Interface Manual

AMC100 & AMC300

Modified: 06/2024
Products: AMC100 | AMC300 | attoDISCOVERY Software | AMC Webserver Application

Table of Contents

1 System Integration

The AMC100 and the AMC300 can be integrated with external systems or devices by

- combining it with third party hardware
- establishing incoming and outgoing trigger connections (/IO upgrade required)
- controlling it with individual software interfaces

1.1 Connecting to Third Party Hardware

CAUTION

General hazard!

Inadequate hardware connections may cause injury and are likely to damage the device or interfere with an appropriate functioning.

- Always contact attocube for technical support, before combining the device with third party hardware.
- Do carefully observe the information in this section when combining the device with third party hardware.

NOTE

Attocube is not liable for any damages resulting from an unauthorized combination of the device with third party hardware. Unauthorized combination with third party hardware is not covered by attocube's warranty.

1.1.1 Cabling Restrictions

For optimal performance, obey the following combination restrictions:

- Do not to connect cabling with a wire resistance > 5 Ω.
- Use EMV housings as enclosure for the D-sub connectors.
- Use extra shielded twisted pair wires for the piezo voltage supply.
- Do not connect any cable > 5 m.

1.1.2 Pin Assignments

The pins of the device’s positioner control cables can be found in the AMC100 or AMC300 User Manual.

2 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.

2.1 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:

NOTE

Part of the conventions mentioned below are specific for the handling of attocube devices and are not necessarily applicable in other contexts.

2.1.1 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.

2.1.2 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.

2.1.3 C/C# DLLs

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

2.1.4 LabVIEW VIs

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

2.1.5 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.

2.2 Overview and implementation of the APIs

2.2.1 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:

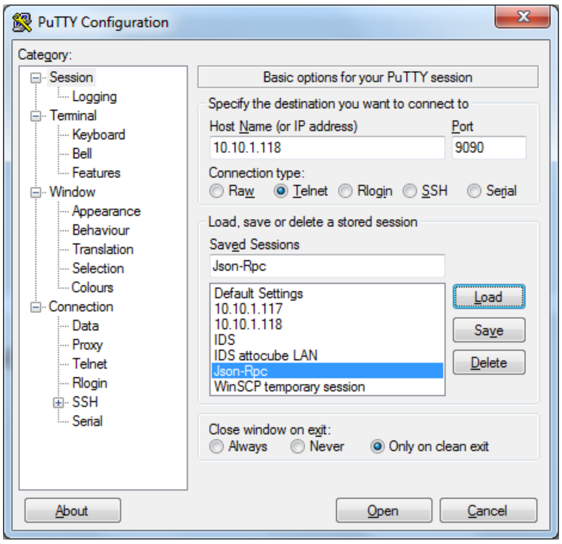
{ "jsonrpc": "2.0", "results": [0, 4], "id": 1}

Example

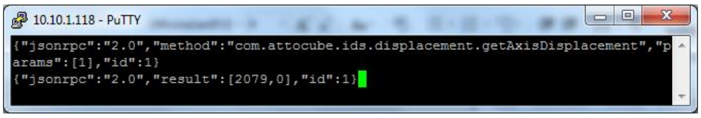
Example:

Communication via PuTTY

Open a Telnet connection with PuTTY.



Sending Json-Rpc commands in the command line interface.



2.2.2 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;
```

2.2.3 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

2.2.4 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.

2.2.5 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

2.2.6 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.

2.3 Error handling

2.3.1 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)

}
```

2.3.2 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
}
```

2.3.3 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)
```

2.3.4 LabVIEW error handling

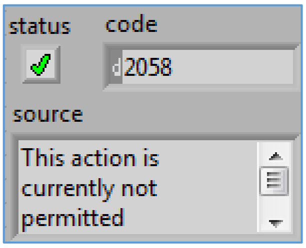
Introduction

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

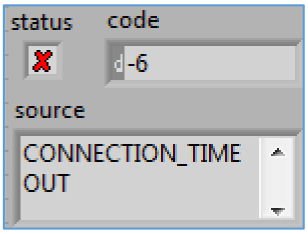
Example for an error message

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.



3 Functions

3.1 Access

getLockStatus

This function returns if the device is locked and if the current client is authorized to use the device.

<table>
<tr><th colspan="3">Function specific parameters</th></tr>
<tr><td rowspan="3">Out</td><td>errNo</td><td>errorCode</td></tr>
<tr><td>locked</td><td>Is the device locked?</td></tr>
<tr><td>authorized</td><td>Is the client authorized?</td></tr>
<tr><th colspan="3">JSON Method</th></tr>
<tr><td colspan="3">method: getLockStatus</td></tr>
<tr><td colspan="3">params: []</td></tr>
<tr><td colspan="3">Result: [errNo, locked, authorized]</td></tr>
<tr><th colspan="3">C-DLL call</th></tr>
<tr><td colspan="3">int AMC_getLockStatus(int deviceHandle, bool* locked, bool* authorized)</td></tr>
<tr><th colspan="3">Python</th></tr>
<tr><td colspan="3">locked, authorized = [dev].access.getLockStatus()</td></tr>
<tr><th colspan="3">Matlab</th></tr>
<tr><td colspan="3">[locked, authorized] = AMC_getLockStatus()</td></tr>
<tr><th colspan="3">C#</th></tr>
<tr><td colspan="3">Tuple<bool,bool> value = [Device].GetLockStatus()</td></tr>
<tr><th colspan="3">LabVIEW</th></tr>
<tr><td colspan="3">getLockStatus.vi</td></tr>
</table>

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 **AMC_grantAccess**(int deviceHandle, const char* password)		
Python		
[dev].access.grantAccess(password)		
Matlab		
[] = **AMC_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 **AMC_lock**(int deviceHandle, const char* password)		
Python		
[dev].access.lock(password)		
Matlab		
[] = **AMC_lock**(password)		
C#		
void value = [Device].**Lock**(string password)		
LabVIEW		
lock.vi		

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 **AMC_unlock**(int deviceHandle, const char* password)		
Python		
[dev].access.unlock(password)		
Matlab		
[] = **AMC_unlock**(password)		
C#		
void value = [Device].**Unlock**(string password)		
LabVIEW		
unlock.vi		

3.2 Amcids

getLowerSoftLimit

Gets the lower boundary of the soft limit protection.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC to get the soft limit status from
Out	errNo	int32 Error number if one occured, 0 in case of no error
	limit	double Lower boundary in pm
JSON Method		
method: com.attocube.amc.amcids.getLowerSoftLimit		
params: [axis]		
Result: [errNo, limit]		
C-DLL call		
int **AMC_amcids_getLowerSoftLimit**(int deviceHandle, int axis, double* limit)		
Python		
limit = **[dev].amcids.getLowerSoftLimit**(axis)		
Matlab		
[limit] = **amcids_getLowerSoftLimit**(axis)		
C#		
double value = [Device].**Amcids_GetLowerSoftLimit**(int axis)		
LabVIEW		
getLowerSoftLimit.vi		

getSoftLimitEnabled

Gets whether the soft limit protection is enabled.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC to get the soft limit status from
Out	errNo	int32 Error number if one occured, 0 in case of no error
	enabled	boolean True, if the soft limit should be enabled on this axis

JSON Method
method: com.attocube.amc.amcids.getSoftLimitEnabled
params: [axis]
Result: [errNo, enabled]
C-DLL call
int **AMC_amcids_getSoftLimitEnabled**(int deviceHandle, int axis, bool* enabled)
Python
enabled = **[dev].amcids.getSoftLimitEnabled**(axis)
Matlab
[enabled] = **amcids_getSoftLimitEnabled**(axis)
C#
bool value = [Device].**Amcids_GetSoftLimitEnabled**(int axis)
LabVIEW
getSoftLimitEnabled.vi

getSoftLimitReached

Gets whether the current position is out of the soft limit boundaries.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC to get the soft limit status from
Out	errNo	int32 Error number if one occured, 0 in case of no error
	enabled	boolean True, if the position is not within the boundaries
JSON Method		
method: com.attocube.amc.amcids.getSoftLimitReached		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_amcids_getSoftLimitReached**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].amcids.getSoftLimitReached**(axis)		
Matlab		
[enabled] = **amcids_getSoftLimitReached**(axis)		
C#		
bool value = [Device].**Amcids_GetSoftLimitReached**(int axis)		
LabVIEW		
getSoftLimitReached.vi		

getUpperSoftLimit

Gets the upper lower boundary of the soft limit protection.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC to get the soft limit status from
Out	errNo	int32 Error number if one occured, 0 in case of no error
	limit	double Upper boundary in pm
JSON Method		
method: com.attocube.amc.amcids.getUpperSoftLimit		
params: [axis]		
Result: [errNo, limit]		
C-DLL call		
int **AMC_amcids_getUpperSoftLimit**(int deviceHandle, int axis, double* limit)		
Python		
limit = **[dev].amcids.getUpperSoftLimit**(axis)		
Matlab		
[limit] = **amcids_getUpperSoftLimit**(axis)		
C#		
double value = [Device].**Amcids_GetUpperSoftLimit**(int axis)		
LabVIEW		
getUpperSoftLimit.vi		

resetIdsAxis

Resets the position value to zero of a specific measurement axis.
Use this for positioners with an IDS as sensor.
This method does not work for NUM and RES sensors. Use com.attocube.amc.control.resetAxis instead.

