].ecu.getRefractiveIndexCompensationMode**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **SEN\_ecu\_getRefractiveIndexCompensationMode**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Ecu\_GetRefractiveIndexCompensationMode**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRefractiveIndexCompensationMode.vi |

|  |
| --- |
| **getRefractiveIndexForCompensation** Reads out the refractive index used according to the current environmental compensation mode for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| rIndex | double refractive index |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getRefractiveIndexForCompensation |
| params: [axis] |
| Result: [errNo, rIndex] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getRefractiveIndexForCompensation**(int deviceHandle, int axis, double\* rIndex) |

|  |
| --- |
| **Python** |

|  |
| --- |
| rIndex = **[dev].ecu.getRefractiveIndexForCompensation**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [rIndex] = **SEN\_ecu\_getRefractiveIndexForCompensation**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Ecu\_GetRefractiveIndexForCompensation**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRefractiveIndexForCompensation.vi |

|  |
| --- |
| **getTemperatureInDegrees** Reads out the ECU measured air temperature in degrees Celsius. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| temperature | double value in degrees Celsius |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.getTemperatureInDegrees |
| params: [] |
| Result: [errNo, temperature] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_getTemperatureInDegrees**(int deviceHandle, double\* temperature) |

|  |
| --- |
| **Python** |

|  |
| --- |
| temperature = **[dev].ecu.getTemperatureInDegrees**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [temperature] = **SEN\_ecu\_getTemperatureInDegrees**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Ecu\_GetTemperatureInDegrees**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getTemperatureInDegrees.vi |

|  |
| --- |
| **setRefractiveIndexCompensationMode** Sets the refractive index compensation mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| mode | 0 = direct ECU mode  1 = uses refractive index calculated from the manually set values 2 = uses manually set refrative index |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.setRefractiveIndexCompensationMode |
| params: [axis, mode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_ecu\_setRefractiveIndexCompensationMode**(int deviceHandle, int axis, int mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].ecu.setRefractiveIndexCompensationMode**(axis, mode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_ecu\_setRefractiveIndexCompensationMode**(axis, mode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Ecu\_SetRefractiveIndexCompensationMode**(int axis, int mode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setRefractiveIndexCompensationMode.vi |

## Manual

|  |
| --- |
| **getHumidityInPercent** Reads out the manually configured humidity for compensation mode 1. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| humidity | double value in percent |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.getHumidityInPercent |
| params: [axis] |
| Result: [errNo, humidity] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_getHumidityInPercent**(int deviceHandle, int axis, double\* humidity) |

|  |
| --- |
| **Python** |

|  |
| --- |
| humidity = **[dev].manual.getHumidityInPercent**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [humidity] = **SEN\_manual\_getHumidityInPercent**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Manual\_GetHumidityInPercent**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getHumidityInPercent.vi |

|  |
| --- |
| **getPressureInHPa** Reads out the manually configured air pressure for compensation mode 1. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| pressure | double value in hPa |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.getPressureInHPa |
| params: [axis] |
| Result: [errNo, pressure] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_getPressureInHPa**(int deviceHandle, int axis, double\* pressure) |

|  |
| --- |
| **Python** |

|  |
| --- |
| pressure = **[dev].manual.getPressureInHPa**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [pressure] = **SEN\_manual\_getPressureInHPa**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Manual\_GetPressureInHPa**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPressureInHPa.vi |

|  |
| --- |
| **getRefractiveIndex** Reads out the manually configured refractive index for compensation mode 2. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| rindex | double refractive index |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.getRefractiveIndex |
| params: [axis] |
| Result: [errNo, rindex] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_getRefractiveIndex**(int deviceHandle, int axis, double\* rindex) |

|  |
| --- |
| **Python** |

|  |
| --- |
| rindex = **[dev].manual.getRefractiveIndex**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [rindex] = **SEN\_manual\_getRefractiveIndex**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Manual\_GetRefractiveIndex**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRefractiveIndex.vi |

|  |
| --- |
| **getTemperatureInDegrees** Reads out the manually configured temperature for compensation mode 1. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| temperature | double value in degrees Celsius |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.getTemperatureInDegrees |
| params: [axis] |
| Result: [errNo, temperature] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_getTemperatureInDegrees**(int deviceHandle, int axis, double\* temperature) |

|  |
| --- |
| **Python** |

|  |
| --- |
| temperature = **[dev].manual.getTemperatureInDegrees**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [temperature] = **SEN\_manual\_getTemperatureInDegrees**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| double value = [Device].**Manual\_GetTemperatureInDegrees**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getTemperatureInDegrees.vi |

|  |
| --- |
| **setHumidityInPercent** Sets the manually configured humidity for compensation mode 1. The input range is defined to 0 to 100 % (valid range for the Ciddor Equation). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| humidity | value from 0 % to 100 % |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.setHumidityInPercent |
| params: [axis, humidity] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_setHumidityInPercent**(int deviceHandle, int axis, double humidity) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].manual.setHumidityInPercent**(axis, humidity) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_manual\_setHumidityInPercent**(axis, humidity) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Manual\_SetHumidityInPercent**(int axis, double humidity) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setHumidityInPercent.vi |

|  |
| --- |
| **setPressureInHPa** Sets the manually configured air pressure for compensation mode 1. The input range is defined to 800 to 1200 hPa (valid range for the Ciddor Equation). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| pressure | value from 800 hPa to 1200 hPa |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.setPressureInHPa |
| params: [axis, pressure] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_setPressureInHPa**(int deviceHandle, int axis, double pressure) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].manual.setPressureInHPa**(axis, pressure) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_manual\_setPressureInHPa**(axis, pressure) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Manual\_SetPressureInHPa**(int axis, double pressure) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setPressureInHPa.vi |

|  |
| --- |
| **setRefractiveIndex** Sets the manually configured refractive index for the compensation mode 2. The input range is defined to be greater than 1. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| rindex | refractive index |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.setRefractiveIndex |
| params: [axis, rindex] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_setRefractiveIndex**(int deviceHandle, int axis, double rindex) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].manual.setRefractiveIndex**(axis, rindex) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_manual\_setRefractiveIndex**(axis, rindex) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Manual\_SetRefractiveIndex**(int axis, double rindex) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setRefractiveIndex.vi |

|  |
| --- |
| **setTemperatureInDegrees** Sets the manually configured temperature for compensation mode 1. The input range is defined to -40 to +100 °C (valid range for the Ciddor Equation). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| temperature | value from -40 °C to +100 °C |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.ecu.manual.setTemperatureInDegrees |
| params: [axis, temperature] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_manual\_setTemperatureInDegrees**(int deviceHandle, int axis, double temperature) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].manual.setTemperatureInDegrees**(axis, temperature) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_manual\_setTemperatureInDegrees**(axis, temperature) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Manual\_SetTemperatureInDegrees**(int axis, double temperature) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setTemperatureInDegrees.vi |

## Network

|  |
| --- |
| **apply** Applies the temporary IP configuration and loads it. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.apply |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_apply**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.apply**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_apply**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_Apply**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| apply.vi |

|  |
| --- |
| **configureWifi** Changes the WiFi configuration and applies it. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 0 = access point; 1: WiFi client |
| SSID | SSID string |
| psk | pre-shared key |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.configureWifi |
| params: [mode, SSID, psk] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_configureWifi**(int deviceHandle, int mode, const char\* SSID, const char\* psk) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.configureWifi**(mode, SSID, psk) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_configureWifi**(mode, SSID, psk) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_ConfigureWifi**(int mode, string SSID, string psk) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| configureWifi.vi |

|  |
| --- |
| **discard** Discards the temporary IP configuration. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.discard |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_discard**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.discard**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_discard**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_Discard**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| discard.vi |

|  |
| --- |
| **getDefaultGateway** Gets the default gateway of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| Default | gateway |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getDefaultGateway |
| params: [] |
| Result: [errNo, Default] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getDefaultGateway**(int deviceHandle, char\* Default, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| Default = **[dev].network.getDefaultGateway**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [Default] = **system\_network\_getDefaultGateway**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetDefaultGateway**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getDefaultGateway.vi |

