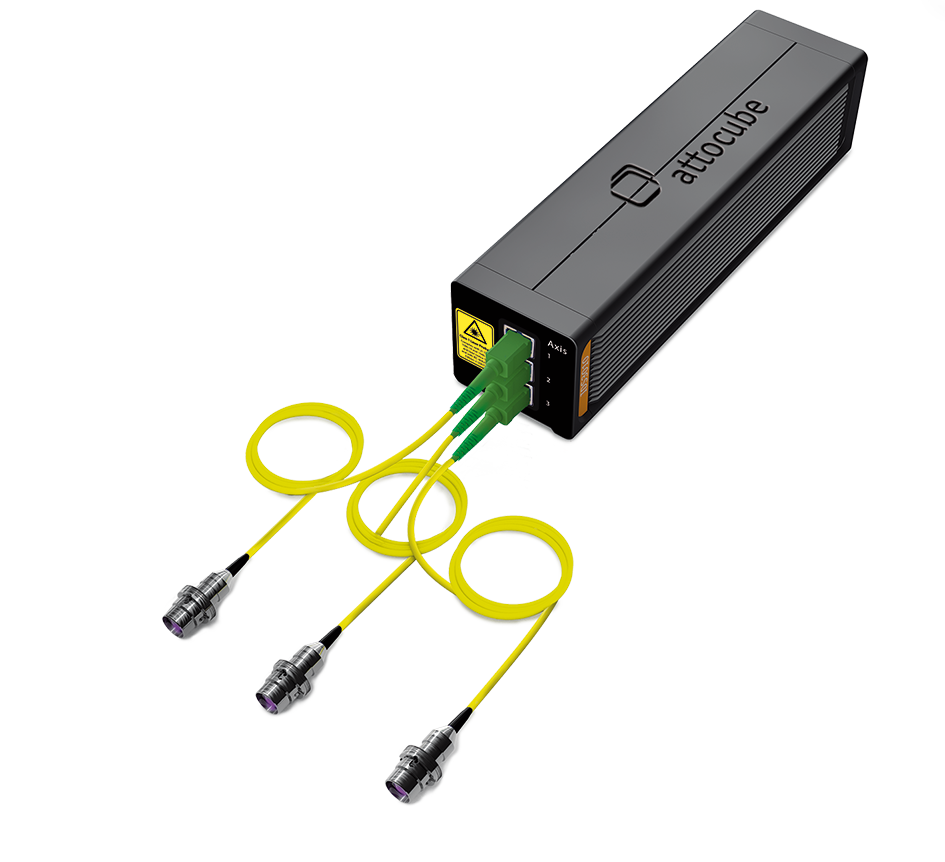


attoSENSORICS

Ultra Precision Sensors

**Interface Manual**

Interferometric Displacement Sensor (IDS)



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Manual version: | 1.7.0 |
| Manual modified: | 3/25/2021 |
|  |  |
| Firmware version | 1.7.0 |
| C# DLL version | 1.6.1 |
| C DLL version | 1.7.0 |
| WAVE Version | 1.3 |
| **Interface Manual**  Interferometric Displacement Sensor (IDS) | |
| attocube systems AG Eglfinger Weg 2 D - 85540 Haar, Germany  Phone: +49 89 - 4207 97 0  Fax: +49 89 - 4207 97 20 190  E-Mail: [info@attocube.com](mailto:info@attocube.com)  [www.attocube.com](http://www.attocube.com)  **For technical queries, contact:**  [support@attocube.com](mailto:support@attocube.com) | |

# Table of Contents

[I. Table of Contents 3](#_Toc67554333)

[II. Real-time interfaces 4](#_Toc67554334)

[II.1. Introduction 4](#_Toc67554335)

[II.2. Digital interfaces 5](#_Toc67554336)

[II.2.a. HSSL 5](#_Toc67554337)

[II.2.b. AquadB 6](#_Toc67554338)

[II.3. Analog interfaces 9](#_Toc67554339)

[II.3.a. Sine-Cosine 9](#_Toc67554340)

[II.3.b. Linear Analog Output 11](#_Toc67554341)

[II.4. Anti-Aliasing Filter 23](#_Toc67554342)

[II.5. Error signal 24](#_Toc67554343)

[II.6. Optional: BiSS-C 26](#_Toc67554344)

[II.6.a. Overview 26](#_Toc67554345)

[II.6.b. Interface connector 26](#_Toc67554346)

[II.6.c. Interface configuration 28](#_Toc67554347)

[II.6.d. Protocol description 29](#_Toc67554348)

[II.7. Real-time interface connector 30](#_Toc67554349)

# Real-time interfaces

## Introduction

|  |  |
| --- | --- |
| Overview | 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) * Sine-Cosine (analog signal, LVTTL or LVDS error signal) * Linear Analog Output (analog signal, LVTTL or LVDS 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.  The settings can be configured via the IDS web interface in the Interface tab (see Figure 1: Setting the output mode in the IDS web interface) or via the available software communication interfaces. |
| Web interface setting | Figure 1: Setting the output mode in the IDS web interface |

## Digital interfaces

### HSSL

|  |  |
| --- | --- |
| Overview | In HSSL (High Speed Serial Link) mode, the absolute current displacement value is periodically transferred using a binary serial format based on 1 pm resolution. The HSSL protocol is defined by its resolution, clock time and gap. 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 and the high resolution tab sets the final bit. 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 clock time is limited by 25 MHz, equal to 40 ns. For real-time outputs it is recommended to use a DIO or DAQ card, which is compatible to a multiple of 40 ns. The clock period, gap and number of bits can be user adjusted by the web interface. Between two sets of position information, synchronization of reader/sender can be gained through a continuous stream of position information. The data are synchronized with the rising edge of the CLK signal.  The signals are available with two different signaling standards (only one at a time):  • Single-ended with LVTTL levels  • Differential with LVDS levels  A third signal (POS\_ERROR) indicates an error condition when the position is lost due to laser beam interruption. |
| Web interface setting | The HSSL signal starts with a zero displacement value as soon as the calibration process is done and the absolute position is shown on the Operation tab. Figure 2 shows an exemplary HSSL setting.  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) | “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  *Resolution = 2^*(Resolution HSSL Low) *\* 1 pm*  The lowest value is 0. |
| Resolution HSSL High | “Resolution HSSL High” specifies the highest bit to be used of the 48-bit distance value. The highest value is 47. The HSSL word size is therefore given by  Word Size = Resolution HSSL High – Resolution HSSL Low +1 |
| 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 in integer nanosecond multiples of 40 ns. |
| 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. |
| Total length of signal and maximum frequency | The total HSSL length of the signal is defined by the formula:  (Word Size + Number of gap bits) \* clock time = total HSSL length of signal.  The achievable position update rate (in Hz) is the inverse of the HSSL length.  The maximum total HSSL length of signal is 2.00 ms (500 Hz). |
| Example trace of the HSSL signal | 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 represents the time trace of the data channel and the clock channel, respectively. Each word is separated by 3 omitted clock bits. Z:\Production\SEN\Product Information\04 IDS\Manual\Vektorgrafiken\HSSL.png Figure 3: Signal pattern of a HSSL transfer |
| Applying the HSSL settings | To apply the settings, press the orange Apply button (see Figure 2). A green floppy disk (see Figure 4) indicates that the parameters were successfully stored.  Figure 4: Saving the HSSL settings |

### AquadB

|  |  |
| --- | --- |
| Overview | In contrast to HSSL, the AquadB protocol is an incremental protocol that only transfers information about the change of the measured position. Using this method higher bandwidths are achievable compared to HSSL. 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 encoded over two different 2-level-channels, A and B. The levels’ amplitude is either LVTTL or LVDS (one at a time):   * Single-ended with LVTTL levels * Differential with LVDS levels   A measured position change of size *Resolution* in the IDS results in a state change of either A or B, depending on the change direction. This is sometimes also referred to X4-Encoding. This way, the AquadB interface allows transmitting both the displacement and direction of the target movement.  Figure 5 illustrates how to calculate the current position from the A/B signals. 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 5: Signal pattern of a AquadB transfer  A third signal POS\_ERROR uses the same definition as the HSSL format.  Another way of visualizing the AquadB protocol is to plot Channel A and B against each other, which creates a square. In Figure 6, a signal change of 90° represents, depending of the direction, a displacement of ± Resolution. Going counterclockwise represents an increment in position (B-leads-A) and going clockwise represents a decrement in position (A-leads-B). From this way of visualizing the AquadB protocol it also becomes clear that only one channel can change its state at a time.    Figure 6: Interpretating the AquadB signal lines as a square pattern |
| Resolution and clock | The scaling and the update rate of the A/B outputs are defined by *Resolution* and *Clock* (see Figure 7).  *Resolution*  The resolution defines the position resolution of the real-time interface. The resolution of the AquadB interface is user adjustable and ranges from 1 pm to 64.93 nm (programmable in 2^*n* steps, where *n* is an integer number). We recommend to start to use a high resolution for the first test, afterwards you can decrease the resolution and adjust it to the application.  *Clock*  The clock defines the minimal 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 10 240 ns (i.e. 98 kHz). The clock is programmable in integer multiples of 40 ns.    Figure 7: Defining the AquadB settings in the IDS web interface |
| Maximum target velocity | The maximum velocity the AquadB protocol can represent is the position change per state change of A/B (*Resolution*) divided by the frequency at which the A/B channels are able to change their state (*Clock*).  maximum velocity = *Resolution* / *Clock*.  As a result, when choosing the right *Resolution*, there is always a trade-off between the maximum velocity the AquadB protocol can transmit and the resolution of the transmitted position. Choosing the right *Clock* is simpler, as there are no negative side-effects when just choosing the fastest possible rate the receiving system (e.g. a DAQ-Card or a Counter) can handle, other than resulting in larger datasets.  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:*** The combination of *Resolution* and *Clock* parameters defines a type of low pass filter.  The combination of *Resolution* and *Clock* parameters defines a speed limit of  maximum speed = *Resolution* / *Clock*.  If the target velocity exceeds that limit, the AquadB interface cannot follow and an error signal is raised. To clear the error, the position has to be reset or a calibration has to be started. |
| 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 activate. |

## Analog interfaces

### Sine-Cosine

|  |  |
| --- | --- |
| Overview | Sinus-Cosine uses two analog signals in quadrature. Levels are chosen according the following definition:   * Differential signal 1 Vpp +/- 10% * Common Mode voltage: 2 V +/- 5% * R\_SOURCE = 30 Ohm * 7 mA ≤ I\_LOAD ≤ 20 mA * Max. output bandwidth 25 MHz   Each channel increment/decrement represents a signal change of a value *Resolution*. A position change of size *Resolution* results in an increment or decrement of the signals A and B, depending on the change direction. This allows the determination of the signal direction and the value with a step unit of *Resolution*. The *Clock* parameter defines the maximum frequency at which the incremental change of the signal can be outputted. Figure 8 shows how the settings are applied within the IDS web interface.    Figure 8: Sine Cosine real-time interface settings  A third signal POS\_ERROR uses the same definition as the HSSL format. In this mode, only a change in distance can be displayed by means of an analog quadrature signal pair. The scaling and the update rate of the sin/cos output is configurable. The fastest possible update interval is 40 ns. The resolution specifies the distance corresponding to a 90-degree rotation at the sin/cos output.  Signal pattern:  C:\Users\matheus\Desktop\oszi.png  Figure 9: Sine-Cosine 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).  The SIN/COS outputs are controlled by D/A converters with a word length of 12 bit, providing 2^12 = 4096 incremental voltage steps. |
| Resolution Sin-Cos | The largest Sine-Cosine interface resolution (see Figure 10) is 16.8 µm/90° (corresponding to a resolution range of 24 bits given in pm units), the smallest resolution is 1 pm.    Figure 10: Sine-Cosine signal resolution  The Sine/Cosine output is controlled by a 12-bit D/A converter. Therefore, the resolution for a whole sine/cosine period (360°) contains 212 = 4096 steps.  For example setting the position resolution to 5000 nm for a 90° turn of the sine/cosine 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. |