Function specific parameters		
In	axis	Axis of the IDS to reset the position
Out	errNo	int32 Error number if one occured, 0 in case of no error
JSON Method		
method: com.attocube.amc.amcids.resetIdsAxis		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_amcids_resetIdsAxis**(int deviceHandle, int axis)		
Python		
[dev].amcids.resetIdsAxis(axis)		
Matlab		
[] = **amcids_resetIdsAxis**(axis)		
C#		
void value = [Device].**Amcids_ResetIdsAxis**(int axis)		
LabVIEW		
resetIdsAxis.vi		

setLowerSoftLimit

Sets the lower boundary of the soft limit protection in pm.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC where the soft limit should be changed
	limit	Lower boundary in pm
Out	errNo	int32 Error number if one occured, 0 in case of no error
JSON Method		
method: com.attocube.amc.amcids.setLowerSoftLimit		
params: [axis, limit]		
Result: [errNo]		
C-DLL call		
int **AMC_amcids_setLowerSoftLimit**(int deviceHandle, int axis, double limit)		
Python		
[dev].amcids.setLowerSoftLimit(axis, limit)		
Matlab		
[] = **amcids_setLowerSoftLimit**(axis, limit)		
C#		
void value = [Device].**Amcids_SetLowerSoftLimit**(int axis, double limit)		
LabVIEW		
setLowerSoftLimit.vi		

setSoftLimitEnabled

Enables/disables the soft limit protection.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC where the soft limit should be changed
	enabled	True, if the soft limit should be enabled on this axis
Out	errNo	int32 Error number if one occured, 0 in case of no error
JSON Method		
method: com.attocube.amc.amcids.setSoftLimitEnabled		
params: [axis, enabled]		
Result: [errNo]		
C-DLL call		
int **AMC_amcids_setSoftLimitEnabled**(int deviceHandle, int axis, bool enabled)		
Python		
[dev].amcids.setSoftLimitEnabled(axis, enabled)		
Matlab		
[] = **amcids_setSoftLimitEnabled**(axis, enabled)		
C#		
void value = [Device].**Amcids_SetSoftLimitEnabled**(int axis, bool enabled)		
LabVIEW		
setSoftLimitEnabled.vi		

setUpperSoftLimit

Sets the upper boundary of the soft limit protection in pm.

This protection is needed if the IDS working range is smaller than the positioners travel range.

It is no hard limit, so, it is possible to overshoot it!

Function specific parameters		
In	axis	Axis of the AMC where the soft limit should be changed
	limit	Upper boundary in pm
Out	errNo	int32 Error number if one occured, 0 in case of no error
JSON Method		
method: com.attocube.amc.amcids.setUpperSoftLimit		
params: [axis, limit]		
Result: [errNo]		
C-DLL call		
int **AMC_amcids_setUpperSoftLimit**(int deviceHandle, int axis, double limit)		
Python		
[dev].amcids.setUpperSoftLimit(axis, limit)		
Matlab		
[] = **amcids_setUpperSoftLimit**(axis, limit)		
C#		
void value = [Device].**Amcids_SetUpperSoftLimit**(int axis, double limit)		
LabVIEW		
setUpperSoftLimit.vi		

3.3 Control

MultiAxisPositioning

By means of this function you can set target positions for all axes simultaneously (depending on the boolean parameters set1, set2, set3)

Additionally the current position, the status of the reference and the reference position of all three axes are returned.

This function can only be used in conjunction with NUM.

Function specific parameters		
In	set1	set target position on axis1, if "false" target1-parameter is ignored
	set2	set target position on axis2, if "false" target2-parameter is ignored
	set3	set target position on axis3, if "false" target3-parameter is ignored
	target1	target position of axis 1
	target2	target position of axis 2
	target3	target position of axis 3
Out	errNo	errNo
	ref1	Status of axis 1
	ref2	Status of axis 2
	ref3	Status of axis 3
	refpos1	reference Position of axis 1
	refpos2	reference Position of axis 2
	refpos3	reference Position of axis 3
	pos1	position of axis 1
	pos2	position of axis 2

<table>
<tr><td></td><td>pos3</td><td>position of axis 3</td></tr>
<tr><td colspan="3">JSON Method</td></tr>
<tr><td colspan="3">method: com.attocube.amc.control.MultiAxisPositioning</td></tr>
<tr><td colspan="3">params: [set1, set2, set3, target1, target2, target3]</td></tr>
<tr><td colspan="3">Result: [errNo, ref1, ref2, ref3, refpos1, refpos2, refpos3, pos1, pos2, pos3]</td></tr>
<tr><td colspan="3">C-DLL call</td></tr>
<tr><td colspan="3">int AMC_control_MultiAxisPositioning(int deviceHandle, bool set1, bool set2, bool set3, int target1, int target2, int target3, bool* ref1, bool* ref2, bool* ref3, int* refpos1, int* refpos2, int* refpos3, int* pos1, int* pos2, int* pos3)</td></tr>
<tr><td colspan="3">Python</td></tr>
<tr><td colspan="3">ref1, ref2, ref3, refpos1, refpos2, refpos3, pos1, pos2, pos3 =
[dev].control.MultiAxisPositioning(set1, set2, set3, target1, target2, target3)</td></tr>
<tr><td colspan="3">Matlab</td></tr>
<tr><td colspan="3">[ref1, ref2, ref3, refpos1, refpos2, refpos3, pos1, pos2, pos3] =
control_MultiAxisPositioning(set1, set2, set3, target1, target2, target3)</td></tr>
<tr><td colspan="3">C#</td></tr>
<tr><td colspan="3">Tuple<bool,bool,bool,int,int,int, Tuple<int,int,int>> value =
[Device].Control_MultiAxisPositioning(bool set1, bool set2, bool set3, int target1, int target2, int target3)</td></tr>
<tr><td colspan="3">LabVIEW</td></tr>
<tr><td colspan="3">MultiAxisPositioning.vi</td></tr>
</table>

MultiAxisPositioningWithTime

In addition to "MultiAxisPositioning", this function returns a timestamp to the current position of each axis.

Please refer to the description of "getPositionWithTime" for more information on the timestamp.

This function can only be used in conjunction with NUM.

Function specific parameters		
In	set1	set target position on axis1, if "false" target1-parameter is ignored
	set2	set target position on axis2, if "false" target2-parameter is ignored
	set3	set target position on axis3, if "false" target3-parameter is ignored
	target1	target position of axis 1
	target2	target position of axis 2
	target3	target position of axis 3
Out	errNo	errNo
	ref1	Status of axis 1
	ref2	Status of axis 2
	ref3	Status of axis 3
	refpos1	reference Position of axis 1
	refpos2	reference Position of axis 2
	refpos3	reference Position of axis 3
	pos1	position of axis 1
	pos2	position of axis 2
	pos3	position of axis 3

	time1	timestamp of axis 1
	time2	timestamp of axis 2
	time3	timestamp of axis 3

JSON Method
method: com.attocube.amc.control.MultiAxisPositioningWithTime
params: [set1, set2, set3, target1, target2, target3]
Result: [errNo, ref1, ref2, ref3, refpos1, refpos2, refpos3, pos1, pos2, pos3, time1, time2, time3]
C-DLL call
int **AMC_control_MultiAxisPositioningWithTime**(int deviceHandle, bool set1, bool set2, bool set3, int target1, int target2, int target3, bool* ref1, bool* ref2, bool* ref3, int* refpos1, int* refpos2, int* refpos3, int* pos1, int* pos2, int* pos3, double* time1, double* time2, double* time3)
Python
ref1, ref2, ref3, refpos1, refpos2, refpos3, pos1, pos2, pos3, time1, time2, time3 = **[dev].control.MultiAxisPositioningWithTime**(set1, set2, set3, target1, target2, target3)
Matlab
[ref1, ref2, ref3, refpos1, refpos2, refpos3, pos1, pos2, pos3, time1, time2, time3] = **control_MultiAxisPositioningWithTime**(set1, set2, set3, target1, target2, target3)
C#
Tuple<bool,bool,bool,int,int,int, Tuple<int,int,int,double,double,double>> value = [Device].**Control_MultiAxisPositioningWithTime**(bool set1, bool set2, bool set3, int target1, int target2, int target3)
LabVIEW
MultiAxisPositioningWithTime.vi

getActorName

This function gets the name of the positioner of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	actor_name	actor_name

JSON Method
method: com.attocube.amc.control.getActorName
params: [axis]
Result: [errNo, actor_name]
C-DLL call
int **AMC_control_getActorName**(int deviceHandle, int axis, char* actor_name, int size0)
Python
actor_name = **[dev].control.getActorName**(axis)
Matlab
[actor_name] = **control_getActorName**(axis)
C#
string value = [Device].**Control_GetActorName**(int axis)
LabVIEW
getActorName.vi

getActorParametersActorName

Control the actors parameter: actor name

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	actorname	actorname
JSON Method		
method: com.attocube.amc.control.getActorParametersActorName		
params: [axis]		
Result: [errNo, actorname]		
C-DLL call		
int **AMC_control_getActorParametersActorName**(int deviceHandle, int axis, char* actorname, int size0)		
Python		
actorname = **[dev].control.getActorParametersActorName**(axis)		
Matlab		
[actorname] = **control_getActorParametersActorName**(axis)		
C#		
string value = [Device].**Control_GetActorParametersActorName**(int axis)		
LabVIEW		
getActorParametersActorName.vi		

getActor

Get the setting for the actor parameter sensitivity

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	sensitivity	[1 .. 10] → 1: fast approach … 10: slow approach The behavior changes exponentially.
JSON Method		
method: com.attocube.amc.control.getActorSensitivity		
params: [axis]		
Result: [errNo, sensitivity]		
C-DLL call		
int **AMC_control_getActorSensitivity**(int deviceHandle, int axis, int* sensitivity)		
Python		
sensitivity = **[dev].control.getActorSensitivity**(axis)		
Matlab		
[sensitivity] = **control_getActorSensitivity**(axis)		
C#		
int value = [Device].**Control_GetActorSensitivity**(int axis)		
LabVIEW		
getActorSensitivity.vi		

getActorType

This function gets the type of the positioner of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	actor_type	0: linear, 1: rotator, 2: goniometer
JSON Method		
method: com.attocube.amc.control.getActorType		
params: [axis]		
Result: [errNo, actor_type]		
C-DLL call		
int **AMC_control_getActorType**(int deviceHandle, int axis, int* actor_type)		
Python		
actor_type = **[dev].control.getActorType**(axis)		
Matlab		
[actor_type] = **control_getActorType**(axis)		
C#		
int value = [Device].**Control_GetActorType**(int axis)		
LabVIEW		
getActorType.vi		

getAutoMeasure

This function returns if the automeasurement on axis enable is enabled

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enable	true: enable automeasurement, false: disable automeasurement