|  |
| --- |
| **getDnsResolver** Gets the DNS resolver. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | priority | priority of DNS resolver (0 = default, 1 = backup) |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| resolver | IP address of DNS resolver |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getDnsResolver |
| params: [priority] |
| Result: [errNo, resolver] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getDnsResolver**(int deviceHandle, int priority, char\* resolver, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| resolver = **[dev].network.getDnsResolver**(priority) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [resolver] = **system\_network\_getDnsResolver**(priority) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetDnsResolver**(int priority) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getDnsResolver.vi |

|  |
| --- |
| **getEnableDhcpClient** Gets the status of DHCP client. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled, false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getEnableDhcpClient |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getEnableDhcpClient**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].network.getEnableDhcpClient**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **system\_network\_getEnableDhcpClient**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Network\_GetEnableDhcpClient**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getEnableDhcpClient.vi |

|  |
| --- |
| **getEnableDhcpServer** Gets the status of DHCP server. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled, false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getEnableDhcpServer |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getEnableDhcpServer**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].network.getEnableDhcpServer**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **system\_network\_getEnableDhcpServer**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Network\_GetEnableDhcpServer**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getEnableDhcpServer.vi |

|  |
| --- |
| **getIpAddress** Gets the IP address of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| IP | address as string |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getIpAddress |
| params: [] |
| Result: [errNo, IP] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getIpAddress**(int deviceHandle, char\* IP, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| IP = **[dev].network.getIpAddress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [IP] = **system\_network\_getIpAddress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetIpAddress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getIpAddress.vi |

|  |
| --- |
| **getProxyServer** Gets the proxy settings of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| proxyServer | proxy server setting, empty for no proxy |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getProxyServer |
| params: [] |
| Result: [errNo, proxyServer] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getProxyServer**(int deviceHandle, char\* proxyServer, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| proxyServer = **[dev].network.getProxyServer**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [proxyServer] = **system\_network\_getProxyServer**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetProxyServer**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getProxyServer.vi |

|  |
| --- |
| **getRealIpAddress** Gets the real IP address of the device set to the network interface (br0, eth1 or eth0). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| IP | IP address |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getRealIpAddress |
| params: [] |
| Result: [errNo, IP] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getRealIpAddress**(int deviceHandle, char\* IP, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| IP = **[dev].network.getRealIpAddress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [IP] = **system\_network\_getRealIpAddress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetRealIpAddress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRealIpAddress.vi |

|  |
| --- |
| **getSubnetMask** Gets the subnet mask of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| Subnet | mask as string |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getSubnetMask |
| params: [] |
| Result: [errNo, Subnet] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getSubnetMask**(int deviceHandle, char\* Subnet, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| Subnet = **[dev].network.getSubnetMask**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [Subnet] = **system\_network\_getSubnetMask**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetSubnetMask**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getSubnetMask.vi |

|  |
| --- |
| **getWifiMode** Gets the operation mode of the WiFi adapter. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| mode | 0 = access point; 1 = WiFi client |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getWifiMode |
| params: [] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getWifiMode**(int deviceHandle, int\* mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].network.getWifiMode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **system\_network\_getWifiMode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Network\_GetWifiMode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getWifiMode.vi |

|  |
| --- |
| **getWifiPassphrase** Gets the passphrase of the network hosted (mode: Access point) or connected to (mode: client). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| psk | pre-shared key |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getWifiPassphrase |
| params: [] |
| Result: [errNo, psk] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getWifiPassphrase**(int deviceHandle, char\* psk, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| psk = **[dev].network.getWifiPassphrase**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [psk] = **system\_network\_getWifiPassphrase**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetWifiPassphrase**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getWifiPassphrase.vi |

|  |
| --- |
| **getWifiPresent** Checks if a WiFi interface is present. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| present | true if the interface is present, else false |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getWifiPresent |
| params: [] |
| Result: [errNo, present] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getWifiPresent**(int deviceHandle, bool\* present) |

|  |
| --- |
| **Python** |

|  |
| --- |
| present = **[dev].network.getWifiPresent**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [present] = **system\_network\_getWifiPresent**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Network\_GetWifiPresent**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getWifiPresent.vi |

|  |
| --- |
| **getWifiSSID** Gets the SSID of the network hosted (mode: access point) or connected to (mode: client). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| SSID | SSID string |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getWifiSSID |
| params: [] |
| Result: [errNo, SSID] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getWifiSSID**(int deviceHandle, char\* SSID, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| SSID = **[dev].network.getWifiSSID**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [SSID] = **system\_network\_getWifiSSID**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetWifiSSID**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getWifiSSID.vi |

|  |
| --- |
| **setDefaultGateway** Sets the default gateway of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | gateway | Default gateway as string |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setDefaultGateway |
| params: [gateway] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setDefaultGateway**(int deviceHandle, const char\* gateway) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setDefaultGateway**(gateway) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setDefaultGateway**(gateway) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetDefaultGateway**(string gateway) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setDefaultGateway.vi |

|  |
| --- |
| **setDnsResolver** Sets the DNS resolver. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | priority | priority of DNS resolver (0 = default, 1 = backup) |
| resolver | IP address of DNS resolver |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setDnsResolver |
| params: [priority, resolver] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setDnsResolver**(int deviceHandle, int priority, const char\* resolver) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setDnsResolver**(priority, resolver) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setDnsResolver**(priority, resolver) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetDnsResolver**(int priority, string resolver) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setDnsResolver.vi |

|  |
| --- |
| **setEnableDhcpClient** Enables or disables DHCP client. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | enable | true = enable, false = disable |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setEnableDhcpClient |
| params: [enable] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setEnableDhcpClient**(int deviceHandle, bool enable) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setEnableDhcpClient**(enable) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setEnableDhcpClient**(enable) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetEnableDhcpClient**(bool enable) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setEnableDhcpClient.vi |

|  |
| --- |
| **setEnableDhcpServer** Enables or disables DHCP server. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | enable | true = enable, false = disable |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setEnableDhcpServer |
| params: [enable] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setEnableDhcpServer**(int deviceHandle, bool enable) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setEnableDhcpServer**(enable) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setEnableDhcpServer**(enable) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetEnableDhcpServer**(bool enable) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setEnableDhcpServer.vi |

|  |
| --- |
| **setIpAddress** Sets the IP address of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | address | IP address as string |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setIpAddress |
| params: [address] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setIpAddress**(int deviceHandle, const char\* address) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setIpAddress**(address) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setIpAddress**(address) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetIpAddress**(string address) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setIpAddress.vi |

|  |
| --- |
| **setProxyServer** Sets the proxy server of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | proxyServer | proxy server setting |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setProxyServer |
| params: [proxyServer] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setProxyServer**(int deviceHandle, const char\* proxyServer) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setProxyServer**(proxyServer) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setProxyServer**(proxyServer) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetProxyServer**(string proxyServer) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setProxyServer.vi |

|  |
| --- |
| **setSubnetMask** Sets the subnet mask of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | netmask | subnet mask as string |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setSubnetMask |
| params: [netmask] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setSubnetMask**(int deviceHandle, const char\* netmask) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setSubnetMask**(netmask) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setSubnetMask**(netmask) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetSubnetMask**(string netmask) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setSubnetMask.vi |

|  |
| --- |
| **setWifiMode** Changes the operation mode of the WiFi adapter. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 0 = access point; 1 = WiFi client |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setWifiMode |
| params: [mode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setWifiMode**(int deviceHandle, int mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setWifiMode**(mode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setWifiMode**(mode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetWifiMode**(int mode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setWifiMode.vi |

|  |
| --- |
| **setWifiPassphrase** Changes the passphrase of the network hosted (mode: access point) or connected to (mode: client). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | psk | pre-shared key |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setWifiPassphrase |
| params: [psk] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setWifiPassphrase**(int deviceHandle, const char\* psk) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setWifiPassphrase**(psk) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setWifiPassphrase**(psk) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetWifiPassphrase**(string psk) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setWifiPassphrase.vi |

|  |
| --- |
| **setWifiSSID** Changes the SSID of the network hosted (mode: access point) or connected to (mode: client). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | SSID | SSID string |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.setWifiSSID |
| params: [SSID] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_setWifiSSID**(int deviceHandle, const char\* SSID) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.setWifiSSID**(SSID) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_setWifiSSID**(SSID) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_SetWifiSSID**(string SSID) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setWifiSSID.vi |

|  |
| --- |
| **verify** Verifies if the temporary IP configuration is correct. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.verify |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_verify**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].network.verify**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_network\_verify**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Network\_Verify**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| verify.vi |