### Linear Analog Output

|  |  |
| --- | --- |
| Overview | The Linear Analog Output mode (see Figure 11) returns a voltage level, which is proportional to the relative performed displacement. 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.  In general, this output provides two different functions, “Displacement” and “Vibrometry”. On one hand, the displacement is directly represented by the given voltage, on the other hand 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 regularly frequency analysis can be performed up to 5 MHz, with respect to the Nyquist-theorem.  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. The output is quantized by a 12 bit bipolar Digital Analog Converter (DAC) and is principally decoded like HSSL (Chapter II.2.a). 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.    Figure 11: Using the Linear Analog Output mode |
| Resolution and range | The resolution for the output is given by the formula  Since the outputed voltage level is converted by 12 bit, the minimum resolution of 1 pm can only be achieved for . It’s recommendalbe to readout 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. The table below (see Figure 12) indicates all possible results and parameters in terms of linear analog configuration.    Figure 12: Signal range and resolution configurations for the linear analog output in accordance to 0..34 different HSSL configurations that are used for the Linear Analog Output    Figure 13: 0..34 different HSSL configurations that are transferred to the Linear Analog Output |
| Apply settings | The value for N can directly be typed into the corresponding box (Figure 14). N is limitted between 0 and 34, following an integer row. For example, if N = 7 a range of has been applied to the output, the resolution for the DAC equals 128pm (see Figure 12).    Figure 14: Applying the Linear Analog Output settings  The range can be confirmed by pushing the “Apply” button. |
| Determining full range | The linear analog feature provides an additional window in the Interface tab to activate a test signal (see Figure 15). The output is based on a rectangular signal to fill and determine the full output range (see Figure 16).    Figure 15: Activating the test channel  By setting the check mark to the box, the test signal is transferred to Pin 21(+) and 8(-) (see Chapter II.7 Real-time interface connector).    Figure 16: Oscilloscope screenshot of the activated test channel that is used to determine the full range |
| Determining offset | As displayed, the signal defines the boundaries for the Linear Analog Output range. Figure 16 shows a full range of 749 mV. The yellow line shows the level of the positive pin (21) from the realtime output. After a calibration, the signal of the linear analog will be centered around the offset of the voltage output. The offset is simply calculated as:  which is for this example. |
| Differential mode | Since linear analog can also be used in differential mode, the levels of Pin 21(+) and 8(-) can be subtracted to evaluate the full range [V] for differential usage (see Figure 17). |
|  | 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. |
| 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 Figure 18), which is attributed to the characteristics of the DAC itself. Of course, theoretically the full range can also be determined by the size of thoses jumps.    Figure 18: Exceeding the Linear Analog Output range  In Figure 18, this artificial discontinuity of the linear analog range is displayed. If the “reset axes” command is executed, the signal of the Linear Analog Output will be centered around the general signal offset. 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 acitvely excited by a waveform generator with a triangle voltage, according to the principal test setup in the picture above. In this scenario N = 10 and as an appropriate data analysis tool a LabVIEW based software in combination with a realtime National Instruments Data Acquisition (DAQ)- Card was used.  If the maximum range is not well aligned, the signal cosequently delivers jumps (see Figure 20).    Figure 20: Signal jumps due to exceeding the Linear Analog Output range  It appeals to execeute a reset axes command to center the signal around the offset, when the votlage exceedts it’s boundaries on first glance. If the oscillation is still to high, the measurement needs to be stopped and the range might be changed in terms of a higher maximum range. |
| Range-conform displacement measurements | Figure 21 shows a triangle oscillation of a piezo scanner within the predefined boundaries of the Linear Analog Output, there are no more jumps distorting the displacement information. The range of the ouput 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. In the end it is desireable to achieve a good workload of the analog range according to signal to noise, because the DAC only quantizes 4095 steps within the whole range.  Figure 21: Range-conform displacement measurement |
| Vibrometry | Since signal jumps in the position signal lead to artifacts in spectral data analysis, it’s a major point to avoid those disturbances (sub chapter: Exceeding full range) by means of determining the full output range [V] correctly. The resolution also follows the DAC bit configuration from the table in the former sub chapter.    Figure 22: Vibrometry setting  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. The filter function is built by a cascaded first order two filter block. In general, the high pass cut off frequency is defined as the -3 dB point in high pass filtering (pass band) and is given by the formula  where . All values are listed in the table below (see Figure 23). The pass band decreases with a slope of -20 dB/decade.    Figure 23: Defining the high pass cut-off frequency in the Linear Analog Vibrometry setting |
| Vibrometry example | In general the measured field is reduced by a factor of 0.71 at the -3 dB pass band point. This was exemplary verified for the high pass cut off frequency at 195 Hz. A schematic of the principal setup can be found in Figure 24.  Aufbau_vib  Figure 24: Vibrometry example setup  The first screenshot indicates an measurement amplitude of 8.18 mV at an oscillation frequency of 194 Hz. The applied N in this case was 24 to gain the full spectral information (see Figure 25).    Figure 25: HF2LI result for N=24  In the next step the cut off frequency was set to N = 13 which equals 195 Hz. Now the measured oscillation is supposed to attenuate by a factor of 0.71, since the excitation is directly set into the pass band point (see Figure 26).    Figure 26: HF2LI result for N=13  After the cut off frequency is set, the amplitude reduces to 5.79 mV. The relation between both values reveals a factor of 0.707. This relation can also be verified for other pass band frequencies. |
| Vibrometry measurement within boundaries | As already mentioned it’s mandatory to avoid jumps within in the position signal to prevent spectral data analysis from disturbing artifacts. If the adjustment of the boundaries was succesful, the position time signal can be transferred into the corresponding frequency domain. In this example the data have been acquired by a National Instruments DAQ Card with a further Fast Fouriér Transformation (FFT) post processing step. The pictures below show a measurement based on the output of the Linear Analog Output (see Figure 27). The setup refers to the one, which is shown on the previous page. For this test N was set to 6 and the high pass cut off frequency to 12.5 kHz. The waveform generator excited an oscillation frequency of 200 kHz.    Figure 27: Sampled output of the Linear Analog Output    Figure 28: FFT result of the sampled Linear Analog Output data  The linear analog signal was sampled with 1 MHz to perform FFT analysis up to 500 kHz (see Figure 28). In another example, the linear analog was directly given to a HF2LI Zurich Instruments Lock in (see Figure 29).    Figure 29: HF2LI result for FFT analysis |
| Differential measurements with the Linear Analog Output | The Linear Analog Output also provides differential signals. The advantage of differential measurement is noise and overshoot reduction. In principal both channels (1,2,3(+) and 1,2,3(-) ) of the corresponding axis are substracted by a simple math function (see Figure 30), as it was also explained in the previous chapter for Sine-Cosine. |
|  | Figure 30: Interpretating the Linear Analog Output signal at differential transfer  Figure 30 shows a snapshoot of a scope indicates the function of differential measurements. In this case a low frequent oscillation was excited. The yellow line shows the positive signal pin, the blue line shows the inverted negative measurement pin. According to every axis, all pin configurations can be found in chapter 0. |
| **Detector Signal during Alignment** | Figure 31: Detector signal routed to Linear Analog output during Optical Alignment  The detector (Figure 31) signal 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 assigned test channel, first. Comparing this signal to the indicator bar in the web interface reveals, that the Linear Analog output is of course much faster. The feedback from adjusting a target with Linear Analog is more precise and free of any quantization, like it’s used in the web interface’s alignment bar algorithm. |

## Anti-Aliasing Filter

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 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 following table:   |  |  | | --- | --- | | Output mode |  | | Linear Analog |  | | HSSL |  | | SinCos/ AquadB |  |   The filter window can be selected between:   * Rectangular * Cosine * Cosine^2 * Hamming * Raised Cosine   The attenuation of the filter can be varied between 3 and 30 dB. A detailed function description can also be found in the related header file for C and help.chm for C#. |

## 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. |
| Voltage level | The voltage level of the error signal corresponds to that of the selected interface. For the Sine Cosine signal, which is only provided as differential signal, the interface tab provides two possible signal levels for the error signal, either LVDS or LVTTL (see Figure 32).  Figure 32: LVTTL and LVDS options are available |
| Error situation: signal too high (overload) | If the interference signal saturates the detector an overload situation is recognized and indicated by the error signal (see Figure 33).    Figure 33: Overload detection situation via the error signal |
| Error situation: signal too low | The error signal indicates the case that the interference signal went below the minimum threshold (see Figure 34)  Figure 34: Signal loss detection situation via the error signal |
| Error situation: beam interruption | The error signal indicates a beam interruption (Figure 35)    Figure 35: Beam interruption detected via the error signal |

## Optional: BiSS-C

### Overview

|  |  |
| --- | --- |
|  | The IDS can optionally be upgraded with the BiSS-C in the point-to-point configuration. 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 at the 14 pin GPIO connector. 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 (MA) clock frequency is supported up to a frequency of 10 MHz. The high-speed data path is clocked by the incoming master clock “MA”. 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. |

### Interface connector

|  |  |
| --- | --- |
| Overview | The BiSS-C signals are provided at the 14 pin GPIO connector. A matching connector is the Honda HDR-E14MAG1+ (see Figure 36).    Figure 36: Honda HDR-E14MAG1+ connector  The scope of delivery includes one GPIO interface cable with 14 open ended wires (see Figure 37). Each wire is numberes according to the pinout table provided below.  Beschreibung: Z:\Production\SEN\Work Instructions\Arbeitsanweisungen\IDS GPIO Kabel\P1030443.JPG  Figure 37: BiSS-C connector cable as part of the BiSS-C IDS delivery scope |
| Pinout | Figure 38 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).    Figure 38: BiSS-C connector pin layout |
| Factory reset | The IDS system provides the possibility to initiate a factory reset through the GPIO connector. During the boot-process of the IDS system, a test-pattern is sent to the UART.TX port. If the user has set up a connection from UART.TX to UART.RX (“factory reset dongle”), the IDS system receives the test-pattern and executes a factory reset.  A factory reset consequently requires the short-circuit of the following the pins:  DataOut1\_p 🡪 ClockIn1\_p  DataOut1\_n 🡪 ClockIn1\_n |
| M00 | During the boot procedure of the IDS, a test patter is sent at 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! |