JSON Method
method: com.attocube.amc.control.getAutoMeasure
params: [axis]
Result: [errNo, enable]
C-DLL call
int **AMC_control_getAutoMeasure**(int deviceHandle, int axis, bool* enable)
Python
enable = **[dev].control.getAutoMeasure**(axis)
Matlab
[enable] = **control_getAutoMeasure**(axis)
C#
bool value = [Device].**Control_GetAutoMeasure**(int axis)
LabVIEW
getAutoMeasure.vi

getControlAmplitude

This function gets the amplitude of the actuator signal of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	amplitude	in mV
JSON Method		
method: com.attocube.amc.control.getControlAmplitude		
params: [axis]		
Result: [errNo, amplitude]		
C-DLL call		
int **AMC_control_getControlAmplitude**(int deviceHandle, int axis, int* amplitude)		
Python		
amplitude = **[dev].control.getControlAmplitude**(axis)		
Matlab		
[amplitude] = **control_getControlAmplitude**(axis)		
C#		
int value = [Device].**Control_GetControlAmplitude**(int axis)		
LabVIEW		
getControlAmplitude.vi		

getControlAutoReset

This function resets the position every time the reference position is detected.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	boolean
JSON Method		
method: com.attocube.amc.control.getControlAutoReset		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_control_getControlAutoReset**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].control.getControlAutoReset**(axis)		
Matlab		
[enabled] = **control_getControlAutoReset**(axis)		
C#		
bool value = [Device].**Control_GetControlAutoReset**(int axis)		
LabVIEW		
getControlAutoReset.vi		

getControlFixOutputVoltage

This function gets the DC level output of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	amplitude_mv	in mV
JSON Method		
method: com.attocube.amc.control.getControlFixOutputVoltage		
params: [axis]		
Result: [errNo, amplitude_mv]		
C-DLL call		
int **AMC_control_getControlFixOutputVoltage**(int deviceHandle, int axis, int* amplitude_mv)		
Python		
amplitude_mv = **[dev].control.getControlFixOutputVoltage**(axis)		
Matlab		
[amplitude_mv] = **control_getControlFixOutputVoltage**(axis)		
C#		
int value = [Device].**Control_GetControlFixOutputVoltage**(int axis)		
LabVIEW		
getControlFixOutputVoltage.vi		

getControlFrequency

This function gets the frequency of the actuator signal of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	frequency	in mHz
JSON Method		
method: com.attocube.amc.control.getControlFrequency		
params: [axis]		
Result: [errNo, frequency]		
C-DLL call		
int **AMC_control_getControlFrequency**(int deviceHandle, int axis, int* frequency)		
Python		
frequency = **[dev].control.getControlFrequency**(axis)		
Matlab		
[frequency] = **control_getControlFrequency**(axis)		
C#		
int value = [Device].**Control_GetControlFrequency**(int axis)		
LabVIEW		
getControlFrequency.vi		

getControlMove

This function gets the approach of the selected axis' positioner to the target position.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enable	boolean true: closed loop control enabled, false: closed loop control disabled
JSON Method		
method: com.attocube.amc.control.getControlMove		
params: [axis]		
Result: [errNo, enable]		
C-DLL call		
int **AMC_control_getControlMove**(int deviceHandle, int axis, bool* enable)		
Python		
enable = **[dev].control.getControlMove**(axis)		
Matlab		
[enable] = **control_getControlMove**(axis)		
C#		
bool value = [Device].**Control_GetControlMove**(int axis)		
LabVIEW		
getControlMove.vi		

getControlOutput

This function gets the status of the output relays of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	power status (true = enabled,false = disabled)
JSON Method		
method: com.attocube.amc.control.getControlOutput		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_control_getControlOutput**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].control.getControlOutput**(axis)		
Matlab		
[enabled] = **control_getControlOutput**(axis)		
C#		
bool value = [Device].**Control_GetControlOutput**(int axis)		
LabVIEW		
getControlOutput.vi		

getControlReferenceAutoUpdate

This function gets the status of whether the reference position is updated when the reference mark is hit.

When this function is disabled, the reference marking will be considered only the first time and after then ignored.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	boolen
JSON Method		
method: com.attocube.amc.control.getControlReferenceAutoUpdate		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_control_getControlReferenceAutoUpdate**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].control.getControlReferenceAutoUpdate**(axis)		
Matlab		
[enabled] = **control_getControlReferenceAutoUpdate**(axis)		
C#		
bool value = [Device].**Control_GetControlReferenceAutoUpdate**(int axis)		
LabVIEW		
getControlReferenceAutoUpdate.vi		

getControlTargetRange

This function gets the range around the target position in which the flag "In Target Range" becomes active.

Please note that this setting is only used to identify the target position, it has no influence on the closed loop control!

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	targetrange	in nm
JSON Method		
method: com.attocube.amc.control.getControlTargetRange		
params: [axis]		
Result: [errNo, targetrange]		
C-DLL call		
int **AMC_control_getControlTargetRange**(int deviceHandle, int axis, int* targetrange)		
Python		
targetrange = **[dev].control.getControlTargetRange**(axis)		
Matlab		
[targetrange] = **control_getControlTargetRange**(axis)		
C#		
int value = [Device].**Control_GetControlTargetRange**(int axis)		
LabVIEW		
getControlTargetRange.vi		

getCrosstalkThreshold

This function gets the threshold range and slip phase time which is used while moving another axis

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	range	in pm
	time	after slip phase which is waited until the controller is acting again in microseconds

JSON Method
method: com.attocube.amc.control.getCrosstalkThreshold
params: [axis]
Result: [errNo, range, time]
C-DLL call
int **AMC_control_getCrosstalkThreshold**(int deviceHandle, int axis, int* range, int* time)
Python
range, time = **[dev].control.getCrosstalkThreshold**(axis)
Matlab
[range, time] = **control_getCrosstalkThreshold**(axis)
C#
Tuple<int,int> value = [Device].**Control_GetCrosstalkThreshold**(int axis)
LabVIEW
getCrosstalkThreshold.vi

getCurrentOutputVoltage

This function gets the current Voltage which is applied to the Piezo.
Please note that the actual values may be slightly different (due to analog/digital converters and amplifiers etc.)

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	amplitude	in mV
JSON Method		
method: com.attocube.amc.control.getCurrentOutputVoltage		
params: [axis]		
Result: [errNo, amplitude]		
C-DLL call		
int **AMC_control_getCurrentOutputVoltage**(int deviceHandle, int axis, int* amplitude)		
Python		
amplitude = **[dev].control.getCurrentOutputVoltage**(axis)		
Matlab		
[amplitude] = **control_getCurrentOutputVoltage**(axis)		
C#		
int value = [Device].**Control_GetCurrentOutputVoltage**(int axis)		
LabVIEW		
getCurrentOutputVoltage.vi		

getEoTParameters

Gets the two parameters, that define the behavior of the eot detection (how sensitive respectively how robust it works)

Function specific parameters		
In	axis	[0|1|2] (will be ignored, if minAvgStepSize equals nil)
Out	err	err
	minAvgStepSize_nm[type=int]	this correpsonds to the "eot_threshold"-parameter
	numOfAvgedSteps[type=int]	this defines the number of steps, over which the average step size is calculated
JSON Method		
method: com.attocube.amc.control.getEoTParameters		
params: [axis]		
Result: [err, minAvgStepSize_nm[type=int], numOfAvgedSteps[type=int]]		
C-DLL call		
int **AMC_control_getEoTParameters**(int deviceHandle, int axis, int* err, int* minAvgStepSize_nm[type=int], int* numOfAvgedSteps[type=int])		
Python		
err, minAvgStepSize_nm[type=int], numOfAvgedSteps[type=int] = **[dev].control.getEoTParameters**(axis)		
Matlab		
[err, minAvgStepSize_nm[type=int], numOfAvgedSteps[type=int]] = **control_getEoTParameters**(axis)		
C#		
Tuple<int,int,int> value = [Device].**Control_GetEoTParameters**(int axis)		
LabVIEW		
getEoTParameters.vi		

getExternalSensor

This function gets whether the sensor source of closed loop is IDS

It is only available when the feature AMC/IDS closed loop has been activated

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	enabled
JSON Method		
method: com.attocube.amc.control.getExternalSensor		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_control_getExternalSensor**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].control.getExternalSensor**(axis)		
Matlab		
[enabled] = **control_getExternalSensor**(axis)		
C#		
bool value = [Device].**Control_GetExternalSensor**(int axis)		
LabVIEW		
getExternalSensor.vi		

getFinePositioningRange

This function gets the fine positioning DC-range

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	range	in nm
JSON Method		
method: com.attocube.amc.control.getFinePositioningRange		
params: [axis]		
Result: [errNo, range]		
C-DLL call		
int **AMC_control_getFinePositioningRange**(int deviceHandle, int axis, int* range)		
Python		
range = **[dev].control.getFinePositioningRange**(axis)		
Matlab		
[range] = **control_getFinePositioningRange**(axis)		
C#		
int value = [Device].**Control_GetFinePositioningRange**(int axis)		
LabVIEW		
getFinePositioningRange.vi		