## Nlc

|  |
| --- |
| **clearLut** Deactivates the LUT and removes it from the device for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.clearLut |
| params: [axis] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_clearLut**(int deviceHandle, int axis) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.clearLut**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_clearLut**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_ClearLut**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| clearLut.vi |

|  |
| --- |
| **createLut** Creates a new LUT for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.createLut |
| params: [axis] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_createLut**(int deviceHandle, int axis) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.createLut**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_createLut**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_CreateLut**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| createLut.vi |

|  |
| --- |
| **estimateNonlinearities** Estimates the nonlinearity error for the current device settings without changing or updating any settings. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.estimateNonlinearities |
| params: [axis] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_estimateNonlinearities**(int deviceHandle, int axis) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.estimateNonlinearities**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_estimateNonlinearities**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_EstimateNonlinearities**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| estimateNonlinearities.vi |

|  |
| --- |
| **getDynamicNormalization** Returns the normalization mode of a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| mode | int32 0 = dynamic normalization (default) 1 = normalization frozen (for slow target drifts) 2 = normalization mode determined by target velocity |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getDynamicNormalization |
| params: [axis] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getDynamicNormalization**(int deviceHandle, int axis, int\* mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].nlc.getDynamicNormalization**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **SEN\_nlc\_getDynamicNormalization**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Nlc\_GetDynamicNormalization**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getDynamicNormalization.vi |

|  |
| --- |
| **getHistogram** Returns a histogram of the measured nonlinearity errors obtained from the last call of the createLut or estimateNonlinearites functions. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| histogram | string histogram array |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getHistogram |
| params: [axis] |
| Result: [errNo, histogram] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getHistogram**(int deviceHandle, int axis, char\* histogram, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| histogram = **[dev].nlc.getHistogram**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [histogram] = **SEN\_nlc\_getHistogram**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Nlc\_GetHistogram**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getHistogram.vi |

|  |
| --- |
| **getLut** Returns the LUT determined by the createLut function, which can be applied by the setLutApplied function.    Attention: the LUT values represent the internal units. To convert the LUT values into nm, please multiply the LUT array by factor ((2^16-1)\*(1530/2)). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 errNo |
| lut | string LUT array with 512 integers |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getLut |
| params: [axis] |
| Result: [errNo, lut] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getLut**(int deviceHandle, int axis, char\* lut, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| lut = **[dev].nlc.getLut**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [lut] = **SEN\_nlc\_getLut**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Nlc\_GetLut**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getLut.vi |

|  |
| --- |
| **getLutApplied** Returns whether a LUT is applied or not for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| apply | boolean true = applied; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getLutApplied |
| params: [axis] |
| Result: [errNo, apply] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getLutApplied**(int deviceHandle, int axis, bool\* apply) |

|  |
| --- |
| **Python** |

|  |
| --- |
| apply = **[dev].nlc.getLutApplied**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [apply] = **SEN\_nlc\_getLutApplied**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Nlc\_GetLutApplied**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getLutApplied.vi |

|  |
| --- |
| **getLutStatus** Reads out whether the LUT is available or not for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | warningNo | int32 warning code, can be converted into a string using the errorNumberToString function |
| status | boolean true if a LUT exists, else false |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getLutStatus |
| params: [axis] |
| Result: [warningNo, status] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getLutStatus**(int deviceHandle, int axis, int\* warningNo, bool\* status) |

|  |
| --- |
| **Python** |

|  |
| --- |
| warningNo, status = **[dev].nlc.getLutStatus**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [warningNo, status] = **SEN\_nlc\_getLutStatus**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,bool> value = [Device].**Nlc\_GetLutStatus**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getLutStatus.vi |

|  |
| --- |
| **getNonlinearityEstimation** Returns the LUT created by the estimateNonlinearities function (read-only mode) to compensate the nonlinearity error of the device for the current device settings. If no estimation was created an array of zeros is returned.    Attention: the LUT values represent the internal units. To convert the LUT values into nm, please multiply the LUT array by factor ((2^16-1)\*(1530/2)). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| lut | string LUT array with 512 integers |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getNonlinearityEstimation |
| params: [] |
| Result: [errNo, lut] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getNonlinearityEstimation**(int deviceHandle, char\* lut, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| lut = **[dev].nlc.getNonlinearityEstimation**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [lut] = **SEN\_nlc\_getNonlinearityEstimation**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Nlc\_GetNonlinearityEstimation**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getNonlinearityEstimation.vi |

|  |
| --- |
| **getVelocityThresholds** Returns the threshold velocities for mode 2 of the setDynamicNormalization function. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |
| velocityOn | int32 target velocity in µm/s (default: 10 µm/s) |
| velocityOff | int32 target velocity in µm/s (default: 5 µm/s) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.getVelocityThresholds |
| params: [] |
| Result: [errNo, velocityOn, velocityOff] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_getVelocityThresholds**(int deviceHandle, int\* velocityOn, int\* velocityOff) |

|  |
| --- |
| **Python** |

|  |
| --- |
| velocityOn, velocityOff = **[dev].nlc.getVelocityThresholds**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [velocityOn, velocityOff] = **SEN\_nlc\_getVelocityThresholds**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| Tuple<int,int> value = [Device].**Nlc\_GetVelocityThresholds**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getVelocityThresholds.vi |

|  |
| --- |
| **setDynamicNormalization** Sets the normalization mode of a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| mode | 0 = dynamic normalization  1 = normalization frozen (for slow target drifts) 2 = automatic alternation between mode 0 and 1 depending on the target velocity |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.setDynamicNormalization |
| params: [axis, mode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_setDynamicNormalization**(int deviceHandle, int axis, int mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.setDynamicNormalization**(axis, mode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_setDynamicNormalization**(axis, mode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_SetDynamicNormalization**(int axis, int mode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setDynamicNormalization.vi |

|  |
| --- |
| **setLut** Uploads a LUT for a specific measurement axis, which can be applied by the setLutApplied function. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| lut | LUT array with 512 integers |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.setLut |
| params: [axis, lut] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_setLut**(int deviceHandle, int axis, const char\* lut) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.setLut**(axis, lut) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_setLut**(axis, lut) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_SetLut**(int axis, string lut) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setLut.vi |

|  |
| --- |
| **setLutApplied** Applies the LUT for a specific measurement axis. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| apply | true = apply LUT; false = disable LUT |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.setLutApplied |
| params: [axis, apply] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_setLutApplied**(int deviceHandle, int axis, bool apply) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.setLutApplied**(axis, apply) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_setLutApplied**(axis, apply) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_SetLutApplied**(int axis, bool apply) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setLutApplied.vi |

|  |
| --- |
| **setVelocityThresholds** Sets the threshold velocities for mode 2 of the setDynamicNormalization function. By default, the normalization in mode 2 is frozen for velocities below 5 µm/s and switches to dynamic mode for velocities above 10 µm/s. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | velocityOn | target velocity in µm/s (default: 10 µm/s) |
| velocityOff | target velocity in µm/s (default: 5 µm/s) |
| Out | errNo | int32 error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.nlc.setVelocityThresholds |
| params: [velocityOn, velocityOff] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_nlc\_setVelocityThresholds**(int deviceHandle, int velocityOn, int velocityOff) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].nlc.setVelocityThresholds**(velocityOn, velocityOff) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_nlc\_setVelocityThresholds**(velocityOn, velocityOff) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Nlc\_SetVelocityThresholds**(int velocityOn, int velocityOff) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setVelocityThresholds.vi |

## Pilotlaser

|  |
| --- |
| **disable** Disables the pilot laser. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.pilotlaser.disable |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_pilotlaser\_disable**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].pilotlaser.disable**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_pilotlaser\_disable**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Pilotlaser\_Disable**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| disable.vi |

|  |
| --- |
| **enable** Enables the pilot laser. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.pilotlaser.enable |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_pilotlaser\_enable**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].pilotlaser.enable**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_pilotlaser\_enable**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Pilotlaser\_Enable**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| enable.vi |

|  |
| --- |
| **getEnabled** Reads out whether the pilot laser is enabled or not. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.pilotlaser.getEnabled |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_pilotlaser\_getEnabled**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].pilotlaser.getEnabled**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **SEN\_pilotlaser\_getEnabled**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Pilotlaser\_GetEnabled**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getEnabled.vi |