### Interface configuration

|  |  |
| --- | --- |
| Overview | The BiSS-C interface can be configured in the Interface tab of the web interface (see Figure 39).    Figure 39: Configurating the BiSS-C interface |
| Resolution and range | “Resolution BiSS-C” defines the position resolution of the BiSS-C interface. It allows specifying the lowest bit of the 32 bit distance value in units of bits. Those 32 bits are taken out of the full 48 bit position word. The position resolution is therefore given by  *Resolution = 2^*(Resolution BiSS-C) \* 1 pm  The lowest value is 0.  The given resolution, as well as maximum range can be found in the table below.    Figure 40: Range and resolution according to 0..16 different bit-configurations  This corresponds to the following bit configurations:    Figure 41: 0..16 different bit-configurations |

### Protocol description

|  |  |
| --- | --- |
| Overview | The supports the BiSS-C point-to-point configuration. The word length is 32 bit. The resolution can be set by the user in the web interface. 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 displacement 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). |

## Real-time interface connector

|  |  |
| --- | --- |
| Overview | The IDS electronic interface has a female 26 pin HDR connector which transports the real-time output signals. Figure 42 shows the front view of the female HDR connector:    Figure 42: Front view of the female HDR connector  The IDS is also delivered with a 26 pin IDC type male connector (HDR-E26 MAG1+), which can be used to finish a real-time output cable:    Figure 43: HDR-E26 MAG1+ male connector that can be used to finish a cable  It is possible to order a prefabricated cable with open ends (three quad cables (1,2, and 3) and six twisted pairs), see Figure 44.  C:\Users\hirschmann\Documents\IDS\PIN Layout\Stecker_offen.JPG  Figure 44: Real-time output cable with open ends |

|  |  |
| --- | --- |
| Proper termination | In order to reduce the noise level of the real-time outputs, the signal output can be terminated with a proper termination impedance.  The Sin/Cos output is optimized for a differential termination impedance of 120 Ω. This termination results in a voltage level of 1 V. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **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) | | GPIO-RT | | General purpose IO, always LVTTL levels (same circuit as on GPIO) | | | | | | | | | | |  |  |   **Pin-description:** |

# 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. In particular, these are:

• 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. A 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 the 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 a fast and easy implementation in National Instrument’s LabVIEW environment.

### Matlab Library

Based on the JSON interface, we offer ready-made Matlab functios to have a fast and easy implementation in Mathwork’s Matlab 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 the 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.ids.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.    Sending Json-Rpc commands in the command line interface. |

### 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:  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) |
| **Establing 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). |
| **Establing 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.

|  |  |
| --- | --- |
| **Establing 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** |
| **Establing 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[BUFFER\_SIZE];  system\_errorNumberToString(device, 0, ret, errorNameBuf, BUFFER\_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 the variable "err"  string errmsg = attoDevice .ErrorNumberToString(0, err);  // converts "err" into the corresponding errormessage 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 picture below ). 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 a following VI, this VI still is executed. |
|  | Example for a “real” error  Facing a negative error code value, a “real” error is indicated (see picture below) Therefore, also the status is set on red, which means that the error is active. When an active error is inputted in a following VI, the VI will not execute its function. |

# Functions

## Adjustment

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

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| enable | true = enabled; false = disabled |

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

|  |
| --- |
| method: com.attocube.ids.adjustment.getAdjustmentEnabled |
| params: [] |
| Result: [errNo, enable] |

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

|  |
| --- |
| int **IDS\_adjustment\_getAdjustmentEnabled**(int deviceHandle, bool\* enable) |

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

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

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

|  |
| --- |
| [enable] = **IDS\_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 basline (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 needs to be added, otherwise an overload can  occure without being recognized. Example: baseline 15permille and contrast 850permille. Just  from the contrast it looks good. Adding the baseline the overload is visible. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to get the value from [0..2] |
| Out | warningNo | Warning code, can be converted into a string using the errorNumberToString function |
| contast | Contrast of the base band signal in permille |
| baseline | Offset of the contrast measurement in permille |
| mixcontrast | lower contrast measurment when measuring a mix contrast (indicated by error code) |

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

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

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

|  |
| --- |
| int **IDS\_adjustment\_getContrastInPermille**(int deviceHandle, int axis, int\* warningNo, int\* contast, int\* baseline, int\* mixcontrast) |

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

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

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

|  |
| --- |
| [warningNo, contast, baseline, mixcontrast] = **IDS\_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 | errNo |

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

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

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

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

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

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

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

|  |
| --- |
| [] = **IDS\_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 | errNo |

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

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

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

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **getMasterAxis** Returns the master axis (for more information please refer to the IDS User Manual).  ADD MORE INFO |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| masteraxis | Axis which is operating as masteraxis [0..2] |

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

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

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

|  |
| --- |
| int **IDS\_axis\_getMasterAxis**(int deviceHandle, int\* masteraxis) |

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

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

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

|  |
| --- |
| [masteraxis] = **IDS\_axis\_getMasterAxis**() |

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| mode | 0 = single; pass 1 = dual pass |

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

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

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

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **setMasterAxis** Sets the master axis (for more information please refer to the IDS User Manual). |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis which is operating as masteraxis [0..2] |
| Out | errNo | errNo |

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

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

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

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

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

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

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

|  |
| --- |
| [] = **IDS\_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. |

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

|  |  |  |
| --- | --- | --- |
| In | mode | 0 = single pass; 1 = dual pass |
| Out | errNo | errNo |

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

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

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

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

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

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

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

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

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

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

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

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

## Displacement

|  |
| --- |
| **getAbsolutePosition** The absolute position information is estimated at the measurement initialization procedure.  This initial absolute position information is not updated during system  operation, whereas the IDS continuously measures the relative displacement from  that point. This function outputs the sum of the static absolute position register value  and the continuously updated displacement register value for a single axis. |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **getAbsolutePositions** The absolute position information is estimated at the measurement initialization  procedure. This initial absolute position information is not updated during system  operation, whereas the IDS continuously measures the relative displacement from  that point. This function outputs the sum of the static absolute position register value  and the continuously updated displacement register value for all three axes. |

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

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

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

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

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

|  |
| --- |
| int **IDS\_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] = **IDS\_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 refere to the manual for a table with stopband and 3dB cut-off frequency. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| averageN | A value from 0 to 24 |

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

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

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

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

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

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

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

|  |
| --- |
| [averageN] = **IDS\_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.ids.displacement.getAxesDisplacement |
| params: [] |
| Result: [warningNo, displacement0, displacement1, displacement2] |

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

|  |
| --- |
| int **IDS\_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] = **IDS\_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.ids.displacement.getAxisDisplacement |
| params: [axis] |
| Result: [warningNo, displacement] |

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

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

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

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

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

|  |
| --- |
| [warningNo, displacement] = **IDS\_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 alignment contrast (peak-to-peak of the basic  interference signal amplitude) and the basline (its offset) during a running  measurement. 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 IDS high accuracy initialization to obtain  correct values directly after measurement initialization. When using the quick  initialization, the initial value might be falsified 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 ‰ |
| baseline | Offset of the contrast measurement in ‰ |

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

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

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

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

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

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

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

|  |
| --- |
| [warningNo, contrast, baseline] = **IDS\_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 | errNo |
| enable | true = enabled; false = disabled |

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

|  |
| --- |
| method: com.attocube.ids.displacement.getMeasurementEnabled |
| params: [] |
| Result: [errNo, enable] |

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

|  |
| --- |
| int **IDS\_displacement\_getMeasurementEnabled**(int deviceHandle, bool\* enable) |

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

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

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

|  |
| --- |
| [enable] = **IDS\_displacement\_getMeasurementEnabled**() |

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

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

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

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

|  |
| --- |
| **getReferencePosition** The reference position information is estimated at the measurement initialization procedure or on reset.  This initial absolute position information is not updated during system  operation, whereas the IDS 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.ids.displacement.getReferencePosition |
| params: [axis] |
| Result: [warningNo, position] |

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

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **getReferencePositions** The reference position information is estimated at the measurement initialization procedure or on reset.  This initial absolute position information is not updated during system  operation, whereas the IDS 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 | position of the axis 0 in pm |
| position1 | position of the axis 1 in pm |
| position2 | position of the axis 2 in pm |

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

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

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

|  |
| --- |
| int **IDS\_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] = **IDS\_displacement\_getReferencePositions**() |

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

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

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

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

|  |
| --- |
| **getSignalQuality** This function can be used to monitor the alignment contrast (peak-to-peak of the basic  interference signal amplitude) and the basline (its offset) during a running  measurement. 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 IDS high accuracy initialization to obtain  correct values directly after measurement initialization. When using the quick  initialization, the initial value might be falsified 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 ‰ |
| baseline | Offset of the contrast measurement in ‰ |

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

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

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

|  |
| --- |
| int **IDS\_displacement\_getSignalQuality**(int deviceHandle, int axis, int\* warningNo, int\* contrast, int\* baseline) |

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

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

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

|  |
| --- |
| [warningNo, contrast, baseline] = **IDS\_displacement\_getSignalQuality**(axis) |

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

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

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

|  |
| --- |
| getSignalQuality.vi |

|  |
| --- |
| **linProc** Important note: This function is not actively supported anymore. With IDS firmware 1.7.0, a new set of functions corresponding to the linearization procedure was introduced and effectively outdated this function.  Starts linearization procedure (additional IDS feature). |

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

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| fringesnbr | Number of fringes to be acquired |
| samplesperfringe | Number of samples per fringe |
| set | 0 = evaluate current nonlinear amplitude  1 = perform linearization and upload look up table  2 = Clear look up table  3 = Perform only calculation of Phi file |
| Out | errNo | errNo |
| lintable | String, which contains all 512 phase related correction values |
| nonlinearamp | String which contains the residual positive and negative maximal nonlinear amplitude |

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

|  |
| --- |
| method: com.attocube.ids.displacement.linProc |
| params: [axis, fringesnbr, samplesperfringe, set] |
| Result: [errNo, lintable, nonlinearamp] |

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

|  |
| --- |
| int **IDS\_displacement\_linProc**(int deviceHandle, int axis, int fringesnbr, int samplesperfringe, int set, int\* lintable, int\* nonlinearamp) |

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

|  |
| --- |
| lintable, nonlinearamp = **[dev].displacement.linProc**(axis, fringesnbr, samplesperfringe, set) |

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

|  |
| --- |
| [lintable, nonlinearamp] = **IDS\_displacement\_linProc**(axis, fringesnbr, samplesperfringe, set) |

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

|  |
| --- |
| Tuple<int,int> value = [Device].**Displacement\_LinProc**(int axis, int fringesnbr, int samplesperfringe, int set) |

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

|  |
| --- |
| linProc.vi |

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

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

|  |  |  |
| --- | --- | --- |
| In | value | AverageN value from 0 to 24 |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.displacement.setAverageN |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_displacement\_setAverageN**(int deviceHandle, int value) |

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

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

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

|  |
| --- |
| [] = **IDS\_displacement\_setAverageN**(value) |

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

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

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

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

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

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

|  |
| --- |
| [] = **IDS\_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 **IDS\_ecu\_enable**(int deviceHandle) |

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

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

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

|  |
| --- |
| [] = **IDS\_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 if connected, false if not |