getFinePositioningSlewRate

This function gets the fine positioning slew rate

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	slewrate	[0|1|2|3]
JSON Method		
method: com.attocube.amc.control.getFinePositioningSlewRate		
params: [axis]		
Result: [errNo, slewrate]		
C-DLL call		
int **AMC_control_getFinePositioningSlewRate**(int deviceHandle, int axis, int* slewrate)		
Python		
slewrate = **[dev].control.getFinePositioningSlewRate**(axis)		
Matlab		
[slewrate] = **control_getFinePositioningSlewRate**(axis)		
C#		
int value = [Device].**Control_GetFinePositioningSlewRate**(int axis)		
LabVIEW		
getFinePositioningSlewRate.vi		

getMotionControlThreshold

This function gets the threshold range within the closed-loop controlled movement stops to regulate.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	threshold	in pm
JSON Method		
method: com.attocube.amc.control.getMotionControlThreshold		
params: [axis]		
Result: [errNo, threshold]		
C-DLL call		
int **AMC_control_getMotionControlThreshold**(int deviceHandle, int axis, int* threshold)		
Python		
threshold = **[dev].control.getMotionControlThreshold**(axis)		
Matlab		
[threshold] = **control_getMotionControlThreshold**(axis)		
C#		
int value = [Device].**Control_GetMotionControlThreshold**(int axis)		
LabVIEW		
getMotionControlThreshold.vi		

getPositionsAndVoltages

Simultaneously get 3 axes positions as well as the DC offset to maximize sampling rate over network

Function specific parameters		
Out	errNo	errNo
	pos1	position of axis 1
	pos2	position of axis 2
	pos3	position of axis 3
	val1	dc voltage of of axis 1 in mV
	val2	dc voltage of of axis 2 in mV
	val3	dc voltage of of axis 3 in mV

JSON Method
method: com.attocube.amc.control.getPositionsAndVoltages
params: []
Result: [errNo, pos1, pos2, pos3, val1, val2, val3]
C-DLL call
int **AMC_control_getPositionsAndVoltages**(int deviceHandle, int* pos1, int* pos2, int* pos3, int* val1, int* val2, int* val3)
Python
pos1, pos2, pos3, val1, val2, val3 = **[dev].control.getPositionsAndVoltages**()
Matlab
[pos1, pos2, pos3, val1, val2, val3] = **control_getPositionsAndVoltages**()
C#
Tuple<int,int,int,int,int,int> value = [Device].**Control_GetPositionsAndVoltages**()
LabVIEW
getPositionsAndVoltages.vi

getReferencePosition

This function gets the reference position of the selected axis.

It can only be used in conjunction with NUM.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	position	position: For linear type actors the position is defined in nm for goniometer an rotator type actors it is µ°.

JSON Method
method: com.attocube.amc.control.getReferencePosition
params: [axis]
Result: [errNo, position]
C-DLL call
int **AMC_control_getReferencePosition**(int deviceHandle, int axis, int* position)
Python
position = **[dev].control.getReferencePosition**(axis)
Matlab
[position] = **control_getReferencePosition**(axis)
C#
int value = [Device].**Control_GetReferencePosition**(int axis)
LabVIEW
getReferencePosition.vi

getSensorDirection

This function gets whether the IDS sensor source of closed loop is inverted

It is only available when the feature AMC/IDS closed loop has been activated

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	inverted	boolen
JSON Method		
method: com.attocube.amc.control.getSensorDirection		
params: [axis]		
Result: [errNo, inverted]		
C-DLL call		
int **AMC_control_getSensorDirection**(int deviceHandle, int axis, bool* inverted)		
Python		
inverted = **[dev].control.getSensorDirection**(axis)		
Matlab		
[inverted] = **control_getSensorDirection**(axis)		
C#		
bool value = [Device].**Control_GetSensorDirection**(int axis)		
LabVIEW		
getSensorDirection.vi		

getSensorEnabled

Get sensot power supply status

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	value	true if enabled, false otherwise
JSON Method		
method: com.attocube.amc.control.getSensorEnabled		
params: [axis]		
Result: [errNo, value]		
C-DLL call		
int **AMC_control_getSensorEnabled**(int deviceHandle, int axis, bool* value)		
Python		
value = **[dev].control.getSensorEnabled**(axis)		
Matlab		
[value] = **control_getSensorEnabled**(axis)		
C#		
bool value = [Device].**Control_GetSensorEnabled**(int axis)		
LabVIEW		
getSensorEnabled.vi		

getStatusMovingAllAxes

Get Status of all axes, see getStatusMoving for coding of the values

Function specific parameters		
Out	errNo	errNo
	moving1	status of axis 1
	moving2	status of axis 2
	moving3	status of axis 3
JSON Method		
method: com.attocube.amc.control.getStatusMovingAllAxes		
params: []		
Result: [errNo, moving1, moving2, moving3]		
C-DLL call		
int **AMC_control_getStatusMovingAllAxes**(int deviceHandle, int* moving1, int* moving2, int* moving3)		
Python		
moving1, moving2, moving3 = **[dev].control.getStatusMovingAllAxes**()		
Matlab		
[moving1, moving2, moving3] = **control_getStatusMovingAllAxes**()		
C#		
Tuple<int,int,int> value = [Device].**Control_GetStatusMovingAllAxes**()		
LabVIEW		
getStatusMovingAllAxes.vi		

searchReferencePosition

This function searches for the reference position of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.searchReferencePosition		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_control_searchReferencePosition**(int deviceHandle, int axis)		
Python		
[dev].control.searchReferencePosition(axis)		
Matlab		
[] = **control_searchReferencePosition**(axis)		
C#		
void value = [Device].**Control_SearchReferencePosition**(int axis)		
LabVIEW		
searchReferencePosition.vi		

setActorParametersByName

This function sets the name for the positioner on the selected axis. The possible names can be retrieved by executing getPositionersList

Function specific parameters		
In	axis	[0|1|2]
	actorname	name of the actor
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setActorParametersByName		
params: [axis, actorname]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setActorParametersByName**(int deviceHandle, int axis, const char* actorname)		
Python		
[dev].control.setActorParametersByName(axis, actorname)		
Matlab		
[] = **control_setActorParametersByName**(axis, actorname)		
C#		
void value = [Device].**Control_SetActorParametersByName**(int axis, string actorname)		
LabVIEW		
setActorParametersByName.vi		

setActorParametersJson

This function can be used to select a positioner from the standard list and overwrite the current setting. In addition, certain parameters that need to be changed can be transferred.

For all other parameters, either the values from the standard list or the previously changed values continue to apply.

Function specific parameters		
In	axis	[0|1|2]
	json_dict	dict with override params
Out	errNo	errorCode

JSON Method
method: com.attocube.amc.control.setActorParametersJson
params: [axis, json_dict]
Result: [errNo]
C-DLL call
int **AMC_control_setActorParametersJson**(int deviceHandle, int axis, const char* json_dict)
Python
[dev].control.setActorParametersJson(axis, json_dict)
Matlab
[] = **control_setActorParametersJson**(axis, json_dict)
C#
void value = [Device].**Control_SetActorParametersJson**(int axis, string json_dict)
LabVIEW
setActorParametersJson.vi

setActorSensitivity

Control the actor parameter closed loop sensitivity

Function specific parameters		
In	axis	[0|1|2]
In	sensitivity	[1 .. 10] → 1: fast approach … 10: slow approach The behavior changes exponentially.
Out	errNo	errNo

JSON Method
method: com.attocube.amc.control.setActorSensitivity
params: [axis, sensitivity]
Result: [errNo]
C-DLL call
int **AMC_control_setActorSensitivity**(int deviceHandle, int axis, int sensitivity)
Python
[dev].control.setActorSensitivity(axis, sensitivity)
Matlab
[] = **control_setActorSensitivity**(axis, sensitivity)
C#
void value = [Device].**Control_SetActorSensitivity**(int axis, int sensitivity)
LabVIEW
setActorSensitivity.vi

setAutoMeasure

This function enables/disables the automatic C/R measurement on axis enable

<table>
<tr><th colspan="3">Function specific parameters</th></tr>
<tr><td rowspan="2">In</td><td>axis</td><td>[0|1|2]</td></tr>
<tr><td>enable</td><td>true: enable automeasurement, false: disable automeasurement</td></tr>
<tr><td>Out</td><td>errNo</td><td>errNo</td></tr>
<tr><th colspan="3">JSON Method</th></tr>
<tr><td colspan="3">method: com.attocube.amc.control.setAutoMeasure</td></tr>
<tr><td colspan="3">params: [axis, enable]</td></tr>
<tr><td colspan="3">Result: [errNo]</td></tr>
<tr><th colspan="3">C-DLL call</th></tr>
<tr><td colspan="3">int AMC_control_setAutoMeasure(int deviceHandle, int axis, bool enable)</td></tr>
<tr><th colspan="3">Python</th></tr>
<tr><td colspan="3">[dev].control.setAutoMeasure(axis, enable)</td></tr>
<tr><th colspan="3">Matlab</th></tr>
<tr><td colspan="3">[] = control_setAutoMeasure(axis, enable)</td></tr>
<tr><th colspan="3">C#</th></tr>
<tr><td colspan="3">void value = [Device].Control_SetAutoMeasure(int axis, bool enable)</td></tr>
<tr><th colspan="3">LabVIEW</th></tr>
<tr><td colspan="3">setAutoMeasure.vi</td></tr>
</table>

setControlAmplitude

This function sets the amplitude of the actuator signal of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
	amplitude	in mV
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setControlAmplitude		
params: [axis, amplitude]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setControlAmplitude**(int deviceHandle, int axis, int amplitude)		
Python		
[dev].control.setControlAmplitude(axis, amplitude)		
Matlab		
[] = **control_setControlAmplitude**(axis, amplitude)		
C#		
void value = [Device].**Control_SetControlAmplitude**(int axis, int amplitude)		
LabVIEW		
setControlAmplitude.vi		

setControlAutoReset

This function controls the behaviour, that the current position (which can be retrieved with a getPosition-call) is reset every time the reference mark of the positioner is detected.