## Realtime

|  |
| --- |
| **apply** Applies new real-time settings. Necessary after JSON real-time set commands. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.apply |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_apply**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.apply**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_apply**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_Apply**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| apply.vi |

|  |
| --- |
| **disableTestChannel** Disables the test channel.  This function is only available if the test channels were previously activated, and the system is in the "test channels enabled" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.disableTestChannel |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_disableTestChannel**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.disableTestChannel**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_disableTestChannel**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_DisableTestChannel**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| disableTestChannel.vi |

|  |
| --- |
| **discard** Discards new real-time settings. Necessary after JSON set commands in case of failure. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.discard |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_discard**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.discard**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_discard**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_Discard**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| discard.vi |

|  |
| --- |
| **enableTestChannel** Enables the test channel on measurement axis 0, which can be used for estimating the maximum signal range. This function is only available in the "system idle" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.enableTestChannel |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_enableTestChannel**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.enableTestChannel**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_enableTestChannel**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_EnableTestChannel**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| enableTestChannel.vi |

|  |
| --- |
| **getAafAttenuation** Returns the current attenuation of the anti-aliasing filter. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| attenuation | value from 3 dB to 30 dB (default: 15 dB) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getAafAttenuation |
| params: [] |
| Result: [errNo, attenuation] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getAafAttenuation**(int deviceHandle, int\* attenuation) |

|  |
| --- |
| **Python** |

|  |
| --- |
| attenuation = **[dev].realtime.getAafAttenuation**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [attenuation] = **SEN\_realtime\_getAafAttenuation**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetAafAttenuation**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAafAttenuation.vi |

|  |
| --- |
| **getAafEnabled** Checks if the anti-aliasing filter is enabled. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getAafEnabled |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getAafEnabled**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].realtime.getAafEnabled**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **SEN\_realtime\_getAafEnabled**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Realtime\_GetAafEnabled**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAafEnabled.vi |

|  |
| --- |
| **getAafWindow** Returns the current filter window of the anti-aliasing filter. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| window | 0 = Rectangular, 1 = Cosine, 2 = Cosine^2, 3 = Hamming, 4 = Raised Cosine (default) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getAafWindow |
| params: [] |
| Result: [errNo, window] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getAafWindow**(int deviceHandle, int\* window) |

|  |
| --- |
| **Python** |

|  |
| --- |
| window = **[dev].realtime.getAafWindow**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [window] = **SEN\_realtime\_getAafWindow**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetAafWindow**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getAafWindow.vi |

|  |
| --- |
| **getHighPassCutOffFreq** Reads out the high pass filter number of Linear analog output mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| filternumber | value from 1 to 24 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getHighPassCutOffFreq |
| params: [] |
| Result: [errNo, filternumber] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getHighPassCutOffFreq**(int deviceHandle, int\* filternumber) |

|  |
| --- |
| **Python** |

|  |
| --- |
| filternumber = **[dev].realtime.getHighPassCutOffFreq**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [filternumber] = **SEN\_realtime\_getHighPassCutOffFreq**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetHighPassCutOffFreq**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getHighPassCutOffFreq.vi |

|  |
| --- |
| **getLinearRange** Reads out the range number N of Linear analog output mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| rangenumber | value from 0 to 34 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getLinearRange |
| params: [] |
| Result: [errNo, rangenumber] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getLinearRange**(int deviceHandle, int\* rangenumber) |

|  |
| --- |
| **Python** |

|  |
| --- |
| rangenumber = **[dev].realtime.getLinearRange**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [rangenumber] = **SEN\_realtime\_getLinearRange**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetLinearRange**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getLinearRange.vi |

|  |
| --- |
| **getPeriodHsslClk** Reads out the HSSL period clock. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| period | value from 40 ns to 10200 ns |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getPeriodHsslClk |
| params: [] |
| Result: [errNo, period] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getPeriodHsslClk**(int deviceHandle, int\* period) |

|  |
| --- |
| **Python** |

|  |
| --- |
| period = **[dev].realtime.getPeriodHsslClk**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [period] = **SEN\_realtime\_getPeriodHsslClk**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetPeriodHsslClk**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPeriodHsslClk.vi |

|  |
| --- |
| **getPeriodHsslGap** Reads out the HSSL period gap as number of clocks. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| gap | value from 1 to 255 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getPeriodHsslGap |
| params: [] |
| Result: [errNo, gap] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getPeriodHsslGap**(int deviceHandle, int\* gap) |

|  |
| --- |
| **Python** |

|  |
| --- |
| gap = **[dev].realtime.getPeriodHsslGap**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [gap] = **SEN\_realtime\_getPeriodHsslGap**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetPeriodHsslGap**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPeriodHsslGap.vi |

|  |
| --- |
| **getPeriodSinCosClk** Reads out the Sin/Cos and AquadB period clock. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| period | value from 40 ns to 10200 ns |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getPeriodSinCosClk |
| params: [] |
| Result: [errNo, period] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getPeriodSinCosClk**(int deviceHandle, int\* period) |

|  |
| --- |
| **Python** |

|  |
| --- |
| period = **[dev].realtime.getPeriodSinCosClk**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [period] = **SEN\_realtime\_getPeriodSinCosClk**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetPeriodSinCosClk**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getPeriodSinCosClk.vi |

|  |
| --- |
| **getResolutionBissC** Reads out the BiSS C resolution. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| resolution | value from 0 to 16 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getResolutionBissC |
| params: [] |
| Result: [errNo, resolution] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getResolutionBissC**(int deviceHandle, int\* resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| resolution = **[dev].realtime.getResolutionBissC**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [resolution] = **SEN\_realtime\_getResolutionBissC**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionBissC**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getResolutionBissC.vi |

|  |
| --- |
| **getResolutionHsslHigh** Reads out the HSSL resolution high bit. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| resolution | value from 1 to 47 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getResolutionHsslHigh |
| params: [] |
| Result: [errNo, resolution] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getResolutionHsslHigh**(int deviceHandle, int\* resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| resolution = **[dev].realtime.getResolutionHsslHigh**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [resolution] = **SEN\_realtime\_getResolutionHsslHigh**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionHsslHigh**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getResolutionHsslHigh.vi |

|  |
| --- |
| **getResolutionHsslLow** Reads out the HSSL resolution low bit. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| resolution | value from 0 to 46 |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getResolutionHsslLow |
| params: [] |
| Result: [errNo, resolution] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getResolutionHsslLow**(int deviceHandle, int\* resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| resolution = **[dev].realtime.getResolutionHsslLow**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [resolution] = **SEN\_realtime\_getResolutionHsslLow**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionHsslLow**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getResolutionHsslLow.vi |

|  |
| --- |
| **getResolutionSinCos** Reads out the Sin/Cos and AquadB resolution. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| resolution | value from 1 pm to 65535 pm |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getResolutionSinCos |
| params: [] |
| Result: [errNo, resolution] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getResolutionSinCos**(int deviceHandle, int\* resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| resolution = **[dev].realtime.getResolutionSinCos**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [resolution] = **SEN\_realtime\_getResolutionSinCos**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionSinCos**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getResolutionSinCos.vi |

|  |
| --- |
| **getRtDistanceMode** Reads out the distance mode. Depending on the real-time output mode, the mode can be Displacement, Absolute Distance or Vibrometry. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| mode | 1 = Displacement (all modes) 2 = Absolute Distance (HSSL mode and BiSS C mode) 3 = Vibrometry (Linear mode only) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getRtDistanceMode |
| params: [] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getRtDistanceMode**(int deviceHandle, int\* mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].realtime.getRtDistanceMode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **SEN\_realtime\_getRtDistanceMode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetRtDistanceMode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRtDistanceMode.vi |

|  |
| --- |
| **getRtOutMode** Reads out the current real-time output mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| rtOutMode | 0 = HSSL (LVTTL), 1 = HSSL (LVDS), 2 = AquadB (LVTTL), 3 = AquadB (LVDS), 4 = Sin/Cos (LVTTL error signal), 5 = Sin/Cos (LVDS error signal), 6 = Linear (LVTTL error signal), 7 = Linear (LVDS error signal), 8 = BiSS-C, 9 = Deactivated |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getRtOutMode |
| params: [] |
| Result: [errNo, rtOutMode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getRtOutMode**(int deviceHandle, int\* rtOutMode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| rtOutMode = **[dev].realtime.getRtOutMode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [rtOutMode] = **SEN\_realtime\_getRtOutMode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Realtime\_GetRtOutMode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getRtOutMode.vi |