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

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

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

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

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

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

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

|  |
| --- |
| [connected] = **IDS\_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 if enabled, false if not |

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

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

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

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

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

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

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

|  |
| --- |
| [enabled] = **IDS\_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 humidity in % |

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

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

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

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

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

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

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

|  |
| --- |
| [humidity] = **IDS\_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 pressure in hPa |

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

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

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

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **getRefractiveIndex** Reads out the ECU estimated refractive index for the current  ECU readings. To get the refractive index for other modes, please see  getRefractiveIndexForCompensation. |

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

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to get the mode for. 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 mode see defintion in set function |

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

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

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

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

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

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

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

|  |
| --- |
| [mode] = **IDS\_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 this axis. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to get the refractive index for. 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 **IDS\_ecu\_getRefractiveIndexForCompensation**(int deviceHandle, int axis, double\* rIndex) |

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

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

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

|  |
| --- |
| [rIndex] = **IDS\_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 temperature in degrees C |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to set the mode for. Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| mode | can be 0 for direct ECU mode, 1 to take the manual values and calculate the refractive index from this or 2 to directly take the 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 **IDS\_ecu\_setRefractiveIndexCompensationMode**(int deviceHandle, int axis, int mode) |

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

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

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

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

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

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

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

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

## Access

|  |
| --- |
| **getLockStatus** This function returns if the device is locked and if the current client is authorized to use the device. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| locked | Is the device locked? |
| authorized | Is the client authorized? |

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

|  |
| --- |
| method: getLockStatus |
| params: [] |
| Result: [errNo, locked, authorized] |

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

|  |
| --- |
| int **IDS\_getLockStatus**(int deviceHandle, bool\* locked, bool\* authorized) |

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

|  |
| --- |
| locked, authorized = **[dev].access.getLockStatus**() |

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

|  |
| --- |
| [locked, authorized] = **IDS\_getLockStatus**() |

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

|  |
| --- |
| Tuple<bool,bool> value = [Device].**GetLockStatus**() |

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

|  |
| --- |
| getLockStatus.vi |

|  |
| --- |
| **grantAccess** Grants access to a locked device for the requesting IP by checking against the password |

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

|  |  |  |
| --- | --- | --- |
| In | password | string the current password |
| Out | errNo | errorCode |

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

|  |
| --- |
| method: grantAccess |
| params: [password] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_grantAccess**(int deviceHandle, const char\* password) |

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

|  |
| --- |
| **[dev].access.grantAccess**(password) |

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

|  |
| --- |
| [] = **IDS\_grantAccess**(password) |

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

|  |
| --- |
| void value = [Device].**GrantAccess**(string password) |

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

|  |
| --- |
| grantAccess.vi |

|  |
| --- |
| **lock** This function locks the device with a password, so the calling of functions is only possible with this password. The locking IP is automatically added to the devices which can access functions |

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

|  |  |  |
| --- | --- | --- |
| In | password | string the password to be set |
| Out | errNo | errorCode |

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

|  |
| --- |
| method: lock |
| params: [password] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_lock**(int deviceHandle, const char\* password) |

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

|  |
| --- |
| **[dev].access.lock**(password) |

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

|  |
| --- |
| [] = **IDS\_lock**(password) |

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

|  |
| --- |
| void value = [Device].**Lock**(string password) |

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

|  |
| --- |
| lock.vi |

## Manual

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to get the humidity for. 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 humidity in hPa |

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

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

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

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **getPressureInHPa** Reads out the manually configured Pressure (compensation mode 1). |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to get the pressure for. 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 pressure in hPa |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to get the mode for. 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 **IDS\_manual\_getRefractiveIndex**(int deviceHandle, int axis, double\* rindex) |

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

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

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

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

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

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

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

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

|  |
| --- |
| **getTemperatureInDegrees** Reads out the manually configured Temperature (compensation mode 1). |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to set the temperature for. 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 temperature in degree celsius |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to set the humidity for. Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| humidity | in Percent |
| 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 **IDS\_manual\_setHumidityInPercent**(int deviceHandle, int axis, double humidity) |

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to set the pressure for- Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| pressure | in 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 **IDS\_manual\_setPressureInHPa**(int deviceHandle, int axis, double pressure) |

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to set the mode for. 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 **IDS\_manual\_setRefractiveIndex**(int deviceHandle, int axis, double rindex) |

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

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

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

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

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis to set the temperature for. Parameter has to be -1 for the moment, individual axes will be supported in the next firmware release |
| temperature | in degree celcius |
| 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 **IDS\_manual\_setTemperatureInDegrees**(int deviceHandle, int axis, double temperature) |

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

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

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

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

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

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

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

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

## Nlc

|  |
| --- |
| **clearLut** Deactivates the LUT and removes it from the device |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS |
| Out | errNo | int32 Error number if one occured during deletion of the LUT |

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

|  |
| --- |
| method: com.attocube.ids.nlc.clearLut |
| params: [axis] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_clearLut**(int deviceHandle, int axis) |

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

|  |
| --- |
| **[dev].nlc.clearLut**(axis) |

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

|  |
| --- |
| [] = **IDS\_nlc\_clearLut**(axis) |

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

|  |
| --- |
| void value = [Device].**Nlc\_ClearLut**(int axis) |

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

|  |
| --- |
| clearLut.vi |

|  |
| --- |
| **createLut** Creates a new LUT for the given axis. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS where the LUT should be generated |
| Out | errNo | int32 Error number if one occured while starting the creation. Errors during creation can be pulled by getLutStatus |

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

|  |
| --- |
| method: com.attocube.ids.nlc.createLut |
| params: [axis] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_createLut**(int deviceHandle, int axis) |

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

|  |
| --- |
| **[dev].nlc.createLut**(axis) |

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

|  |
| --- |
| [] = **IDS\_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 | Axis [0|1|2] of the IDS of which the nonlinearity error is to be estimated |
| Out | errNo | int32 Error number if one occured while estimating the nonlinearity error |

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

|  |
| --- |
| method: com.attocube.ids.nlc.estimateNonlinearities |
| params: [axis] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_estimateNonlinearities**(int deviceHandle, int axis) |

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

|  |
| --- |
| **[dev].nlc.estimateNonlinearities**(axis) |

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

|  |
| --- |
| [] = **IDS\_nlc\_estimateNonlinearities**(axis) |

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

|  |
| --- |
| void value = [Device].**Nlc\_EstimateNonlinearities**(int axis) |

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

|  |
| --- |
| estimateNonlinearities.vi |

|  |
| --- |
| **getDynamicNormalization** Returns the normalization mode of a specific axis. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS of which the normalization mode is queried |
| Out | errNo | int32 Error number if an error occured while getting normalization mode |
| mode | int32 Normalization Mode 0 Dynamic normalization 1 Normalization frozen 2 Normalization mode determined by target velocity |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getDynamicNormalization |
| params: [axis] |
| Result: [errNo, mode] |

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

|  |
| --- |
| int **IDS\_nlc\_getDynamicNormalization**(int deviceHandle, int axis, int\* mode) |

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

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

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

|  |
| --- |
| [mode] = **IDS\_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 createLut or estimateNonlinearites. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS |
| Out | errNo | int32 Error number if one occured during retrieving the histogram |
| histogram | string Json dumped histogram array |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getHistogram |
| params: [axis] |
| Result: [errNo, histogram] |

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

|  |
| --- |
| int **IDS\_nlc\_getHistogram**(int deviceHandle, int axis, char\* histogram, int size0) |

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

|  |
| --- |
| histogram = **[dev].nlc.getHistogram**(axis) |

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

|  |
| --- |
| [histogram] = **IDS\_nlc\_getHistogram**(axis) |

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

|  |
| --- |
| string value = [Device].**Nlc\_GetHistogram**(int axis) |

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

|  |
| --- |
| getHistogram.vi |

|  |
| --- |
| **getLut** Returns the LUT determined by createLut (which can be applied by setLutApplied). |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS |
| Out | errNo | int32 Error number if one occured during retrieving the LUT |
| lut | string Json dumped LUT array with 512 integers |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getLut |
| params: [axis] |
| Result: [errNo, lut] |

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

|  |
| --- |
| int **IDS\_nlc\_getLut**(int deviceHandle, int axis, char\* lut, int size0) |

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

|  |
| --- |
| lut = **[dev].nlc.getLut**(axis) |

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

|  |
| --- |
| [lut] = **IDS\_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 given axis. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS of which the LUT apply rule is queried |
| Out | errNo | int32 Error number if an error occured while quering the LUT apply rule |
| apply | boolean True, if LUT is applied on this axis else False |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getLutApplied |
| params: [axis] |
| Result: [errNo, apply] |

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

|  |
| --- |
| int **IDS\_nlc\_getLutApplied**(int deviceHandle, int axis, bool\* apply) |

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

|  |
| --- |
| apply = **[dev].nlc.getLutApplied**(axis) |

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

|  |
| --- |
| [apply] = **IDS\_nlc\_getLutApplied**(axis) |

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

|  |
| --- |
| bool value = [Device].**Nlc\_GetLutApplied**(int axis) |

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

|  |
| --- |
| getLutApplied.vi |

|  |
| --- |
| **getLutStatus** Returns if a LUT is available and if warnings or errors occurred during creation. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS of which the status of the LUT should be returned |
| Out | creationWarning | int32 Error or warning number if one occured while creating the LUT, 0 in case of no error |
| status | boolean True, if a LUT exists else False |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getLutStatus |
| params: [axis] |
| Result: [creationWarning, status] |

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

|  |
| --- |
| int **IDS\_nlc\_getLutStatus**(int deviceHandle, int axis, int\* creationWarning, bool\* status) |

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

|  |
| --- |
| creationWarning, status = **[dev].nlc.getLutStatus**(axis) |

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

|  |
| --- |
| [creationWarning, status] = **IDS\_nlc\_getLutStatus**(axis) |

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

|  |
| --- |
| Tuple<int,bool> value = [Device].**Nlc\_GetLutStatus**(int axis) |

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

|  |
| --- |
| getLutStatus.vi |

|  |
| --- |
| **getNonlinearityEstimation** Returns the LUT created by estimateNonlinearities (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. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 Error number if one occured loading the LUT |
| lut | string Json dumped LUT array with 512 integers |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getNonlinearityEstimation |
| params: [] |
| Result: [errNo, lut] |

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

|  |
| --- |
| int **IDS\_nlc\_getNonlinearityEstimation**(int deviceHandle, char\* lut, int size0) |

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

|  |
| --- |
| lut = **[dev].nlc.getNonlinearityEstimation**() |

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

|  |
| --- |
| [lut] = **IDS\_nlc\_getNonlinearityEstimation**() |

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

|  |
| --- |
| string value = [Device].**Nlc\_GetNonlinearityEstimation**() |

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

|  |
| --- |
| getNonlinearityEstimation.vi |

|  |
| --- |
| **getVelocityThresholds** Returns the threshold velocity (in µm/s) for mode 2 of setDynamicNormalization. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | int32 Error number if an error occured |
| velocityOn | int32 Velocity of the target in µm/s when to switch the normalization on (default: 10 µm/s) |
| velocityOff | int32 Velocity of the target in µm/s when to switch the normalization off (default: 5 µm/s) |