This function can only be used in conjunction with NUM.

Function specific parameters		
In	axis	[0|1|2]
	enable	boolean
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setControlAutoReset		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setControlAutoReset**(int deviceHandle, int axis, bool enable)		
Python		
[dev].control.setControlAutoReset(axis, enable)		
Matlab		
[] = **control_setControlAutoReset**(axis, enable)		
C#		
void value = [Device].**Control_SetControlAutoReset**(int axis, bool enable)		
LabVIEW		
setControlAutoReset.vi		

setControlFixOutputVoltage

This function sets the DC level output of the selected axis.

Function specific parameters		
In	axis	[0|1|2]
	amplitude_mv	in mV
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setControlFixOutputVoltage		
params: [axis, amplitude_mv]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setControlFixOutputVoltage**(int deviceHandle, int axis, int amplitude_mv)		
Python		
[dev].control.setControlFixOutputVoltage(axis, amplitude_mv)		
Matlab		
[] = **control_setControlFixOutputVoltage**(axis, amplitude_mv)		
C#		
void value = [Device].**Control_SetControlFixOutputVoltage**(int axis, int amplitude_mv)		
LabVIEW		
setControlFixOutputVoltage.vi		

setControlFrequency

This function sets the frequency of the actuator signal of the selected axis.

Note: Approximate the slewrate of the motion controller according to Input Frequency

Function specific parameters		
In	axis	[0|1|2]
	frequency	in mHz
Out	errNo	errNo

JSON Method
method: com.attocube.amc.control.setControlFrequency
params: [axis, frequency]
Result: [errNo]
C-DLL call
int **AMC_control_setControlFrequency**(int deviceHandle, int axis, int frequency)
Python
[dev].control.setControlFrequency(axis, frequency)
Matlab
[] = **control_setControlFrequency**(axis, frequency)
C#
void value = [Device].**Control_SetControlFrequency**(int axis, int frequency)
LabVIEW
setControlFrequency.vi

setControlMove

This function sets the approach of the selected axis' positioner to the target position.

Function specific parameters		
In	axis	[0|1|2]
	enable	boolean true: eanble the approach , false: disable the approach
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setControlMove		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setControlMove**(int deviceHandle, int axis, bool enable)		
Python		
[dev].control.setControlMove(axis, enable)		
Matlab		
[] = **control_setControlMove**(axis, enable)		
C#		
void value = [Device].**Control_SetControlMove**(int axis, bool enable)		
LabVIEW		
setControlMove.vi		

setControlOutput

This function sets the status of the output relays of the selected axis.

Enable only if cable is connected and FlyBack is enabled

use a PWM startup of 1sec

Function specific parameters		
In	axis	[0|1|2]
	enable	true: enable drives, false: disable drives
Out	errNo	errNo

JSON Method
method: com.attocube.amc.control.setControlOutput
params: [axis, enable]
Result: [errNo]
C-DLL call
int **AMC_control_setControlOutput**(int deviceHandle, int axis, bool enable)
Python
[dev].control.setControlOutput(axis, enable)
Matlab
[] = **control_setControlOutput**(axis, enable)
C#
void value = [Device].**Control_SetControlOutput**(int axis, bool enable)
LabVIEW
setControlOutput.vi

setControlReferenceAutoUpdate

This function sets the status of whether the reference position is updated when the reference mark is hit.

When this function is disabled, the reference marking will be considered only the first time and after then ignored.

This function can only be used in conjunction with NUM.

Function specific parameters		
In	axis	[0|1|2]
	enable	boolean
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setControlReferenceAutoUpdate		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setControlReferenceAutoUpdate**(int deviceHandle, int axis, bool enable)		
Python		
[dev].control.setControlReferenceAutoUpdate(axis, enable)		
Matlab		
[] = **control_setControlReferenceAutoUpdate**(axis, enable)		
C#		
void value = [Device].**Control_SetControlReferenceAutoUpdate**(int axis, bool enable)		
LabVIEW		
setControlReferenceAutoUpdate.vi		

setControlTargetRange

This function sets the range around the target position in which the flag "In Target Range" (see VIII.7.a) becomes active.

Please note that this setting is only used to identify the target position, it has no influence on the closed loop control!

Function specific parameters		
In	axis	[0|1|2]
	range	in nm
Out	errNo	errNo

JSON Method
method: com.attocube.amc.control.setControlTargetRange
params: [axis, range]
Result: [errNo]
C-DLL call
int **AMC_control_setControlTargetRange**(int deviceHandle, int axis, int range)
Python
[dev].control.setControlTargetRange(axis, range)
Matlab
[] = **control_setControlTargetRange**(axis, range)
C#
void value = [Device].**Control_SetControlTargetRange**(int axis, int range)
LabVIEW
setControlTargetRange.vi

setCrosstalkThreshold

This function sets the threshold range to avoid axis-crosstalk and slip phase time which is used while moving another axis

Function specific parameters		
In	axis	[0|1|2]
	threshold	[max:2147483647][pm]; has to be greater than the motion-control-threshold
	slipphasetime	[min=0,max=65535][us] time after slip phase which is waited until the controller acts again
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setCrosstalkThreshold		
params: [axis, threshold, slipphasetime]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setCrosstalkThreshold**(int deviceHandle, int axis, int threshold, int slipphasetime)		
Python		
[dev].control.setCrosstalkThreshold(axis, threshold, slipphasetime)		
Matlab		
[] = **control_setCrosstalkThreshold**(axis, threshold, slipphasetime)		
C#		
void value = [Device].**Control_SetCrosstalkThreshold**(int axis, int threshold, int slipphasetime)		
LabVIEW		
setCrosstalkThreshold.vi		

setEoTParameters

Sets the two parameters, that define the behavior of the eot detection (how sensitive respectively how robust it works)

Function specific parameters		
In	axis	[0|1|2] (will be ignored, if minAvgStepSize equals nil)
	minAvgStepSize_nm	[type=int] this correpsonds to the "eot_threshold"-parameter
	numOfAvgedSteps	[type=int] this defines the number of steps, over which the average step size is calculated
Out	err	err

JSON Method
method: com.attocube.amc.control.setEoTParameters
params: [axis, minAvgStepSize_nm, numOfAvgedSteps]
Result: [err]
C-DLL call
int **AMC_control_setEoTParameters**(int deviceHandle, int axis, int minAvgStepSize_nm, int numOfAvgedSteps, int* err)
Python
err = **[dev].control.setEoTParameters**(axis, minAvgStepSize_nm, numOfAvgedSteps)
Matlab
[err] = **control_setEoTParameters**(axis, minAvgStepSize_nm, numOfAvgedSteps)
C#
int value = [Device].**Control_SetEoTParameters**(int axis, int minAvgStepSize_nm, int numOfAvgedSteps)
LabVIEW
setEoTParameters.vi

setExternalSensor

This function sets the sensor source of closed loop to the IDS when enabled. Otherwise the normal AMC Sensor depending on the configuration (e.g. NUM or RES) is used

It is only available when the feature AMC/IDS closed loop has been activated

Function specific parameters		
In	axis	[0|1|2]
	enabled	
Out	warningNo	Warning code, can be converted into a string using the errorNumberToString function
JSON Method		
method: com.attocube.amc.control.setExternalSensor		
params: [axis, enabled]		
Result: [warningNo]		
C-DLL call		
int **AMC_control_setExternalSensor**(int deviceHandle, int axis, bool enabled, int* warningNo)		
Python		
warningNo = **[dev].control.setExternalSensor**(axis, enabled)		
Matlab		
[warningNo] = **control_setExternalSensor**(axis, enabled)		
C#		
int value = [Device].**Control_SetExternalSensor**(int axis, bool enabled)		
LabVIEW		
setExternalSensor.vi		

setFinePositioningRange

This function sets the fine positioning DC-range

Function specific parameters		
In	axis	[0|1|2]
	range	in nm
Out	errNo	errNo

JSON Method
method: com.attocube.amc.control.setFinePositioningRange
params: [axis, range]
Result: [errNo]
C-DLL call
int **AMC_control_setFinePositioningRange**(int deviceHandle, int axis, int range)
Python
[dev].control.setFinePositioningRange(axis, range)
Matlab
[] = **control_setFinePositioningRange**(axis, range)
C#
void value = [Device].**Control_SetFinePositioningRange**(int axis, int range)
LabVIEW
setFinePositioningRange.vi

setFinePositioningSlewRate

This function sets the fine positioning slew rate

Function specific parameters		
In	axis	[0|1|2]
	slewrate	[0|1|2|3]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setFinePositioningSlewRate		
params: [axis, slewrate]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setFinePositioningSlewRate**(int deviceHandle, int axis, int slewrate)		
Python		
[dev].control.setFinePositioningSlewRate(axis, slewrate)		
Matlab		
[] = **control_setFinePositioningSlewRate**(axis, slewrate)		
C#		
void value = [Device].**Control_SetFinePositioningSlewRate**(int axis, int slewrate)		
LabVIEW		
setFinePositioningSlewRate.vi		

setMotionControlThreshold

This function sets the threshold range within the closed-loop controlled movement stops to regulate. Default depends on connected sensor type