|  |
| --- |
| **getTestChannelEnabled** Checks if the test channels are enabled. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| enabled | true = enabled; false = disabled |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.getTestChannelEnabled |
| params: [] |
| Result: [errNo, enabled] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_getTestChannelEnabled**(int deviceHandle, bool\* enabled) |

|  |
| --- |
| **Python** |

|  |
| --- |
| enabled = **[dev].realtime.getTestChannelEnabled**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [enabled] = **SEN\_realtime\_getTestChannelEnabled**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Realtime\_GetTestChannelEnabled**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getTestChannelEnabled.vi |

|  |
| --- |
| **setAaf** Sets the anti-aliasing filter with assigned filter window. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | enable | true = enable; false = disable |
| attenuation | value from 3 dB to 30 dB |
| window | 0 = Rectangular, 1 = Cosine, 2 = Cosine^2, 3 = Hamming, 4 = Raised Cosine |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setAaf |
| params: [enable, attenuation, window] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setAaf**(int deviceHandle, bool enable, int attenuation, int window) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setAaf**(enable, attenuation, window) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setAaf**(enable, attenuation, window) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetAaf**(bool enable, int attenuation, int window) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setAaf.vi |

|  |
| --- |
| **setHighPassCutOffFreq** Sets the high pass filter number of Linear analog output mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | filternumber | value from 1 to 24 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setHighPassCutOffFreq |
| params: [filternumber] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setHighPassCutOffFreq**(int deviceHandle, int filternumber) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setHighPassCutOffFreq**(filternumber) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setHighPassCutOffFreq**(filternumber) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetHighPassCutOffFreq**(int filternumber) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setHighPassCutOffFreq.vi |

|  |
| --- |
| **setLinearRange** Sets the range number of Linear analog output mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | rangenumber | value from 0 to 34 |
| Out | errNo | errNo |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setLinearRange |
| params: [rangenumber] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setLinearRange**(int deviceHandle, int rangenumber) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setLinearRange**(rangenumber) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setLinearRange**(rangenumber) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetLinearRange**(int rangenumber) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setLinearRange.vi |

|  |
| --- |
| **setPeriodHsslClk** Set the HSSL period clock. The value must be a multiple of 40 ns. If not, the value will be rounded automatically. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | period | value from 40 ns to 10200 ns |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setPeriodHsslClk |
| params: [period] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setPeriodHsslClk**(int deviceHandle, int period) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setPeriodHsslClk**(period) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setPeriodHsslClk**(period) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetPeriodHsslClk**(int period) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setPeriodHsslClk.vi |

|  |
| --- |
| **setPeriodHsslGap** Set the HSSL period gap as number of clocks. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | gap | value from 1 to 255 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setPeriodHsslGap |
| params: [gap] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setPeriodHsslGap**(int deviceHandle, int gap) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setPeriodHsslGap**(gap) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setPeriodHsslGap**(gap) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetPeriodHsslGap**(int gap) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setPeriodHsslGap.vi |

|  |
| --- |
| **setPeriodSinCosClk** Sets the Sin/Cos and AquadB period clock. The value must be a multiple of 40ns. If not, the value will be rounded automatically. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | period | value from 40 ns to 10200 ns |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setPeriodSinCosClk |
| params: [period] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setPeriodSinCosClk**(int deviceHandle, int period) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setPeriodSinCosClk**(period) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setPeriodSinCosClk**(period) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetPeriodSinCosClk**(int period) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setPeriodSinCosClk.vi |

|  |
| --- |
| **setResolutionBissC** Sets the BiSS C resolution. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | resolution | value from 0 to 16 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setResolutionBissC |
| params: [resolution] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setResolutionBissC**(int deviceHandle, int resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setResolutionBissC**(resolution) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setResolutionBissC**(resolution) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionBissC**(int resolution) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setResolutionBissC.vi |

|  |
| --- |
| **setResolutionHsslHigh** Sets the HSSL resolution high bit. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | resolution | value from 1 to 47 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setResolutionHsslHigh |
| params: [resolution] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setResolutionHsslHigh**(int deviceHandle, int resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setResolutionHsslHigh**(resolution) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setResolutionHsslHigh**(resolution) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionHsslHigh**(int resolution) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setResolutionHsslHigh.vi |

|  |
| --- |
| **setResolutionHsslLow** Sets the HSSL resolution low bit. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | resolution | value from 0 to 46 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setResolutionHsslLow |
| params: [resolution] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setResolutionHsslLow**(int deviceHandle, int resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setResolutionHsslLow**(resolution) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setResolutionHsslLow**(resolution) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionHsslLow**(int resolution) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setResolutionHsslLow.vi |

|  |
| --- |
| **setResolutionSinCos** Sets the Sin/Cos and AquadB resolution. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | resolution | value from 1 pm to 65535 pm |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setResolutionSinCos |
| params: [resolution] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setResolutionSinCos**(int deviceHandle, int resolution) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setResolutionSinCos**(resolution) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setResolutionSinCos**(resolution) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionSinCos**(int resolution) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setResolutionSinCos.vi |

|  |
| --- |
| **setRtDistanceMode** Sets the distance mode. Depending on the configuration of the device, the mode can be Displacement, Absolute Distance or Vibrometry. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 1 = Displacement (all modes) 2 = Absolute Distance (HSSL mode and BiSS C mode) 3 = Vibrometry (Linear mode only) |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setRtDistanceMode |
| params: [mode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setRtDistanceMode**(int deviceHandle, int mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setRtDistanceMode**(mode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setRtDistanceMode**(mode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetRtDistanceMode**(int mode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setRtDistanceMode.vi |

|  |
| --- |
| **setRtOutMode** Sets the real-time output mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | rtOutMode | 0 = HSSL (LVTTL), 1 = HSSL (LVDS), 2 = AquadB (LVTTL), 3 = AquadB (LVDS), 4 = Sin/Cos (LVTTL error signal), 5 = Sin/Cos (LVDS error signal), 6 = Linear (LVTTL error signal), 7 = Linear (LVDS error signal), 8 = BiSS-C, 9 = Deactivated |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.realtime.setRtOutMode |
| params: [rtOutMode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_realtime\_setRtOutMode**(int deviceHandle, int rtOutMode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].realtime.setRtOutMode**(rtOutMode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_realtime\_setRtOutMode**(rtOutMode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Realtime\_SetRtOutMode**(int rtOutMode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setRtOutMode.vi |

## System

|  |
| --- |
| **getCurrentMode** Reads out the current device system state. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| mode | "system idle", "measurement starting", "measurement running", "measurement restarting", "optics alignment starting", "optics alignment running", "test channels enabled" |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getCurrentMode |
| params: [] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getCurrentMode**(int deviceHandle, char\* mode, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].system.getCurrentMode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **SEN\_system\_getCurrentMode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**System\_GetCurrentMode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getCurrentMode.vi |

|  |
| --- |
| **getDeviceType** Reads out the device type. For different device types please refer to the device user manual. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| type | type of the device (e.g. "IDS3010") |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getDeviceType |
| params: [] |
| Result: [errNo, type] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getDeviceType**(int deviceHandle, char\* type, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| type = **[dev].system.getDeviceType**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [type] = **SEN\_system\_getDeviceType**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**System\_GetDeviceType**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getDeviceType.vi |

|  |
| --- |
| **getFeaturesName** Converts the device feature number to its corresponding name. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | featurenumber | number of features |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| names | name of the corresponding feature |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getFeaturesName |
| params: [featurenumber] |
| Result: [errNo, names] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getFeaturesName**(int deviceHandle, int featurenumber, char\* names, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| names = **[dev].system.getFeaturesName**(featurenumber) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [names] = **SEN\_system\_getFeaturesName**(featurenumber) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**System\_GetFeaturesName**(int featurenumber) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getFeaturesName.vi |

|  |
| --- |
| **getFpgaVersion** Reads out the device FPGA version. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| version | version in the form X.Y.Z |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getFpgaVersion |
| params: [] |
| Result: [errNo, version] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getFpgaVersion**(int deviceHandle, char\* version, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| version = **[dev].system.getFpgaVersion**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [version] = **SEN\_system\_getFpgaVersion**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**System\_GetFpgaVersion**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getFpgaVersion.vi |