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

|  |
| --- |
| method: com.attocube.ids.nlc.getVelocityThresholds |
| params: [] |
| Result: [errNo, velocityOn, velocityOff] |

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

|  |
| --- |
| int **IDS\_nlc\_getVelocityThresholds**(int deviceHandle, int\* velocityOn, int\* velocityOff) |

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

|  |
| --- |
| velocityOn, velocityOff = **[dev].nlc.getVelocityThresholds**() |

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

|  |
| --- |
| [velocityOn, velocityOff] = **IDS\_nlc\_getVelocityThresholds**() |

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

|  |
| --- |
| Tuple<int,int> value = [Device].**Nlc\_GetVelocityThresholds**() |

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

|  |
| --- |
| getVelocityThresholds.vi |

|  |
| --- |
| **setDynamicNormalization** Sets the normalization mode of a specific axis. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS of which the normalization mode should be set |
| mode | Normalization Mode  0 Dynamic normalization (default)  1 Normalization frozen (for slow target drifts)  2 Automatic alternation between mode 0 and 1 depending on the target velocity |
| Out | errNo | int32 Error number if an error occured during switching the normalization mode |

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

|  |
| --- |
| method: com.attocube.ids.nlc.setDynamicNormalization |
| params: [axis, mode] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_setDynamicNormalization**(int deviceHandle, int axis, int mode) |

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

|  |
| --- |
| **[dev].nlc.setDynamicNormalization**(axis, mode) |

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

|  |
| --- |
| [] = **IDS\_nlc\_setDynamicNormalization**(axis, mode) |

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

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

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

|  |
| --- |
| setDynamicNormalization.vi |

|  |
| --- |
| **setLut** Uploads a LUT for a specific axis (which can be applied by setLutApplied) |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS |
| lut | Json dumped LUT array with 512 integers |
| Out | errNo | int32 Error number if one occured during uploading the LUT |

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

|  |
| --- |
| method: com.attocube.ids.nlc.setLut |
| params: [axis, lut] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_setLut**(int deviceHandle, int axis, const char\* lut) |

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

|  |
| --- |
| **[dev].nlc.setLut**(axis, lut) |

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

|  |
| --- |
| [] = **IDS\_nlc\_setLut**(axis, lut) |

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

|  |
| --- |
| void value = [Device].**Nlc\_SetLut**(int axis, string lut) |

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

|  |
| --- |
| setLut.vi |

|  |
| --- |
| **setLutApplied** Sets the apply rule for the given axis |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Axis [0|1|2] of the IDS of which the apply rule should be set |
| apply | True for applying a LUT, False for disabling a LUT |
| Out | errNo | int32 Error number if an error occured while enabling or disabling a LUT |

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

|  |
| --- |
| method: com.attocube.ids.nlc.setLutApplied |
| params: [axis, apply] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_setLutApplied**(int deviceHandle, int axis, bool apply) |

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

|  |
| --- |
| **[dev].nlc.setLutApplied**(axis, apply) |

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

|  |
| --- |
| [] = **IDS\_nlc\_setLutApplied**(axis, apply) |

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

|  |
| --- |
| void value = [Device].**Nlc\_SetLutApplied**(int axis, bool apply) |

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

|  |
| --- |
| setLutApplied.vi |

|  |
| --- |
| **setVelocityThresholds** Sets the threshold velocity (in µm/s) for mode 2 of setDynamicNormalization. By default, the normalization in mode 2 is frozen for velocities below 5 µm/s and switched to dynamic mode for velocities above 10 µm/s. |

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

|  |  |  |
| --- | --- | --- |
| In | velocityOn | Velocity of the target in µm/s when to switch the normalization on (default: 10 µm/s) |
| velocityOff | Velocity of the target in µm/s when to switch the normalization off (default: 5 µm/s) |
| Out | errNo | int32 Error number if an error occured |

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

|  |
| --- |
| method: com.attocube.ids.nlc.setVelocityThresholds |
| params: [velocityOn, velocityOff] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_nlc\_setVelocityThresholds**(int deviceHandle, int velocityOn, int velocityOff) |

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

|  |
| --- |
| **[dev].nlc.setVelocityThresholds**(velocityOn, velocityOff) |

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

|  |
| --- |
| [] = **IDS\_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 | errNo |

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

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

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

|  |
| --- |
| int **IDS\_pilotlaser\_disable**(int deviceHandle) |

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

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

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

|  |
| --- |
| [] = **IDS\_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.ids.pilotlaser.enable |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_pilotlaser\_enable**(int deviceHandle) |

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

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

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

|  |
| --- |
| [] = **IDS\_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 | errNo |
| enable | true = enabled; false = disabled |

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

|  |
| --- |
| method: com.attocube.ids.pilotlaser.getEnabled |
| params: [] |
| Result: [errNo, enable] |

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

|  |
| --- |
| int **IDS\_pilotlaser\_getEnabled**(int deviceHandle, bool\* enable) |

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

|  |
| --- |
| enable = **[dev].pilotlaser.getEnabled**() |

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

|  |
| --- |
| [enable] = **IDS\_pilotlaser\_getEnabled**() |

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

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

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

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

## Realtime

|  |
| --- |
| **AafIsEnabled** Checks if the anti-aliasing filter is enabled. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| enabled | false: Anti-Aliasing Filter is disabled  true: Anti-Aliasing Filter is enabled |

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

|  |
| --- |
| method: com.attocube.ids.realtime.AafIsEnabled |
| params: [] |
| Result: [errNo, enabled] |

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

|  |
| --- |
| int **IDS\_realtime\_AafIsEnabled**(int deviceHandle, int\* enabled) |

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

|  |
| --- |
| enabled = **[dev].realtime.AafIsEnabled**() |

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

|  |
| --- |
| [enabled] = **IDS\_realtime\_AafIsEnabled**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_AafIsEnabled**() |

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

|  |
| --- |
| AafIsEnabled.vi |

|  |
| --- |
| **apply** Applies new real time settings. Necessary after JSON realtime set commands. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.apply |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_apply**(int deviceHandle) |

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

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

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

|  |
| --- |
| [] = **IDS\_realtime\_apply**() |

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

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

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

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

|  |
| --- |
| **disableTestChannel** Disables the test channel. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.disableTestChannel |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_disableTestChannel**(int deviceHandle) |

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

|  |
| --- |
| **[dev].realtime.disableTestChannel**() |

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

|  |
| --- |
| [] = **IDS\_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.ids.realtime.discard |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_discard**(int deviceHandle) |

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

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

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

|  |
| --- |
| [] = **IDS\_realtime\_discard**() |

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

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

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

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

|  |
| --- |
| **enableTestChannel** Enables the Test Channel, which can be used for estimating the maximum signal range. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | Test Channel Master Axis |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.enableTestChannel |
| params: [axis] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_enableTestChannel**(int deviceHandle, int axis) |

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

|  |
| --- |
| **[dev].realtime.enableTestChannel**(axis) |

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

|  |
| --- |
| [] = **IDS\_realtime\_enableTestChannel**(axis) |

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

|  |
| --- |
| void value = [Device].**Realtime\_EnableTestChannel**(int axis) |

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

|  |
| --- |
| enableTestChannel.vi |

|  |
| --- |
| **getAafAttenuation** Returns the current attenuation at f\_nyquist of the anti-aliasing filter. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| attenuation | [3-30] dB m f\_nyquist |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getAafAttenuation |
| params: [] |
| Result: [errNo, attenuation] |

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

|  |
| --- |
| int **IDS\_realtime\_getAafAttenuation**(int deviceHandle, int\* attenuation) |

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

|  |
| --- |
| attenuation = **[dev].realtime.getAafAttenuation**() |

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

|  |
| --- |
| [attenuation] = **IDS\_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 | errNo |
| enabled | 0 - the Anti-Aliasing Filter is disabled   1 - the Anti-Aliasing Filter is enabled |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getAafEnabled |
| params: [] |
| Result: [errNo, enabled] |

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

|  |
| --- |
| int **IDS\_realtime\_getAafEnabled**(int deviceHandle, int\* enabled) |

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

|  |
| --- |
| enabled = **[dev].realtime.getAafEnabled**() |

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

|  |
| --- |
| [enabled] = **IDS\_realtime\_getAafEnabled**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetAafEnabled**() |

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

|  |
| --- |
| getAafEnabled.vi |

|  |
| --- |
| **getAafWindow** Returns the current filter window of the anti-aliasing filter. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| window | 0 = Rectangular,  1 = Cosine,  2 = Cosine^2,  3 = Hamming,  4 = Raised Cosine,  5 = Automatic |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getAafWindow |
| params: [] |
| Result: [errNo, window] |

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

|  |
| --- |
| int **IDS\_realtime\_getAafWindow**(int deviceHandle, int\* window) |

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

|  |
| --- |
| window = **[dev].realtime.getAafWindow**() |

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

|  |
| --- |
| [window] = **IDS\_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 | errNo |
| value | N, Linear Analog High Pass Cut-Off freqency is 1600/2^N kHz, with N \\in [1,24] |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getHighPassCutOffFreq |
| params: [] |
| Result: [errNo, value] |

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

|  |
| --- |
| int **IDS\_realtime\_getHighPassCutOffFreq**(int deviceHandle, int\* value) |

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

|  |
| --- |
| value = **[dev].realtime.getHighPassCutOffFreq**() |

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

|  |
| --- |
| [value] = **IDS\_realtime\_getHighPassCutOffFreq**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetHighPassCutOffFreq**() |

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

|  |
| --- |
| getHighPassCutOffFreq.vi |

|  |
| --- |
| **getLinearRange** Reads out the range number of Linear/Analog output mode. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| rangenumber | N, Linear Analog Range is +-2^(N+11) pm, with N \\in [0, 34] |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getLinearRange |
| params: [] |
| Result: [errNo, rangenumber] |

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

|  |
| --- |
| int **IDS\_realtime\_getLinearRange**(int deviceHandle, int\* rangenumber) |

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

|  |
| --- |
| rangenumber = **[dev].realtime.getLinearRange**() |

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

|  |
| --- |
| [rangenumber] = **IDS\_realtime\_getLinearRange**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetLinearRange**() |

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

|  |
| --- |
| getLinearRange.vi |

|  |
| --- |
| **getPeriodHsslClk** Reads out the HSSL period clock. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| period | Period in the Range of [40ns..10200ns] |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getPeriodHsslClk |
| params: [] |
| Result: [errNo, period] |

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

|  |
| --- |
| int **IDS\_realtime\_getPeriodHsslClk**(int deviceHandle, int\* period) |

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

|  |
| --- |
| period = **[dev].realtime.getPeriodHsslClk**() |

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

|  |
| --- |
| [period] = **IDS\_realtime\_getPeriodHsslClk**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetPeriodHsslClk**() |

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

|  |
| --- |
| getPeriodHsslClk.vi |

|  |
| --- |
| **getPeriodHsslGap** Reads out the HSSL period gap. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| gap | Number of clocks |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getPeriodHsslGap |
| params: [] |
| Result: [errNo, gap] |

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

|  |
| --- |
| int **IDS\_realtime\_getPeriodHsslGap**(int deviceHandle, int\* gap) |