Function specific parameters		
In	axis	[0|1|2]
	threshold	in pm
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setMotionControlThreshold		
params: [axis, threshold]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setMotionControlThreshold**(int deviceHandle, int axis, int threshold)		
Python		
[dev].control.setMotionControlThreshold(axis, threshold)		
Matlab		
[] = **control_setMotionControlThreshold**(axis, threshold)		
C#		
void value = [Device].**Control_SetMotionControlThreshold**(int axis, int threshold)		
LabVIEW		
setMotionControlThreshold.vi		

setReset

This function resets the actual position of the selected axis given by the NUM sensor to zero and marks the reference position as invalid.
It does not work for RES positioners and positions read by IDS.
For IDS, use com.attocube.ids.displacement.resetAxis() or com.attocube.amc.amcids.resetIdsAxis() instead.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setReset		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setReset**(int deviceHandle, int axis)		
Python		
[dev].control.setReset(axis)		
Matlab		
[] = **control_setReset**(axis)		
C#		
void value = [Device].**Control_SetReset**(int axis)		
LabVIEW		
setReset.vi		

setSensorDirection

This function sets the IDS sensor source of closed loop to inverted when true.
It is only available when the feature AMC/IDS closed loop has been activated

Function specific parameters		
In	axis	[0|1|2]
	inverted	
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setSensorDirection		
params: [axis, inverted]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setSensorDirection**(int deviceHandle, int axis, bool inverted)		
Python		
[dev].control.setSensorDirection(axis, inverted)		
Matlab		
[] = **control_setSensorDirection**(axis, inverted)		
C#		
void value = [Device].**Control_SetSensorDirection**(int axis, bool inverted)		
LabVIEW		
setSensorDirection.vi		

setSensorEnabled

Set sensor power supply status, can be switched off to save heat generated by sensor [NUM or RES]

Positions retrieved will be invalid when activating this, so closed-loop control should be switched off beforehand

Function specific parameters		
In	axis	[0|1|2]
	value	true if enabled, false otherwise
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.control.setSensorEnabled		
params: [axis, value]		
Result: [errNo]		
C-DLL call		
int **AMC_control_setSensorEnabled**(int deviceHandle, int axis, bool value)		
Python		
[dev].control.setSensorEnabled(axis, value)		
Matlab		
[] = **control_setSensorEnabled**(axis, value)		
C#		
void value = [Device].**Control_SetSensorEnabled**(int axis, bool value)		
LabVIEW		
setSensorEnabled.vi		

3.4 Description

checkChassisNbr

Get Chassis and Slot Number, only works when AMC is within a Rack

Function specific parameters		
Out	errNo	errorCode
	slotNbr	slotNbr
	chassisNbr	chassisNbr
JSON Method		
method: com.attocube.amc.description.checkChassisNbr		
params: []		
Result: [errNo, slotNbr, chassisNbr]		
C-DLL call		
int **AMC_description_checkChassisNbr**(int deviceHandle, int* slotNbr, int* chassisNbr)		
Python		
slotNbr, chassisNbr = **[dev].description.checkChassisNbr**()		
Matlab		
[slotNbr, chassisNbr] = **description_checkChassisNbr**()		
C#		
Tuple<int,int> value = [Device].**Description_CheckChassisNbr**()		
LabVIEW		
checkChassisNbr.vi		

getDeviceType

This function gets the device type based on its EEPROM configuration.

<table>
<tr><th colspan="3">Function specific parameters</th></tr>
<tr><td rowspan="2">Out</td><td>errNo</td><td>errNo</td></tr>
<tr><td>devicetype</td><td>Device name (AMC100, AMC150, AMC300) with attached feature (AMC100\\NUM, AMC100\\NUM\\PRO)</td></tr>
<tr><th colspan="3">JSON Method</th></tr>
<tr><td colspan="3">method: com.attocube.amc.description.getDeviceType</td></tr>
<tr><td colspan="3">params: []</td></tr>
<tr><td colspan="3">Result: [errNo, devicetype]</td></tr>
<tr><th colspan="3">C-DLL call</th></tr>
<tr><td colspan="3">int AMC_description_getDeviceType(int deviceHandle, char* devicetype, int size0)</td></tr>
<tr><th colspan="3">Python</th></tr>
<tr><td colspan="3">devicetype = [dev].description.getDeviceType()</td></tr>
<tr><th colspan="3">Matlab</th></tr>
<tr><td colspan="3">[devicetype] = description_getDeviceType()</td></tr>
<tr><th colspan="3">C#</th></tr>
<tr><td colspan="3">string value = [Device].Description_GetDeviceType()</td></tr>
<tr><th colspan="3">LabVIEW</th></tr>
<tr><td colspan="3">getDeviceType.vi</td></tr>
</table>

getFeaturesActivated

Get the activated features and return as a string

Function specific parameters		
Out	errNo	errNo
	features	activated on device concatenated by comma e.g. Closed loop Operation, Pro, Wireless Controller, IO
JSON Method		
method: com.attocube.amc.description.getFeaturesActivated		
params: []		
Result: [errNo, features]		
C-DLL call		
int **AMC_description_getFeaturesActivated**(int deviceHandle, char* features, int size0)		
Python		
features = **[dev].description.getFeaturesActivated**()		
Matlab		
[features] = **description_getFeaturesActivated**()		
C#		
string value = [Device].**Description_GetFeaturesActivated**()		
LabVIEW		
getFeaturesActivated.vi		

getPositionersList

This function reads the actor names that can be connected to the device.

Function specific parameters		
Out	errNo	errNo
	PositionersList	PositionersList
JSON Method		
method: com.attocube.amc.description.getPositionersList		
params: []		
Result: [errNo, PositionersList]		
C-DLL call		
int **AMC_description_getPositionersList**(int deviceHandle, char* PositionersList, int size0)		
Python		
PositionersList = **[dev].description.getPositionersList**()		
Matlab		
[PositionersList] = **description_getPositionersList**()		
C#		
string value = [Device].**Description_GetPositionersList**()		
LabVIEW		
getPositionersList.vi		

3.5 Diagnostic

getDiagnosticPower

Returns the current power consumption

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	power	power
JSON Method		
method: com.attocube.amc.diagnostic.getDiagnosticPower		
params: [axis]		
Result: [errNo, power]		
C-DLL call		
int **AMC_diagnostic_getDiagnosticPower**(int deviceHandle, int axis, int* power)		
Python		
power = **[dev].diagnostic.getDiagnosticPower**(axis)		
Matlab		
[power] = **diagnostic_getDiagnosticPower**(axis)		
C#		
int value = [Device].**Diagnostic_GetDiagnosticPower**(int axis)		
LabVIEW		
getDiagnosticPower.vi		

getDiagnosticResults

Returns the results of the last diagnostic run and an error, if there was no run, it is currently running or the run failed

<table>
<tr><th colspan="3">Function specific parameters</th></tr>
<tr><td>In</td><td>axis</td><td>[0|1|2]</td></tr>
<tr><td rowspan="3">Out</td><td>errNo</td><td>errNo</td></tr>
<tr><td>capacity</td><td>in nF</td></tr>
<tr><td>resistance</td><td>in Ohm</td></tr>
<tr><th colspan="3">JSON Method</th></tr>
<tr><td colspan="3">method: com.attocube.amc.diagnostic.getDiagnosticResults</td></tr>
<tr><td colspan="3">params: [axis]</td></tr>
<tr><td colspan="3">Result: [errNo, capacity, resistance]</td></tr>
<tr><th colspan="3">C-DLL call</th></tr>
<tr><td colspan="3">int AMC_diagnostic_getDiagnosticResults(int deviceHandle, int axis, int* capacity, int* resistance)</td></tr>
<tr><th colspan="3">Python</th></tr>
<tr><td colspan="3">capacity, resistance = [dev].diagnostic.getDiagnosticResults(axis)</td></tr>
<tr><th colspan="3">Matlab</th></tr>
<tr><td colspan="3">[capacity, resistance] = diagnostic_getDiagnosticResults(axis)</td></tr>
<tr><th colspan="3">C#</th></tr>
<tr><td colspan="3">Tuple<int,int> value = [Device].Diagnostic_GetDiagnosticResults(int axis)</td></tr>
<tr><th colspan="3">LabVIEW</th></tr>
<tr><td colspan="3">getDiagnosticResults.vi</td></tr>
</table>

getDiagnosticStepSize

Performs 10 steps in forward and backward and calculates the average step size in both directions on a specific axis

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	stepsize_fwd	stepsize_fwd
	stepsize_bwd	stepsize_bwd

JSON Method
method: com.attocube.amc.diagnostic.getDiagnosticStepSize
params: [axis]
Result: [errNo, stepsize_fwd, stepsize_bwd]
C-DLL call
int **AMC_diagnostic_getDiagnosticStepSize**(int deviceHandle, int axis, int* stepsize_fwd, int* stepsize_bwd)
Python
stepsize_fwd, stepsize_bwd = **[dev].diagnostic.getDiagnosticStepSize**(axis)
Matlab
[stepsize_fwd, stepsize_bwd] = **diagnostic_getDiagnosticStepSize**(axis)
C#
Tuple<int,int> value = [Device].**Diagnostic_GetDiagnosticStepSize**(int axis)
LabVIEW
getDiagnosticStepSize.vi

getDiagnosticTemperature

Returns the current axis temperature

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	temperature	temperature

JSON Method
method: com.attocube.amc.diagnostic.getDiagnosticTemperature
params: [axis]
Result: [errNo, temperature]
C-DLL call
int **AMC_diagnostic_getDiagnosticTemperature**(int deviceHandle, int axis, int* temperature)
Python
temperature = **[dev].diagnostic.getDiagnosticTemperature**(axis)
Matlab
[temperature] = **diagnostic_getDiagnosticTemperature**(axis)
C#
int value = [Device].**Diagnostic_GetDiagnosticTemperature**(int axis)
LabVIEW
getDiagnosticTemperature.vi

startDiagnostic

Start the diagnosis procedure for the given axis

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.diagnostic.startDiagnostic		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_diagnostic_startDiagnostic**(int deviceHandle, int axis)		
Python		
[dev].diagnostic.startDiagnostic(axis)		
Matlab		
[] = **diagnostic_startDiagnostic**(axis)		
C#		
void value = [Device].**Diagnostic_StartDiagnostic**(int axis)		
LabVIEW		
startDiagnostic.vi		