|  |
| --- |
| **getInitMode** Returns the initialization mode. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| mode | 0 = High Accuracy Initialization; 1 = Quick Initialization (default: 0) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getInitMode |
| params: [] |
| Result: [errNo, mode] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getInitMode**(int deviceHandle, int\* mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mode = **[dev].system.getInitMode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mode] = **SEN\_system\_getInitMode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**System\_GetInitMode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getInitMode.vi |

|  |
| --- |
| **getNbrFeaturesActivated** Reads out the number of the activated features on the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| nbr | the number of activated features |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getNbrFeaturesActivated |
| params: [] |
| Result: [errNo, nbr] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getNbrFeaturesActivated**(int deviceHandle, int\* nbr) |

|  |
| --- |
| **Python** |

|  |
| --- |
| nbr = **[dev].system.getNbrFeaturesActivated**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [nbr] = **SEN\_system\_getNbrFeaturesActivated**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**System\_GetNbrFeaturesActivated**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getNbrFeaturesActivated.vi |

|  |
| --- |
| **getSystemError** Reads out the system error. The function returns an integer number which represents the error. The number can be converted into a string using the errorNumberToString function.  Use this function to query errors occurred while starting and running measurement and alignment. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.getSystemError |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_getSystemError**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.getSystemError**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_getSystemError**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_GetSystemError**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_getSystemError.vi |

|  |
| --- |
| **resetAxes** Resets the displacement value of all measurement axes to zero. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.resetAxes |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_resetAxes**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.resetAxes**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_resetAxes**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_ResetAxes**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_resetAxes.vi |

|  |
| --- |
| **resetAxis** Resets the displacement value of a specific measurement axis to zero. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.resetAxis |
| params: [axis] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_resetAxis**(int deviceHandle, int axis) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.resetAxis**(axis) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_resetAxis**(axis) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_ResetAxis**(int axis) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_resetAxis.vi |

|  |
| --- |
| **resetError** Resets a measurement error that has been raised with the aim to continue the interrupted measurement. It is configurable if an additional renormalization process (please refer to the device user manual) should be performed or not.    This function can be used in two cases:   1. FALSE: This function only clears displacement errors (e.g. after a beam interruption) of all three axes,  while the displacement measurement is running.   2. TRUE: This function can be used to normalize the Lissajous figure of all three axes  during the running displacement measurement by sweeping the laser temperature and to clear all displacement errors.  This normalization process takes around 14-20 seconds. This function can be used, for example, in two main applications:   A. The alignment contrast decreases due to the angular change of the target and/or sensor head without any  displacements (see Figure 45). Using this function, the Lissajous figure of each measurement axis gets normalized  and the high-resolution measurement are guaranteed.   B. After changing the optical components as, for example, the retroreflector this function can be used to normalize  the Lissajous figure after completely losing the signal (see Figure 46).   Attention: Depending on the boolean input parameter performRenormalization, it can be decided, whether the   renormalization process should be executed or skipped. If it is executed, the system needs around 14-20 seconds  to get back into the measurement mode. It is also important to comment that the recommended action after an error   has been raised is to stop or to restart the measurement. Moreover, losing displacement values due to an occurred   error the internal absolute position can be different to the real absolute position and this can could result in   dynamic movement errors or a wrong ECU compensation during the long-term displacement measurements. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | performRenormalization | true = enable; false = disable |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.resetError |
| params: [performRenormalization] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_resetError**(int deviceHandle, bool performRenormalization) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.resetError**(performRenormalization) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_resetError**(performRenormalization) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_ResetError**(bool performRenormalization) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_resetError.vi |

|  |
| --- |
| **restartMeasurement** Restarts the absolute measurement system state. This function is only available in the "measurement running" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.restartMeasurement |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_restartMeasurement**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.restartMeasurement**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_restartMeasurement**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_RestartMeasurement**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_restartMeasurement.vi |

|  |
| --- |
| **setInitMode** Sets the mode for the initialization procedure that is performed when starting a measurement. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 0 = High Accuracy Initialization; 1 = Quick Initialization |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.setInitMode |
| params: [mode] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_setInitMode**(int deviceHandle, int mode) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.setInitMode**(mode) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_setInitMode**(mode) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_SetInitMode**(int mode) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_setInitMode.vi |

|  |
| --- |
| **startMeasurement** Starts the measurement system state. This function is only available in the "system idle" state (please refer to the getCurrentMode function) |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.startMeasurement |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_startMeasurement**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.startMeasurement**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_startMeasurement**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_StartMeasurement**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_startMeasurement.vi |

|  |
| --- |
| **startOpticsAlignment** Starts the optical alignment system state. This function is only available in the "system idle" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.startOpticsAlignment |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_startOpticsAlignment**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.startOpticsAlignment**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_startOpticsAlignment**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_StartOpticsAlignment**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_startOpticsAlignment.vi |

|  |
| --- |
| **stopMeasurement** Stops the measurement system state.  This function is only available in the "measurement running" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.stopMeasurement |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_stopMeasurement**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.stopMeasurement**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_stopMeasurement**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_StopMeasurement**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_stopMeasurement.vi |

|  |
| --- |
| **stopOpticsAlignment** Stops the optical alignment system state. This function is only available in the "optics alignment running" state (please refer to the getCurrentMode function). |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.sen.system.stopOpticsAlignment |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **SEN\_system\_stopOpticsAlignment**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system.stopOpticsAlignment**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **SEN\_system\_stopOpticsAlignment**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**System\_StopOpticsAlignment**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| system\_stopOpticsAlignment.vi |

## System\_service

|  |
| --- |
| **errorNumberToRecommendation** Gets a recommendation for the error code. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | language | integer: Language code |
| errNbr | error code to translate |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| value\_string1 | string: error recommendation (currently returning an int = 0 until we have recommendations) |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.errorNumberToRecommendation |
| params: [language, errNbr] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_errorNumberToRecommendation**(int deviceHandle, int language, int errNbr, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.errorNumberToRecommendation**(language, errNbr) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_errorNumberToRecommendation**(language, errNbr) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**ErrorNumberToRecommendation**(int language, int errNbr) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| errorNumberToRecommendation.vi |

|  |
| --- |
| **errorNumberToString** Gets a description of an error code. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | language | integer: Language code 0 for the error name, 1 for a more user-friendly error message |
| errNbr | error code to translate |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| message | error description |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.errorNumberToString |
| params: [language, errNbr] |
| Result: [errNo, message] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_errorNumberToString**(int deviceHandle, int language, int errNbr, char\* message, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| message = **[dev].system\_service.errorNumberToString**(language, errNbr) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [message] = **system\_errorNumberToString**(language, errNbr) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**ErrorNumberToString**(int language, int errNbr) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| errorNumberToString.vi |

|  |
| --- |
| **factoryReset** Turns on the factory reset flag. To perform the factory reset, a reboot is necessary afterwards. All settings will be set to default, and the device will be configured as DHCP server. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.factoryReset |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_factoryReset**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.factoryReset**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_factoryReset**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**FactoryReset**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| factoryReset.vi |

|  |
| --- |
| **getDeviceName** Gets the current device name. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| name | current device name |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getDeviceName |
| params: [] |
| Result: [errNo, name] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getDeviceName**(int deviceHandle, char\* name, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| name = **[dev].system\_service.getDeviceName**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [name] = **system\_getDeviceName**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetDeviceName**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getDeviceName.vi |

|  |
| --- |
| **getFirmwareVersion** Gets the firmware version of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code to translate |
| FWversion | firmware version |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getFirmwareVersion |
| params: [] |
| Result: [errNo, FWversion] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getFirmwareVersion**(int deviceHandle, char\* FWversion, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| FWversion = **[dev].system\_service.getFirmwareVersion**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [FWversion] = **system\_getFirmwareVersion**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetFirmwareVersion**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getFirmwareVersion.vi |

|  |
| --- |
| **getFluxCode** Gets the flux code of the system. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| code | flux code |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getFluxCode |
| params: [] |
| Result: [errNo, code] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getFluxCode**(int deviceHandle, char\* code, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| code = **[dev].system\_service.getFluxCode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [code] = **system\_getFluxCode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetFluxCode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getFluxCode.vi |