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

|  |
| --- |
| gap = **[dev].realtime.getPeriodHsslGap**() |

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

|  |
| --- |
| [gap] = **IDS\_realtime\_getPeriodHsslGap**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetPeriodHsslGap**() |

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

|  |
| --- |
| getPeriodHsslGap.vi |

|  |
| --- |
| **getPeriodSinCosClk** Reads out the Sine-Cosine and AquadB period clock. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| period | 40ns to 10200ns |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getPeriodSinCosClk |
| params: [] |
| Result: [errNo, period] |

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

|  |
| --- |
| int **IDS\_realtime\_getPeriodSinCosClk**(int deviceHandle, int\* period) |

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

|  |
| --- |
| period = **[dev].realtime.getPeriodSinCosClk**() |

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

|  |
| --- |
| [period] = **IDS\_realtime\_getPeriodSinCosClk**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetPeriodSinCosClk**() |

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

|  |
| --- |
| getPeriodSinCosClk.vi |

|  |
| --- |
| **getResolutionBissC** Reads out the BissC resolution. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| resolution | 1pm to 65535pm |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getResolutionBissC |
| params: [] |
| Result: [errNo, resolution] |

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

|  |
| --- |
| int **IDS\_realtime\_getResolutionBissC**(int deviceHandle, int\* resolution) |

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

|  |
| --- |
| resolution = **[dev].realtime.getResolutionBissC**() |

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

|  |
| --- |
| [resolution] = **IDS\_realtime\_getResolutionBissC**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionBissC**() |

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

|  |
| --- |
| getResolutionBissC.vi |

|  |
| --- |
| **getResolutionHsslHigh** Reads out the HSSL resolution high bit. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| resolution | Resolution in the Range of [0..46] |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getResolutionHsslHigh |
| params: [] |
| Result: [errNo, resolution] |

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

|  |
| --- |
| int **IDS\_realtime\_getResolutionHsslHigh**(int deviceHandle, int\* resolution) |

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

|  |
| --- |
| resolution = **[dev].realtime.getResolutionHsslHigh**() |

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

|  |
| --- |
| [resolution] = **IDS\_realtime\_getResolutionHsslHigh**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionHsslHigh**() |

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

|  |
| --- |
| getResolutionHsslHigh.vi |

|  |
| --- |
| **getResolutionHsslLow** Reads out the HSSL resolution low bit.# |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| resolution | Resolution in the range of [0..46] |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getResolutionHsslLow |
| params: [] |
| Result: [errNo, resolution] |

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

|  |
| --- |
| int **IDS\_realtime\_getResolutionHsslLow**(int deviceHandle, int\* resolution) |

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

|  |
| --- |
| resolution = **[dev].realtime.getResolutionHsslLow**() |

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

|  |
| --- |
| [resolution] = **IDS\_realtime\_getResolutionHsslLow**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionHsslLow**() |

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

|  |
| --- |
| getResolutionHsslLow.vi |

|  |
| --- |
| **getResolutionSinCos** Reads out the Sine-Cosine and AquadB resolution. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| resolution | 1pm to 65535pm |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getResolutionSinCos |
| params: [] |
| Result: [errNo, resolution] |

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

|  |
| --- |
| int **IDS\_realtime\_getResolutionSinCos**(int deviceHandle, int\* resolution) |

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

|  |
| --- |
| resolution = **[dev].realtime.getResolutionSinCos**() |

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

|  |
| --- |
| [resolution] = **IDS\_realtime\_getResolutionSinCos**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetResolutionSinCos**() |

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

|  |
| --- |
| getResolutionSinCos.vi |

|  |
| --- |
| **getRtDistanceMode** Reads out the distance mode. Depending on the realtime output mode, the mode can  be Displacement (returns 1), Absolute Distance (returns 2) or Vibrometry (returns 3). |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| linearmode | 1 = Displacement (Available in HSSL mode and Linear Mode)  2 = Absolute Distance (Available in HSSL mode only)  3 = Vibrometry (Available in Linear mode) |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getRtDistanceMode |
| params: [] |
| Result: [errNo, linearmode] |

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

|  |
| --- |
| int **IDS\_realtime\_getRtDistanceMode**(int deviceHandle, int\* linearmode) |

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

|  |
| --- |
| linearmode = **[dev].realtime.getRtDistanceMode**() |

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

|  |
| --- |
| [linearmode] = **IDS\_realtime\_getRtDistanceMode**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetRtDistanceMode**() |

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

|  |
| --- |
| getRtDistanceMode.vi |

|  |
| --- |
| **getRtOutMode** Reads out the current realtime output mode. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| rtOutMode | 0 = HSSL (TTL), 1 = HSSL (LVDS), 2 = AquadB (TTL),  3 = AquadB (LVDS), 4 = SinCos (TTL Error Signal),  5 = SinCos (LVDS Error Signal), 6 = Linear (TTL), 7 = Linear (LVDS),  8 = BiSS-C, 9 = Deactivated |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getRtOutMode |
| params: [] |
| Result: [errNo, rtOutMode] |

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

|  |
| --- |
| int **IDS\_realtime\_getRtOutMode**(int deviceHandle, int\* rtOutMode) |

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

|  |
| --- |
| rtOutMode = **[dev].realtime.getRtOutMode**() |

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

|  |
| --- |
| [rtOutMode] = **IDS\_realtime\_getRtOutMode**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetRtOutMode**() |

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

|  |
| --- |
| getRtOutMode.vi |

|  |
| --- |
| **getTestChannelEnabled** Checks if the test channel is enabled |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| enabled | true = enabled, false = disabled |

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

|  |
| --- |
| method: com.attocube.ids.realtime.getTestChannelEnabled |
| params: [] |
| Result: [errNo, enabled] |

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

|  |
| --- |
| int **IDS\_realtime\_getTestChannelEnabled**(int deviceHandle, int\* enabled) |

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

|  |
| --- |
| enabled = **[dev].realtime.getTestChannelEnabled**() |

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

|  |
| --- |
| [enabled] = **IDS\_realtime\_getTestChannelEnabled**() |

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

|  |
| --- |
| int value = [Device].**Realtime\_GetTestChannelEnabled**() |

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

|  |
| --- |
| getTestChannelEnabled.vi |

|  |
| --- |
| **setAaf** Sets the anti-aliasing filter with assigned filter window. |

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

|  |  |  |
| --- | --- | --- |
| In | enabled | 0 - disables the Anti-Aliasing Filter   1 - enables the Anti-Aliasing Filter |
| attenuation | [3-30] dB m f\_nyquist |
| window | 0 = Rectangular,  1 = Cosine,  2 = Cosine^2,  3 = Hamming,  4 = Raised Cosine,  5 = Automatic |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setAaf |
| params: [enabled, attenuation, window] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setAaf**(int deviceHandle, int enabled, int attenuation, int window) |

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

|  |
| --- |
| **[dev].realtime.setAaf**(enabled, attenuation, window) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setAaf**(enabled, attenuation, window) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetAaf**(int enabled, int attenuation, int window) |

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

|  |
| --- |
| setAaf.vi |

|  |
| --- |
| **setHighPassCutOffFreq** Sets the high pass filter number of Linear/Analog output mode. |

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

|  |  |  |
| --- | --- | --- |
| In | value | N, Linear Analog High Pass Cut-Off freqency is 1600/2^N kHz, with N \\in [1,24] |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setHighPassCutOffFreq |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setHighPassCutOffFreq**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setHighPassCutOffFreq**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setHighPassCutOffFreq**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetHighPassCutOffFreq**(int value) |

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

|  |
| --- |
| setHighPassCutOffFreq.vi |

|  |
| --- |
| **setLinearRange** Sets the range number of Linear/Analog output mode. |

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

|  |  |  |
| --- | --- | --- |
| In | rangenumber | N, Linear Analog Range is +-2^(N+11) pm, with N \\in [0, 34] |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setLinearRange |
| params: [rangenumber] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setLinearRange**(int deviceHandle, int rangenumber) |

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

|  |
| --- |
| **[dev].realtime.setLinearRange**(rangenumber) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setLinearRange**(rangenumber) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetLinearRange**(int rangenumber) |

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

|  |
| --- |
| setLinearRange.vi |

|  |
| --- |
| **setPeriodHsslClk** Set the HSSL period clock. The value has to be a multiple of 40ns. If not, the value automatically is rounded. |

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

|  |  |  |
| --- | --- | --- |
| In | period | Period in the Range of [40ns..10200ns] |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setPeriodHsslClk |
| params: [period] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setPeriodHsslClk**(int deviceHandle, int period) |

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

|  |
| --- |
| **[dev].realtime.setPeriodHsslClk**(period) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setPeriodHsslClk**(period) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetPeriodHsslClk**(int period) |

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

|  |
| --- |
| setPeriodHsslClk.vi |

|  |
| --- |
| **setPeriodHsslGap** Set the HSSL gap. |

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

|  |  |  |
| --- | --- | --- |
| In | value | Number of clocks |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setPeriodHsslGap |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setPeriodHsslGap**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setPeriodHsslGap**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setPeriodHsslGap**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetPeriodHsslGap**(int value) |

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

|  |
| --- |
| setPeriodHsslGap.vi |

|  |
| --- |
| **setPeriodSinCosClk** Sets the Sine-Cosine and AquadB period clock. The value has to be a multiple of 40ns. If not, the value automatically is rounded. |

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

|  |  |  |
| --- | --- | --- |
| In | value | period 40ns to 10200ns |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setPeriodSinCosClk |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setPeriodSinCosClk**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setPeriodSinCosClk**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setPeriodSinCosClk**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetPeriodSinCosClk**(int value) |

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

|  |
| --- |
| setPeriodSinCosClk.vi |

|  |
| --- |
| **setResolutionBissC** Sets the BissC resolution. |

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

|  |  |  |
| --- | --- | --- |
| In | value | resolution 1pm to 65535pm |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setResolutionBissC |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setResolutionBissC**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setResolutionBissC**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setResolutionBissC**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionBissC**(int value) |

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

|  |
| --- |
| setResolutionBissC.vi |

|  |
| --- |
| **setResolutionHsslHigh** Sets the HSSL resolution high bit. |

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

|  |  |  |
| --- | --- | --- |
| In | value | Resolution in the Range of [0..46] |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setResolutionHsslHigh |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setResolutionHsslHigh**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setResolutionHsslHigh**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setResolutionHsslHigh**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionHsslHigh**(int value) |

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

|  |
| --- |
| setResolutionHsslHigh.vi |

|  |
| --- |
| **setResolutionHsslLow** Sets the HSSL resolution low bit. |

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

|  |  |  |
| --- | --- | --- |
| In | value | Resolution in the Range of [0..46] |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setResolutionHsslLow |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setResolutionHsslLow**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setResolutionHsslLow**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setResolutionHsslLow**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionHsslLow**(int value) |

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

|  |
| --- |
| setResolutionHsslLow.vi |

|  |
| --- |
| **setResolutionSinCos** Sets the Sine-Cosine and AquadB resolution. |

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

|  |  |  |
| --- | --- | --- |
| In | value | resolution 1pm to 65535pm |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setResolutionSinCos |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setResolutionSinCos**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setResolutionSinCos**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setResolutionSinCos**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetResolutionSinCos**(int value) |