3.6 Move

getControlContinuousBkwd

This function gets the axis' movement status in backward direction.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	true if movement backward is active , false otherwise
JSON Method		
method: com.attocube.amc.move.getControlContinuousBkwd		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_move_getControlContinuousBkwd**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].move.getControlContinuousBkwd**(axis)		
Matlab		
[enabled] = **move_getControlContinuousBkwd**(axis)		
C#		
bool value = [Device].**Move_GetControlContinuousBkwd**(int axis)		
LabVIEW		
getControlContinuousBkwd.vi		

getControlContinuousFwd

This function gets the axis' movement status in positive direction.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	true if movement Fwd is active, false otherwise
JSON Method		
method: com.attocube.amc.move.getControlContinuousFwd		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_move_getControlContinuousFwd**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].move.getControlContinuousFwd**(axis)		
Matlab		
[enabled] = **move_getControlContinuousFwd**(axis)		
C#		
bool value = [Device].**Move_GetControlContinuousFwd**(int axis)		
LabVIEW		
getControlContinuousFwd.vi		

getControlEotOutputDeactive

This function gets the output applied to the selected axis on the end of travel.
/PRO feature.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	If true, the output of the axis will be deactivated on positive EOT detection.
JSON Method		
method: com.attocube.amc.move.getControlEotOutputDeactive		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_move_getControlEotOutputDeactive**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].move.getControlEotOutputDeactive**(axis)		
Matlab		
[enabled] = **move_getControlEotOutputDeactive**(axis)		
C#		
bool value = [Device].**Move_GetControlEotOutputDeactive**(int axis)		
LabVIEW		
getControlEotOutputDeactive.vi		

getControlTargetPosition

This function gets the target position for the movement on the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	position	defined in nm for goniometer an rotator type actors it is µ°.
JSON Method		
method: com.attocube.amc.move.getControlTargetPosition		
params: [axis]		
Result: [errNo, position]		
C-DLL call		
int **AMC_move_getControlTargetPosition**(int deviceHandle, int axis, double* position)		
Python		
position = **[dev].move.getControlTargetPosition**(axis)		
Matlab		
[position] = **move_getControlTargetPosition**(axis)		
C#		
double value = [Device].**Move_GetControlTargetPosition**(int axis)		
LabVIEW		
getControlTargetPosition.vi		

getGroundAxis

Checks if the axis piezo drive is actively grounded

only in AMC300

Function specific parameters		
In	axis	montion controler axis [0|1|2]
Out	errNo	0 or error
	grounded	true or false
JSON Method		
method: com.attocube.amc.move.getGroundAxis		
params: [axis]		
Result: [errNo, grounded]		
C-DLL call		
int **AMC_move_getGroundAxis**(int deviceHandle, int axis, bool* grounded)		
Python		
grounded = **[dev].move.getGroundAxis**(axis)		
Matlab		
[grounded] = **move_getGroundAxis**(axis)		
C#		
bool value = [Device].**Move_GetGroundAxis**(int axis)		
LabVIEW		
getGroundAxis.vi		

getGroundAxisAutoOnTarget

Pull axis piezo drive to GND if positioner is in ground target range

only in AMC300

Function specific parameters		
In	axis	montion controler axis [0|1|2]
Out	errNo	0 or error
	value	true or false
JSON Method		
method: com.attocube.amc.move.getGroundAxisAutoOnTarget		
params: [axis]		
Result: [errNo, value]		
C-DLL call		
int **AMC_move_getGroundAxisAutoOnTarget**(int deviceHandle, int axis, bool* value)		
Python		
value = **[dev].move.getGroundAxisAutoOnTarget**(axis)		
Matlab		
[value] = **move_getGroundAxisAutoOnTarget**(axis)		
C#		
bool value = [Device].**Move_GetGroundAxisAutoOnTarget**(int axis)		
LabVIEW		
getGroundAxisAutoOnTarget.vi		

getGroundTargetRange

Retrieves the range around the target position in which the auto grounding becomes active.

only in AMC300

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	targetrange	in nm
JSON Method		
method: com.attocube.amc.move.getGroundTargetRange		
params: [axis]		
Result: [errNo, targetrange]		
C-DLL call		
int **AMC_move_getGroundTargetRange**(int deviceHandle, int axis, int* targetrange)		
Python		
targetrange = **[dev].move.getGroundTargetRange**(axis)		
Matlab		
[targetrange] = **move_getGroundTargetRange**(axis)		
C#		
int value = [Device].**Move_GetGroundTargetRange**(int axis)		
LabVIEW		
getGroundTargetRange.vi		

getNSteps

This function gets the number of Steps in desired direction.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	nbrstep	nbrstep
JSON Method		
method: com.attocube.amc.move.getNSteps		
params: [axis]		
Result: [errNo, nbrstep]		
C-DLL call		
int **AMC_move_getNSteps**(int deviceHandle, int axis, int* nbrstep)		
Python		
nbrstep = **[dev].move.getNSteps**(axis)		
Matlab		
[nbrstep] = **move_getNSteps**(axis)		
C#		
int value = [Device].**Move_GetNSteps**(int axis)		
LabVIEW		
getNSteps.vi		

getPosition

This function gets the current position of the positioner on the selected axis.
The axis on the web application are indexed from 1 to 3

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	position	defined in nm for goniometer an rotator type actors it is µ°.

JSON Method
method: com.attocube.amc.move.getPosition
params: [axis]
Result: [errNo, position]
C-DLL call
int **AMC_move_getPosition**(int deviceHandle, int axis, double* position)
Python
position = **[dev].move.getPosition**(axis)
Matlab
[position] = **move_getPosition**(axis)
C#
double value = [Device].**Move_GetPosition**(int axis)
LabVIEW
getPosition.vi

getPositionWithTime

This function gets the current position of the positioner and provides time-information to the position.

The time information refers to the elapsed time since the last reboot of the device in microseconds.

The axis on the web application are indexed from 1 to 3

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	Monotnoic_time_usec	Monotnoic_time_usec: elapsed time in microseconds since last reboot of device
	position	defined in nm for goniometer an rotator type actors it is µ°.
JSON Method		
method: com.attocube.amc.move.getPositionWithTime		
params: [axis]		
Result: [errNo, Monotnoic_time_usec, position]		
C-DLL call		
int **AMC_move_getPositionWithTime**(int deviceHandle, int axis, double* Monotnoic_time_usec, double* position)		
Python		
Monotnoic_time_usec, position = **[dev].move.getPositionWithTime**(axis)		
Matlab		
[Monotnoic_time_usec, position] = **move_getPositionWithTime**(axis)		
C#		
Tuple<double,double> value = [Device].**Move_GetPositionWithTime**(int axis)		
LabVIEW		
getPositionWithTime.vi		

getPositionWithTime_32Bit

This function gets the current position of the positioner and provides time-information to the position.

The time information refers to the elapsed time since the last reboot of the device.

This function is to be used if 64 bit numbers are not supported by your platform

It returns two 32 bit numbers as time information instead of one number of 64 Bit (see description of return parameters).

The axis on the web application are indexed from 1 to 3

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	Monotonic_time_sec	Monotonic_time_sec: seconds passed since last reboot of device
	Monotonic_time_nsec	Monotonic_time_nsec: fractional seconds of Monotonic_time_sec
	position	defined in nm for goniometer an rotator type actors it is µ°.
JSON Method		
method: com.attocube.amc.move.getPositionWithTime_32Bit		
params: [axis]		
Result: [errNo, Monotonic_time_sec, Monotonic_time_nsec, position]		
C-DLL call		
int **AMC_move_getPositionWithTime_32Bit**(int deviceHandle, int axis, int* Monotonic_time_sec, int* Monotonic_time_nsec, double* position)		
Python		
Monotonic_time_sec, Monotonic_time_nsec, position = **[dev].move.getPositionWithTime_32Bit**(axis)		
Matlab		
[Monotonic_time_sec, Monotonic_time_nsec, position] = **move_getPositionWithTime_32Bit**(axis)		
C#		
Tuple<int,int,double> value = [Device].**Move_GetPositionWithTime_32Bit**(int axis)		
LabVIEW		
getPositionWithTime_32Bit.vi		

moveReference

This function starts an approach to the reference position. A running motion command is aborted; closed loop moving is switched on. Requires a valid reference position.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.moveReference		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_move_moveReference**(int deviceHandle, int axis)		
Python		
[dev].move.moveReference(axis)		
Matlab		
[] = **move_moveReference**(axis)		
C#		
void value = [Device].**Move_MoveReference**(int axis)		
LabVIEW		
moveReference.vi		

performNSteps

Perform the OL command for N steps

Function specific parameters		
In	axis	[0|1|2]
	backward	Selects the desired direction. False triggers a forward step, true a backward step
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.performNSteps		
params: [axis, backward]		
Result: [errNo]		
C-DLL call		
int **AMC_move_performNSteps**(int deviceHandle, int axis, bool backward)		
Python		
[dev].move.performNSteps(axis, backward)		
Matlab		
[] = **move_performNSteps**(axis, backward)		
C#		
void value = [Device].**Move_PerformNSteps**(int axis, bool backward)		
LabVIEW		
performNSteps.vi		

setControlContinuousBkwd

This function sets a continuous movement on the selected axis in backward direction.