|  |
| --- |
| **getHostname** Returns the device hostname. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| name | hostname |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getHostname |
| params: [] |
| Result: [errNo, name] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getHostname**(int deviceHandle, char\* name, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| name = **[dev].system\_service.getHostname**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [name] = **system\_getHostname**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetHostname**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getHostname.vi |

|  |
| --- |
| **getMacAddress** Gets the mac address of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| mac | mac address |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getMacAddress |
| params: [] |
| Result: [errNo, mac] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getMacAddress**(int deviceHandle, char\* mac, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| mac = **[dev].system\_service.getMacAddress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [mac] = **system\_getMacAddress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetMacAddress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getMacAddress.vi |

|  |
| --- |
| **getSerialNumber** Gets the serial number of the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| SN | serial number |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getSerialNumber |
| params: [] |
| Result: [errNo, SN] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getSerialNumber**(int deviceHandle, char\* SN, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| SN = **[dev].system\_service.getSerialNumber**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [SN] = **system\_getSerialNumber**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetSerialNumber**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getSerialNumber.vi |

|  |
| --- |
| **rebootSystem** Reboots the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.rebootSystem |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_rebootSystem**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.rebootSystem**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_rebootSystem**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**RebootSystem**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| rebootSystem.vi |

|  |
| --- |
| **setDeviceName** Sets custom name for the device. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | name | device name |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.setDeviceName |
| params: [name] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_setDeviceName**(int deviceHandle, const char\* name) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.setDeviceName**(name) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_setDeviceName**(name) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**SetDeviceName**(string name) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setDeviceName.vi |

|  |
| --- |
| **setTime** Sets system time manually. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | day | value from 1 to 31 |
| month | value from 1 to 12 |
| year | value (e.g. 2021) |
| hour | value from 0 to 23 |
| minute | value from 0 to 59 |
| second | value from 0 to 59 |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.setTime |
| params: [day, month, year, hour, minute, second] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_setTime**(int deviceHandle, int day, int month, int year, int hour, int minute, int second) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.setTime**(day, month, year, hour, minute, second) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_setTime**(day, month, year, hour, minute, second) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**SetTime**(int day, int month, int year, int hour, int minute, int second) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| setTime.vi |

|  |
| --- |
| **softReset** Performs a soft reset (factory reset without deleting the network settings). Please reboot the device directly afterwards. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.softReset |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_softReset**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.softReset**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_softReset**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**SoftReset**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| softReset.vi |

|  |
| --- |
| **updateTimeFromInternet** Updates system time by querying attocube.com. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.updateTimeFromInternet |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_updateTimeFromInternet**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.updateTimeFromInternet**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_updateTimeFromInternet**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**UpdateTimeFromInternet**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| updateTimeFromInternet.vi |

## Update

|  |
| --- |
| **getLicenseUpdateProgress** Gets the progress of running license update. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| progress | value in % |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.getLicenseUpdateProgress |
| params: [] |
| Result: [errNo, progress] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_getLicenseUpdateProgress**(int deviceHandle, int\* progress) |

|  |
| --- |
| **Python** |

|  |
| --- |
| progress = **[dev].update.getLicenseUpdateProgress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [progress] = **system\_update\_getLicenseUpdateProgress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Update\_GetLicenseUpdateProgress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getLicenseUpdateProgress.vi |

|  |
| --- |
| **getSwUpdateProgress** Gets the progress of running update. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |
| progress | value in % |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.getSwUpdateProgress |
| params: [] |
| Result: [errNo, progress] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_getSwUpdateProgress**(int deviceHandle, int\* progress) |

|  |
| --- |
| **Python** |

|  |
| --- |
| progress = **[dev].update.getSwUpdateProgress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [progress] = **system\_update\_getSwUpdateProgress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Update\_GetSwUpdateProgress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getSwUpdateProgress.vi |

|  |
| --- |
| **licenseUpdateBase64** Executes the license update with base64 file uploaded. After execution, a manual reboot is necessary. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.licenseUpdateBase64 |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_licenseUpdateBase64**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].update.licenseUpdateBase64**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_update\_licenseUpdateBase64**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Update\_LicenseUpdateBase64**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| licenseUpdateBase64.vi |

|  |
| --- |
| **softwareUpdateBase64** Executes the update with base 64 file uploaded. After completion, a manual reboot is necessary. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.softwareUpdateBase64 |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_softwareUpdateBase64**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].update.softwareUpdateBase64**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_update\_softwareUpdateBase64**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Update\_SoftwareUpdateBase64**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| softwareUpdateBase64.vi |

|  |
| --- |
| **uploadLicenseBase64** Uploads new license file in format base 64. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | offset | offset of the data |
| b64Data | base64 data string |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.uploadLicenseBase64 |
| params: [offset, b64Data] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_uploadLicenseBase64**(int deviceHandle, int offset, const char\* b64Data) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].update.uploadLicenseBase64**(offset, b64Data) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_update\_uploadLicenseBase64**(offset, b64Data) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Update\_UploadLicenseBase64**(int offset, string b64Data) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| uploadLicenseBase64.vi |

|  |
| --- |
| **uploadSoftwareImageBase64** Uploads new firmware image in format base 64. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | offset | offset of the data |
| b64Data | base64 data string |
| Out | errNo | error code, if there was an error, otherwise 0 for ok |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.uploadSoftwareImageBase64 |
| params: [offset, b64Data] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_uploadSoftwareImageBase64**(int deviceHandle, int offset, const char\* b64Data) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].update.uploadSoftwareImageBase64**(offset, b64Data) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_update\_uploadSoftwareImageBase64**(offset, b64Data) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Update\_UploadSoftwareImageBase64**(int offset, string b64Data) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| uploadSoftwareImageBase64.vi |

## Streaming

The streaming API can be used with the optional streaming feature. These functions are used to implement the IDS position data streaming functionality. With streaming, defined sample rates of up to 1 MHz (only one measurement axis) can be retrieved via a software interface. To access the streaming functionality, the streaming feature must be activated on the IDS. Furthermore, the IDS must be already within measurement mode.

Please note that data can only be streamed by one instance. This means, that if you already use WAVE to stream data, you can not stream via the C API in addition.

If the streamed values are zero despite your expectations and a correct implementation, this can have resulted in an already closed stream. If your network resource is not sufficient for the desired streaming rates and number of axes, the IDS automatically stops streaming (please refer to the IDS WAVE Manual).

Please note that we offer different examples for the usage of the streaming API according to the environment.

|  |  |  |
| --- | --- | --- |
| **OpenStream** Initializes the data streaming on the IDS and returns the stream ID that is used as an input for the other streaming functions.  Note that the streaming will be started immediately. Until the target buffer is initialized, the values are written in the operating system’s cache. If the cache runs full, the streaming automatically gets aborted.  The streaming is also aborted if the desired streaming rate is higher than 1 MHz with one measurement axis, higher than 500 kHz with more measurement axes or extends your network capability in general. | | |
| **Function specific parameters** | | |
| In | host | The IDS IP address |
| isMaster | Only needed if multiple IDS are connected (not implemented yet) – therefore please always set this to “true” |
| intervallInMicroseconds | Samping time in microseconds -the desired streaming rate then is 1/intervallInMicroseconds (max. 1 MHz for one measurement axis, 500 kHz for more measurement axes) |
| channelMask (except python) | Defines the disred measurement axes by a channel mask (a value of 2^0 = 1 equals axis 1, 2^1 = 2 equals axis 2, 2^2 = 4 equals axis 3 – use a bitwise OR (“|”) to combine several axes: e.g: “1 | 4” for axis 1 and 3) |
| Axes (python only) | Used to define the channelMask; Desired measurement axis can be enabled/disabled through the class initializer using the respective axis variables, setting them to True or False (e.g. axis0 = True, axis1 = False, axis2 = False) |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| void\* OpenStream(const char \* *host*, const bool *isMaster*, const int *intervalInMicroseconds*, const uint8\_t *channelMask*) | | |
| **C#-DLL call** | | |
| IntPtr Open\_Stream(string host, bool isMaster, int intervalInMicrosecods, uint channelMask) | | |
| **Python** | | |
| **idsStream** = Stream(string host, bool isMaster, int intervalInMicroseconds, axis0=False, axis1=False, axis2=False)  **idsStream.open()** | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| / | | |