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

|  |
| --- |
| setResolutionSinCos.vi |

|  |
| --- |
| **setRtDistanceMode** Sets the distance mode. Depending on the configuration of the IDS the mode can be  Displacement (returns 1), Absolute Distance (returns 2) or Vibrometry (returns 3). |

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

|  |  |  |
| --- | --- | --- |
| In | value | 1 = Displacement (HSSL mode and Linear Mode)  2 = Absolute Distance (HSSL mode only)  3 = Vibrometry (Linear mode) |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setRtDistanceMode |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setRtDistanceMode**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setRtDistanceMode**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setRtDistanceMode**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetRtDistanceMode**(int value) |

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

|  |
| --- |
| setRtDistanceMode.vi |

|  |
| --- |
| **setRtOutMode** Sets the real time output mode. |

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

|  |  |  |
| --- | --- | --- |
| In | value | rtOutMode 0 = HSSL (TTL), 1 = HSSL (LVDS), 2 = AquadB (TTL),  3 = AquadB (LVDS), 4 = SinCos (TTL Error Signal),  5 = SinCos (LVDS Error Signal), 6 = Linear (TTL), 7 = Linear (LVDS),  8 = BiSS-C, 9 = Deactivated |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.realtime.setRtOutMode |
| params: [value] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_realtime\_setRtOutMode**(int deviceHandle, int value) |

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

|  |
| --- |
| **[dev].realtime.setRtOutMode**(value) |

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

|  |
| --- |
| [] = **IDS\_realtime\_setRtOutMode**(value) |

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

|  |
| --- |
| void value = [Device].**Realtime\_SetRtOutMode**(int value) |

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

|  |
| --- |
| setRtOutMode.vi |

## System

|  |
| --- |
| **getCurrentMode** Reads out the current IDS system state. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| mode | Values: "system idle", "measurement starting", "measurement running", "optics alignment starting", "optics alignment running", "test channels enabled" |

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

|  |
| --- |
| method: com.attocube.ids.system.getCurrentMode |
| params: [] |
| Result: [errNo, mode] |

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

|  |
| --- |
| int **IDS\_system\_getCurrentMode**(int deviceHandle, char\* mode, int size0) |

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

|  |
| --- |
| mode = **[dev].system.getCurrentMode**() |

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

|  |
| --- |
| [mode] = **IDS\_system\_getCurrentMode**() |

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

|  |
| --- |
| string value = [Device].**System\_GetCurrentMode**() |

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

|  |
| --- |
| system\_getCurrentMode.vi |

|  |
| --- |
| **getDeviceType** Reads out the IDS device type. For differences between the device types please refer to the IDS User Manual. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| type | Type of IDS (e.g. "IDS3010") |

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

|  |
| --- |
| method: com.attocube.ids.system.getDeviceType |
| params: [] |
| Result: [errNo, type] |

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

|  |
| --- |
| int **IDS\_system\_getDeviceType**(int deviceHandle, char\* type, int size0) |

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

|  |
| --- |
| type = **[dev].system.getDeviceType**() |

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

|  |
| --- |
| [type] = **IDS\_system\_getDeviceType**() |

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

|  |
| --- |
| string value = [Device].**System\_GetDeviceType**() |

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

|  |
| --- |
| system\_getDeviceType.vi |

|  |
| --- |
| **getFeaturesName** Converts the IDS feature number to its corresponding name. |

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

|  |  |  |
| --- | --- | --- |
| In | featurenumber | Number of feature |
| Out | errNo | errNo |
| names | The name of the corresponding feature |

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

|  |
| --- |
| method: com.attocube.ids.system.getFeaturesName |
| params: [featurenumber] |
| Result: [errNo, names] |

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

|  |
| --- |
| int **IDS\_system\_getFeaturesName**(int deviceHandle, int featurenumber, char\* names, int size0) |

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

|  |
| --- |
| names = **[dev].system.getFeaturesName**(featurenumber) |

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

|  |
| --- |
| [names] = **IDS\_system\_getFeaturesName**(featurenumber) |

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

|  |
| --- |
| string value = [Device].**System\_GetFeaturesName**(int featurenumber) |

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

|  |
| --- |
| system\_getFeaturesName.vi |

|  |
| --- |
| **getFpgaVersion** Reads out the IDS FPGA version. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| version | Version in the form X.Y.Z |

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

|  |
| --- |
| method: com.attocube.ids.system.getFpgaVersion |
| params: [] |
| Result: [errNo, version] |

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

|  |
| --- |
| int **IDS\_system\_getFpgaVersion**(int deviceHandle, char\* version, int size0) |

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

|  |
| --- |
| version = **[dev].system.getFpgaVersion**() |

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

|  |
| --- |
| [version] = **IDS\_system\_getFpgaVersion**() |

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

|  |
| --- |
| string value = [Device].**System\_GetFpgaVersion**() |

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

|  |
| --- |
| system\_getFpgaVersion.vi |

|  |
| --- |
| **getInitMode** Returns the Initialization mode. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| mode | 0 = High Accuracy Initialization; 1 = Quick Initialization |

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

|  |
| --- |
| method: com.attocube.ids.system.getInitMode |
| params: [] |
| Result: [errNo, mode] |

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

|  |
| --- |
| int **IDS\_system\_getInitMode**(int deviceHandle, int\* mode) |

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

|  |
| --- |
| mode = **[dev].system.getInitMode**() |

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

|  |
| --- |
| [mode] = **IDS\_system\_getInitMode**() |

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

|  |
| --- |
| int value = [Device].**System\_GetInitMode**() |

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

|  |
| --- |
| system\_getInitMode.vi |

|  |
| --- |
| **getNbrFeaturesActivated** Reads out the amount of activated features activated on the IDS. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |
| nbr | Gives the number of activated features. |

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

|  |
| --- |
| method: com.attocube.ids.system.getNbrFeaturesActivated |
| params: [] |
| Result: [errNo, nbr] |

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

|  |
| --- |
| int **IDS\_system\_getNbrFeaturesActivated**(int deviceHandle, int\* nbr) |

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

|  |
| --- |
| nbr = **[dev].system.getNbrFeaturesActivated**() |

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

|  |
| --- |
| [nbr] = **IDS\_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 occured while starting and running measurements and alignments. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.system.getSystemError |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_getSystemError**(int deviceHandle) |

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

|  |
| --- |
| **[dev].system.getSystemError**() |

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

|  |
| --- |
| [] = **IDS\_system\_getSystemError**() |

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

|  |
| --- |
| void value = [Device].**System\_GetSystemError**() |

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

|  |
| --- |
| system\_getSystemError.vi |

|  |
| --- |
| **resetAxes** Resets the position value of all measurement axes to zero. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.system.resetAxes |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_resetAxes**(int deviceHandle) |

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

|  |
| --- |
| **[dev].system.resetAxes**() |

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

|  |
| --- |
| [] = **IDS\_system\_resetAxes**() |

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

|  |
| --- |
| void value = [Device].**System\_ResetAxes**() |

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

|  |
| --- |
| system\_resetAxes.vi |

|  |
| --- |
| **resetAxis** Resets the position value of a specific measurement axis to zero. |

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

|  |  |  |
| --- | --- | --- |
| In | axis | [0|1|2] |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.system.resetAxis |
| params: [axis] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_resetAxis**(int deviceHandle, int axis) |

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

|  |
| --- |
| **[dev].system.resetAxis**(axis) |

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

|  |
| --- |
| [] = **IDS\_system\_resetAxis**(axis) |

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

|  |
| --- |
| void value = [Device].**System\_ResetAxis**(int axis) |

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

|  |
| --- |
| system\_resetAxis.vi |

|  |
| --- |
| **resetError** Resets a measurement error that can have occurred with the aim to continue the interrupted measurement. It is configurable if an additional renormalization process (please refer to the IDS User Manual) should be performed or not.  This function can be used for two concerns: 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 utilized to normalize the Lissajous-Figure of all three optical 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 high-resolution measurements are guaranteed. B. After changing the optical components as, for example, the retro reflector 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 performRenormalisation, it can be decided, if the renormalization process should be executed or skipped. If it is executed, the IDS system needs around 14-20 seconds to get back to the measurement mode. It is also important to comment that the recommended action after an error is to stop and to restart the displacement 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 long-term displacement measurements. |

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

|  |  |  |
| --- | --- | --- |
| In | perform | renormalization |
| Out | errNo | errNo |

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

|  |
| --- |
| method: com.attocube.ids.system.resetError |
| params: [perform] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_resetError**(int deviceHandle, bool perform) |

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

|  |
| --- |
| **[dev].system.resetError**(perform) |

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

|  |
| --- |
| [] = **IDS\_system\_resetError**(perform) |

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

|  |
| --- |
| void value = [Device].**System\_ResetError**(bool perform) |

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

|  |
| --- |
| system\_resetError.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 | errNo |

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

|  |
| --- |
| method: com.attocube.ids.system.setInitMode |
| params: [mode] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_setInitMode**(int deviceHandle, int mode) |

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

|  |
| --- |
| **[dev].system.setInitMode**(mode) |

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

|  |
| --- |
| [] = **IDS\_system\_setInitMode**(mode) |

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

|  |
| --- |
| void value = [Device].**System\_SetInitMode**(int mode) |

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

|  |
| --- |
| system\_setInitMode.vi |

|  |
| --- |
| **startMeasurement** Starts the displacement measurement system state.  Please query errors during starting and running measurements by system.getSystemError |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | Error on initiating the system mode change |

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

|  |
| --- |
| method: com.attocube.ids.system.startMeasurement |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_startMeasurement**(int deviceHandle) |

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

|  |
| --- |
| **[dev].system.startMeasurement**() |

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

|  |
| --- |
| [] = **IDS\_system\_startMeasurement**() |

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

|  |
| --- |
| void value = [Device].**System\_StartMeasurement**() |

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

|  |
| --- |
| system\_startMeasurement.vi |

|  |
| --- |
| **startOpticsAlignment** Starts the optical alignment system state.  Please query errors during starting and running alignments by system.getSystemError |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | Error on initiating the system mode change |

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

|  |
| --- |
| method: com.attocube.ids.system.startOpticsAlignment |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_startOpticsAlignment**(int deviceHandle) |

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

|  |
| --- |
| **[dev].system.startOpticsAlignment**() |

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

|  |
| --- |
| [] = **IDS\_system\_startOpticsAlignment**() |

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

|  |
| --- |
| void value = [Device].**System\_StartOpticsAlignment**() |

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

|  |
| --- |
| system\_startOpticsAlignment.vi |

|  |
| --- |
| **stopMeasurement** Stops the displacement measurement system state. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | Error on initiating the system mode change |

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

|  |
| --- |
| method: com.attocube.ids.system.stopMeasurement |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_stopMeasurement**(int deviceHandle) |

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

|  |
| --- |
| **[dev].system.stopMeasurement**() |

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

|  |
| --- |
| [] = **IDS\_system\_stopMeasurement**() |

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

|  |
| --- |
| void value = [Device].**System\_StopMeasurement**() |

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

|  |
| --- |
| system\_stopMeasurement.vi |

|  |
| --- |
| **stopOpticsAlignment** Stops the optical alignment system state. |

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

|  |  |  |
| --- | --- | --- |
| Out | errNo | Error on initiating the system mode change |