|  |  |  |
| --- | --- | --- |
| **CloseStream** Terminates the data streaming on the IDS. It is recommended to close the stream when it is not needed anymore. | | |
| **Function specific parameters** | | |
| In | stream  (except python) | The stream ID variable (initialized by the OpenStream function) |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| void CloseStream(void\* *stream*) | | |
| **C#-DLL call** | | |
| void Close\_Stream(IntPtr stream) | | |
| **Python** | | |
| [idsStream].close() | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| / | | |

|  |  |  |
| --- | --- | --- |
| **ReadStream** Retrieves the binary streaming data that has arrived in the system’s cache and returns the number of retrieved bytes. The binary data is packed into a buffer that has to be specified.  The buffer size can be configured between 2^4 (16 bytes) and 2^18 (262,144 bytes), corresponding to values from 1 << 4 to 1 << 18. It is important to note that only full buffers are transmitted, which may affect latency depending on the selected size and data rate.  To ensure optimal performance and minimal latency, the following buffer sizes are recommended for different streaming rates:   * 1 << 4 (16 bytes): up to 50 kHz * 1 << 6 (64 bytes): 50 kHz to 100 kHz * 1 << 11 (2048 bytes): 100 kHz to 250 kHz * 1 << 17 (131,072 bytes): 250 kHz to 1 MHz | | |
| **Function specific parameters** | | |
| In | Stream  (c & c#) | The stream ID variable (initialized by the OpenStream function) |
| Buffer  (c & c#) | The buffer that the retrieved binary streaming data is stored to |
| Count  (c & c#) | The maximum number of bytes that has to be read (the number has to be equal or less to the buffer’s size) |
| bufferSize  (python only) | Buffer’s size (see description above) |
| Out | buffer (python only) | The buffer that the retrieved binary streaming data is stored to |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| int ReadStream(void\* *stream*, uint8\_t \* *buffer*, int *count,* int *bufferSize*) | | |
| **C#-DLL call** | | |
| int Read\_Stream(IntPtr stream, IntPtr buffer, int buffersize) | | |
| **Python** | | |
| bytearray buffer = [idsStream].readRaw(int bufferSize) | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| / | | |

|  |  |  |
| --- | --- | --- |
| **DecodeStream** Decodes the binary data that has been retrieved by the ReadStream function into the final position data including an error bit for each axis. The return value is the number of bytes that were finally decoded. Note that the decodeStream function can only decode full data frames. Therefore, trying to decode less samples than a full frame contains returns 0 samples. IDS frames hold up to 1023 position samples, depending on streaming rate and number of channels. To allow decoding arbitrary "samplesToReceive" number of samples, we simply overallocate the channel buffers by 1022 samples. This is shown in the examples. | | |
| **Function specific parameters** | | |
| In | stream | The stream ID variable (initialized by the OpenStream function) |
| buffer | The buffer that the retrieved binary streaming data was stored to |
| bufferSize | Buffer’s size |
| sampleBuffer 1-3 | Target buffer for the decoded position data from axis 1 - 3 |
| errorBuffer 1-3 | Target buffer for the decoded error bit data from axis 1 - 3 |
| destinationBuffersSize | Target-buffer’s size |
| bytearray buffer  (python only) | List containing binary data from stream |
| Out | decodedSamplesCount | Number of decoded position samples (please be aware of the minimum frame size) |
| axis lists  (python only) | List containing the position data for the selected measurement axes |
| error lists (python only) | List containing the error bit data for the selected measurement axes |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| int DecodeStream(void\* stream, const uint8\_t\* buffer, int bufferSize, int64\_t\* sampleBuffer1, int64\_t\* sampleBuffer2, int64\_t\* sampleBuffer3, uint8\_t\* errorBuffer1, uint8\_t\* errorBuffer2, uint8\_t\* errorBuffer3, int destinationBuffersSize, int\* decodedSamplesCount) | | |
| **C#-DLL call** | | |
| int Decode\_Stream(IntPtr stream, byte[] buffer, int buffersize, long[] sampleBuffer1, long[] sampleBuffer2, long[] sampleBuffer3, byte[] errorBuffer1, byte[] errorBuffer2, byte[] errorBuffer3, int destinationBuffersSize, int\* decodedSamplesCount) | | |
| **Python** | | |
| Int decodedBytes, list axis0, list axis1, list axis2, list err0, list err1, list err2 = [idsStream]. decodeBuffer(bytearray buffer) | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| / | | |

|  |  |  |
| --- | --- | --- |
| **StartStreamRecording** This function saves the retrieved data (note that the ReadStream function has to be executed to retrieve data) directly into a .aws file. The .aws file contains the binary data stream. It can be decoded later by the WAVE Export tool (please refer to the WAVE user manual) or using the decodeStram function (note that in that case the header information has to be removed first). | | |
| **Function specific parameters** | | |
| In | stream  (except python) | The stream ID variable (initialized by the OpenStream function) |
| path | Path and name of the resulting .aws file |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| bool StartStreamRecording(void\* *stream*, const char \* *path*) | | |
| **C#-DLL call** | | |
| bool Start\_StreamRecording(IntPtr stream, string path) | | |
| **Python** | | |
| [idsStream].startRecording(string path) | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| / | | |

|  |  |  |
| --- | --- | --- |
| **StopStreamRecording** Stops the recording started by StartStreamRecording. | | |
| **Function specific parameters** | | |
| In | stream  (except python) | The stream ID variable (initialized by the OpenStream function) |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| bool StopStreamRecording(void\* *stream*) | | |
| **C#-DLL call** | | |
| bool Stop\_StreamRecording(IntPtr stream) | | |
| **Python** | | |
| [idsStream].stopRecording() | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| **/** | | |

|  |  |  |
| --- | --- | --- |
| **Read** Reads and decodes position data in buffer. This function is only available within python and can be seen as a ready-made combination of the ReadStream and DecodeStream function. | | |
| **Function specific parameters** | | |
| In | bufferSize | Size of buffer to be created in Bytes |
| Out | decodedBytes | Number of decoded position samples (please be aware of the minimum frame size) |
| axis lists | List containing the position data for the selected measurement axes |
| error lists | List containing the error bit data for the selected measurement axes |
| **JSON Method** | | |
| / | | |
| **C-DLL call** | | |
| / | | |
| **C#-DLL call** | | |
| / | | |
| **Python** | | |
| Int decodedBytes, list axis0, list axis1, list axis2, list err0, list err1, list err2 = [idsStream]. read(int bufferSize) | | |
| **Matlab** | | |
| / | | |
| **LabVIEW** | | |
| / | | |



attocube systems GmbH

Eglfinger Weg 2

85540 Haar, Germany

Phone: +49-89 -420 7970

E-Mail: info@attocube.com  
www.attocube.com

**For technical queries, contact:**

support@attocube.com

**Reginal US Sales Representatives:**

West Coast: Arcadia, CA  
Phone: +1 510 631 2154

Southwest: Henderson, NV  
Phone: +1 510 631 1784

Midwest: Delaware, OH  
Phone: +1 510 631 133

attocube systems GmbH

Eglfinger Weg 2

85540 Haar, Germany

Phone: +49-89 -420 7970

E-Mail: info@attocube.com  
www.attocube.com

**For technical queries, contact:**

support@attocube.com

**Reginal US Sales Representatives:**

West Coast: Arcadia, CA  
Phone: +1 510 631 2154

Southwest: Henderson, NV  
Phone: +1 510 631 1784

Midwest: Delaware, OH  
Phone: +1 510 631 133

attocube systems GmbH

Eglfinger Weg 2

85540 Haar, Germany

Phone: +49-89 -420 7970

E-Mail: info@attocube.com  
www.attocube.com

**For technical queries, contact:**

support@attocube.com

**Reginal US Sales Representatives:**

West Coast: Arcadia, CA  
Phone: +1 510 631 2154

Southwest: Henderson, NV  
Phone: +1 510 631 1784

Midwest: Delaware, OH  
Phone: +1 510 631 133

attocube systems GmbH

Eglfinger Weg 2

85540 Haar, Germany

Phone: +49-89 -420 7970

E-Mail: info@attocube.com  
www.attocube.com

**For technical queries, contact:**

support@attocube.com

**Reginal US Sales Representatives:**

West Coast: Arcadia, CA  
Phone: +1 510 631 2154

Southwest: Henderson, NV  
Phone: +1 510 631 1784

Midwest: Delaware, OH  
Phone: +1 510 631 133