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

|  |
| --- |
| method: com.attocube.ids.system.stopOpticsAlignment |
| params: [] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_system\_stopOpticsAlignment**(int deviceHandle) |

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

|  |
| --- |
| **[dev].system.stopOpticsAlignment**() |

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

|  |
| --- |
| [] = **IDS\_system\_stopOpticsAlignment**() |

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

|  |
| --- |
| void value = [Device].**System\_StopOpticsAlignment**() |

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

|  |
| --- |
| system\_stopOpticsAlignment.vi |

## Access

|  |
| --- |
| **unlock** This function unlocks the device, so it will not be necessary to execute the grantAccess function to run any function |

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

|  |  |  |
| --- | --- | --- |
| In | password | string the current password |
| Out | errNo | errorCode |

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

|  |
| --- |
| method: unlock |
| params: [password] |
| Result: [errNo] |

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

|  |
| --- |
| int **IDS\_unlock**(int deviceHandle, const char\* password) |

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

|  |
| --- |
| **[dev].access.unlock**(password) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **IDS\_unlock**(password) |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Unlock**(string password) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| unlock.vi |

## About

|  |
| --- |
| **getInstalledPackages** Get list of packages installed on the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| 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 | errorCode |
| 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 |

## System\_service

|  |
| --- |
| **apply** Apply temporary system configuration |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.apply |
| params: [] |
| Result: [errNo] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_apply**(int deviceHandle) |

|  |
| --- |
| **Python** |

|  |
| --- |
| **[dev].system\_service.apply**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [] = **system\_apply**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| void value = [Device].**Apply**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| apply.vi |

|  |
| --- |
| **errorNumberToRecommendation** Get a recommendation for the error code |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | language | integer: Language code |
| errNbr | interger: Error code to translate |
| Out | errNo | errorCode |
| 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** Get 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 | interger: Error code to translate |
| Out | errNo | errorCode |
| value\_string1 | string: Error description |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.errorNumberToString |
| params: [language, errNbr] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_errorNumberToString**(int deviceHandle, int language, int errNbr, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.errorNumberToString**(language, errNbr) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **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 IDS will be configured as DHCP server. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Get the actual device name |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_string1 | string: actual device name |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getDeviceName |
| params: [] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getDeviceName**(int deviceHandle, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.getDeviceName**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_getDeviceName**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetDeviceName**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getDeviceName.vi |

|  |
| --- |
| **getFirmwareVersion** Get the firmware version of the system |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_string1 | string: The firmware version |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getFirmwareVersion |
| params: [] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getFirmwareVersion**(int deviceHandle, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.getFirmwareVersion**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_getFirmwareVersion**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetFirmwareVersion**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getFirmwareVersion.vi |

|  |
| --- |
| **getFluxCode** Get the flux code of the system |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_string1 | string: flux code |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getFluxCode |
| params: [] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getFluxCode**(int deviceHandle, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.getFluxCode**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_getFluxCode**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetFluxCode**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getFluxCode.vi |

|  |
| --- |
| **getHostname** Return device hostname |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| available | available |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getHostname |
| params: [] |
| Result: [errNo, available] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getHostname**(int deviceHandle, char\* available, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| available = **[dev].system\_service.getHostname**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [available] = **system\_getHostname**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetHostname**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getHostname.vi |

|  |
| --- |
| **getMacAddress** Get the mac address of the system |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_string1 | string: Mac address of the system |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getMacAddress |
| params: [] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getMacAddress**(int deviceHandle, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.getMacAddress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_getMacAddress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetMacAddress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getMacAddress.vi |

|  |
| --- |
| **getSerialNumber** Get the serial number of the system |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_string1 | string: Serial number |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.getSerialNumber |
| params: [] |
| Result: [errNo, value\_string1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_getSerialNumber**(int deviceHandle, char\* value\_string1, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_string1 = **[dev].system\_service.getSerialNumber**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_string1] = **system\_getSerialNumber**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**GetSerialNumber**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getSerialNumber.vi |

## Network

|  |
| --- |
| **apply** Apply temporary IP configuration and load it |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Change the wifi configuration and applies it |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 0: Access point, 1: Wifi client |
| ssid |  |
| psk | Pre-shared key |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Discard temporary IP configuration |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Get the default gateway of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| 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** Get the DNS resolver |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | priority | of DNS resolver (Usually: 0 = Default, 1 = Backup) |
| Out | errNo | errorCode |
| IP | address of DNS resolver |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getDnsResolver |
| params: [priority] |
| Result: [errNo, IP] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getDnsResolver**(int deviceHandle, int priority, char\* IP, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| IP = **[dev].network.getDnsResolver**(priority) |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [IP] = **system\_network\_getDnsResolver**(priority) |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetDnsResolver**(int priority) |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getDnsResolver.vi |

|  |
| --- |
| **getEnableDhcpClient** Get the state of DHCP client |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_boolean1 | boolean: true = DHCP client enable, false = DHCP client disable |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getEnableDhcpClient |
| params: [] |
| Result: [errNo, value\_boolean1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getEnableDhcpClient**(int deviceHandle, bool\* value\_boolean1) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_boolean1 = **[dev].network.getEnableDhcpClient**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_boolean1] = **system\_network\_getEnableDhcpClient**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Network\_GetEnableDhcpClient**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getEnableDhcpClient.vi |

|  |
| --- |
| **getEnableDhcpServer** Get the state of DHCP server |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_boolean1 | boolean: true = DHCP server enable, false = DHCP server disable |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getEnableDhcpServer |
| params: [] |
| Result: [errNo, value\_boolean1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getEnableDhcpServer**(int deviceHandle, bool\* value\_boolean1) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_boolean1 = **[dev].network.getEnableDhcpServer**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_boolean1] = **system\_network\_getEnableDhcpServer**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Network\_GetEnableDhcpServer**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getEnableDhcpServer.vi |

|  |
| --- |
| **getIpAddress** Get the IP address of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| 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** Get the proxy settings of the devide |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| Proxy | Server String, empty for no proxy |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getProxyServer |
| params: [] |
| Result: [errNo, Proxy] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getProxyServer**(int deviceHandle, char\* Proxy, int size0) |

|  |
| --- |
| **Python** |

|  |
| --- |
| Proxy = **[dev].network.getProxyServer**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [Proxy] = **system\_network\_getProxyServer**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| string value = [Device].**Network\_GetProxyServer**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getProxyServer.vi |

|  |
| --- |
| **getRealIpAddress** Get the real IP address of the device set to the network interface (br0, eth1 or eth0) |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| IP | address as string |

|  |
| --- |
| **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** Get the subnet mask of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| 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** Get the operation mode of the wifi adapter |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| 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** Get the the passphrase of the network hosted (mode: Access point) or connected to (mode: client) |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| 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** Returns is a Wifi interface is present |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| True | True, if interface is present |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.network.getWifiPresent |
| params: [] |
| Result: [errNo, True] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_network\_getWifiPresent**(int deviceHandle, bool\* True) |

|  |
| --- |
| **Python** |

|  |
| --- |
| True = **[dev].network.getWifiPresent**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [True] = **system\_network\_getWifiPresent**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| bool value = [Device].**Network\_GetWifiPresent**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getWifiPresent.vi |

|  |
| --- |
| **getWifiSSID** Get the the SSID of the network hosted (mode: Access point) or connected to (mode: client) |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| SSID | SSID |

|  |
| --- |
| **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** Set the default gateway of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | gateway | Default gateway as string |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Set the DNS resolver |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | priority | of DNS resolver (Usually: 0 = Default, 1 = Backup) |
| resolver | The resolver's IP address as string |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Enable or disable DHCP client |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | enable | boolean: true = enable DHCP client, false = disable DHCP client |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Enable or disable DHCP server |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | enable | boolean: true = enable DHCP server, false = disable DHCP server |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Set the IP address of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | address | IP address as string |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Set the proxy server of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | proxyServer | Proxy Server Setting as string |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Set the subnet mask of the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | netmask | Subnet mask as string |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Change the operation mode of the wifi adapter |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | mode | 0: Access point, 1: Wifi client |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Change the passphrase of the network hosted (mode: Access point) or connected to (mode: client) |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | psk | Pre-shared key |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Change the SSID of the network hosted (mode: Access point) or connected to (mode: client) |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | ssid |  |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Verify that temporary IP configuration is correct |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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 |

## System\_service

|  |
| --- |
| **rebootSystem** Reboot the system |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Set custom name for the device |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | name | string: device name |
| Out | errNo | errorCode |

|  |
| --- |
| **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 |

|  |
| --- |
| **updateTimeFromInternet** Update system time by querying attocube.com |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Get the progress of running license update |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_int1 | int: progress in percent |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.getLicenseUpdateProgress |
| params: [] |
| Result: [errNo, value\_int1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_getLicenseUpdateProgress**(int deviceHandle, int\* value\_int1) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_int1 = **[dev].update.getLicenseUpdateProgress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_int1] = **system\_update\_getLicenseUpdateProgress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Update\_GetLicenseUpdateProgress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getLicenseUpdateProgress.vi |

|  |
| --- |
| **getSwUpdateProgress** Get the progress of running update |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |
| value\_int1 | int: progress in percent |

|  |
| --- |
| **JSON Method** |

|  |
| --- |
| method: com.attocube.system.update.getSwUpdateProgress |
| params: [] |
| Result: [errNo, value\_int1] |

|  |
| --- |
| **C-DLL call** |

|  |
| --- |
| int **system\_update\_getSwUpdateProgress**(int deviceHandle, int\* value\_int1) |

|  |
| --- |
| **Python** |

|  |
| --- |
| value\_int1 = **[dev].update.getSwUpdateProgress**() |

|  |
| --- |
| **Matlab** |

|  |
| --- |
| [value\_int1] = **system\_update\_getSwUpdateProgress**() |

|  |
| --- |
| **C#** |

|  |
| --- |
| int value = [Device].**Update\_GetSwUpdateProgress**() |

|  |
| --- |
| **LabVIEW** |

|  |
| --- |
| getSwUpdateProgress.vi |

|  |
| --- |
| **licenseUpdateBase64** Execute the license update with base64 file uploaded. After execution, a manual reboot is nevessary. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Execute the update with base64 file uploaded. After completion, a manual reboot is necessary. |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Upload new license file in format base 64 |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | offset | int: offset of the data |
| b64Data | string: base64 data |
| Out | errNo | errorCode |

|  |
| --- |
| **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** Upload new firmware image in format base 64 |

|  |
| --- |
| **Function specific parameters** |

|  |  |  |
| --- | --- | --- |
| In | offset | int: offset of the data |
| b64Data | string: base64 data |
| Out | errNo | errorCode |

|  |
| --- |
| **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 |

|  |
| --- |
| attocube systems AG Eglfinger Weg 2 D - 85540 Haar, Germany  Phone: +49 89 - 4207 97 0  Fax: +49 89 - 4207 97 20 190  E-Mail: info@attocube.com  www.attocube.com  **For technical queries, contact:**  support@attocube.com  **North America Support Hotlines:**  +1 212 962 6930 (East Coast Office)  +1 510 649 9245 (West Coast Office)  **South America Support Hotline:**  +1 510 649 9245 |