Function specific parameters		
In	axis	[0|1|2]
In	enable	If enabled a present movement in the opposite direction is stopped. The parameter "false" stops all movement of the axis regardless its direction
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.setControlContinuousBkwd		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setControlContinuousBkwd**(int deviceHandle, int axis, bool enable)		
Python		
[dev].move.setControlContinuousBkwd(axis, enable)		
Matlab		
[] = **move_setControlContinuousBkwd**(axis, enable)		
C#		
void value = [Device].**Move_SetControlContinuousBkwd**(int axis, bool enable)		
LabVIEW		
setControlContinuousBkwd.vi		

setControlContinuousFwd

This function sets a continuous movement on the selected axis in positive direction.

Function specific parameters		
In	axis	[0|1|2]
	enable	If enabled a present movement in the opposite direction is stopped. The parameter "false" stops all movement of the axis regardless its direction.
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.setControlContinuousFwd		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setControlContinuousFwd**(int deviceHandle, int axis, bool enable)		
Python		
[dev].move.setControlContinuousFwd(axis, enable)		
Matlab		
[] = **move_setControlContinuousFwd**(axis, enable)		
C#		
void value = [Device].**Move_SetControlContinuousFwd**(int axis, bool enable)		
LabVIEW		
setControlContinuousFwd.vi		

setControlEotOutputDeactive

This function sets the output applied to the selected axis on the end of travel.

Function specific parameters		
In	axis	[0|1|2]
	enable	if enabled, the output of the axis will be deactivated on positive EOT detection.
Out	errNo	errNo

JSON Method
method: com.attocube.amc.move.setControlEotOutputDeactive
params: [axis, enable]
Result: [errNo]
C-DLL call
int **AMC_move_setControlEotOutputDeactive**(int deviceHandle, int axis, bool enable)
Python
[dev].move.setControlEotOutputDeactive(axis, enable)
Matlab
[] = **move_setControlEotOutputDeactive**(axis, enable)
C#
void value = [Device].**Move_SetControlEotOutputDeactive**(int axis, bool enable)
LabVIEW
setControlEotOutputDeactive.vi

setControlTargetPosition

This function sets the target position for the movement on the selected axis.
careful: the maximum positon in nm is 2**47/1000

Function specific parameters		
In	axis	[0|1|2]
	target	absolute position : For linear type actors the position is defined in nm for goniometer an rotator type actors it is µ°.
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.setControlTargetPosition		
params: [axis, target]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setControlTargetPosition**(int deviceHandle, int axis, double target)		
Python		
[dev].move.setControlTargetPosition(axis, target)		
Matlab		
[] = **move_setControlTargetPosition**(axis, target)		
C#		
void value = [Device].**Move_SetControlTargetPosition**(int axis, double target)		
LabVIEW		
setControlTargetPosition.vi		

setGroundAxis

Pull axis piezo drive to GND actively

only in AMC300

this is used in MIC-Mode

Function specific parameters		
In	axis	motion controler axis [0|1|2]
	enabled	true or false
Out	errNo	0 or error
JSON Method		
method: com.attocube.amc.move.setGroundAxis		
params: [axis, enabled]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setGroundAxis**(int deviceHandle, int axis, bool enabled)		
Python		
[dev].move.setGroundAxis(axis, enabled)		
Matlab		
[] = **move_setGroundAxis**(axis, enabled)		
C#		
void value = [Device].**Move_SetGroundAxis**(int axis, bool enabled)		
LabVIEW		
setGroundAxis.vi		

setGroundAxisAutoOnTarget

Pull axis piezo drive to GND actively if positioner is in ground target range

only in AMC300

this is used in MIC-Mode

Function specific parameters		
In	axis	montion controler axis [0|1|2]
	enabled	true or false
Out	errNo	0 or error
JSON Method		
method: com.attocube.amc.move.setGroundAxisAutoOnTarget		
params: [axis, enabled]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setGroundAxisAutoOnTarget**(int deviceHandle, int axis, bool enabled)		
Python		
[dev].move.setGroundAxisAutoOnTarget(axis, enabled)		
Matlab		
[] = **move_setGroundAxisAutoOnTarget**(axis, enabled)		
C#		
void value = [Device].**Move_SetGroundAxisAutoOnTarget**(int axis, bool enabled)		
LabVIEW		
setGroundAxisAutoOnTarget.vi		

setGroundTargetRange

Set the range around the target position in which the auto grounding becomes active.

only in AMC300

Function specific parameters		
In	axis	[0|1|2]
	range	in nm
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.setGroundTargetRange		
params: [axis, range]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setGroundTargetRange**(int deviceHandle, int axis, int range)		
Python		
[dev].move.setGroundTargetRange(axis, range)		
Matlab		
[] = **move_setGroundTargetRange**(axis, range)		
C#		
void value = [Device].**Move_SetGroundTargetRange**(int axis, int range)		
LabVIEW		
setGroundTargetRange.vi		

setNSteps

This function triggers n steps on the selected axis in desired direction. /PRO feature.

Function specific parameters		
In	axis	[0|1|2]
	backward	Selects the desired direction. False triggers a forward step, true a backward step
	step	number of step
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.setNSteps		
params: [axis, backward, step]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setNSteps**(int deviceHandle, int axis, bool backward, int step)		
Python		
[dev].move.setNSteps(axis, backward, step)		
Matlab		
[] = **move_setNSteps**(axis, backward, step)		
C#		
void value = [Device].**Move_SetNSteps**(int axis, bool backward, int step)		
LabVIEW		
setNSteps.vi		

setSingleStep

This function triggers one step on the selected axis in desired direction.

Function specific parameters		
In	axis	[0|1|2]
	backward	Selects the desired direction. False triggers a forward step, true a backward step
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.setSingleStep		
params: [axis, backward]		
Result: [errNo]		
C-DLL call		
int **AMC_move_setSingleStep**(int deviceHandle, int axis, bool backward)		
Python		
[dev].move.setSingleStep(axis, backward)		
Matlab		
[] = **move_setSingleStep**(axis, backward)		
C#		
void value = [Device].**Move_SetSingleStep**(int axis, bool backward)		
LabVIEW		
setSingleStep.vi		

writeNSteps

Sets the number of steps to perform on stepwise movement. /PRO feature.

Function specific parameters		
In	axis	[0|1|2]
	step	number of step
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.move.writeNSteps		
params: [axis, step]		
Result: [errNo]		
C-DLL call		
int **AMC_move_writeNSteps**(int deviceHandle, int axis, int step)		
Python		
[dev].move.writeNSteps(axis, step)		
Matlab		
[] = **move_writeNSteps**(axis, step)		
C#		
void value = [Device].**Move_WriteNSteps**(int axis, int step)		
LabVIEW		
writeNSteps.vi		

3.7 Res

getChainGain

Get chain gain, see setChainGain for parameter description

Function specific parameters		
In	axis	number of axis
Out	errNo	errNo
	gaincoeff	gaincoeff
JSON Method		
method: com.attocube.amc.res.getChainGain		
params: [axis]		
Result: [errNo, gaincoeff]		
C-DLL call		
int **AMC_res_getChainGain**(int deviceHandle, int axis, int* gaincoeff)		
Python		
gaincoeff = **[dev].res.getChainGain**(axis)		
Matlab		
[gaincoeff] = **res_getChainGain**(axis)		
C#		
int value = [Device].**Res_GetChainGain**(int axis)		
LabVIEW		
getChainGain.vi		

getLinearization

Gets wether linearization is enabled or not

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	true when enabled

JSON Method
method: com.attocube.amc.res.getLinearization
params: [axis]
Result: [errNo, enabled]
C-DLL call
int **AMC_res_getLinearization**(int deviceHandle, int axis, bool* enabled)
Python
enabled = **[dev].res.getLinearization**(axis)
Matlab
[enabled] = **res_getLinearization**(axis)
C#
bool value = [Device].**Res_GetLinearization**(int axis)
LabVIEW
getLinearization.vi

getLutSn

get the identifier of the loaded lookuptable (will be empty if disabled)

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	value_string1	string : identifier
JSON Method		
method: com.attocube.amc.res.getLutSn		
params: [axis]		
Result: [errNo, value_string1]		
C-DLL call		
int **AMC_res_getLutSn**(int deviceHandle, int axis, char* value_string1, int size0)		
Python		
value_string1 = **[dev].res.getLutSn**(axis)		
Matlab		
[value_string1] = **res_getLutSn**(axis)		
C#		
string value = [Device].**Res_GetLutSn**(int axis)		
LabVIEW		
getLutSn.vi		

getMode

Get mode of RES application, see setMode for the description of possible parameters

Function specific parameters		
Out	errNo	errNo
	mode	mode
JSON Method		
method: com.attocube.amc.res.getMode		
params: []		
Result: [errNo, mode]		
C-DLL call		
int **AMC_res_getMode**(int deviceHandle, int* mode)		
Python		
mode = **[dev].res.getMode**()		
Matlab		
[mode] = **res_getMode**()		
C#		
int value = [Device].**Res_GetMode**()		
LabVIEW		
getMode.vi		

getSensorStatus

Gets wether a valid RES position signal is present (always true for a disabled sensor and for rotators)

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	present	true when present
JSON Method		
method: com.attocube.amc.res.getSensorStatus		
params: [axis]		
Result: [errNo, present]		
C-DLL call		
int **AMC_res_getSensorStatus**(int deviceHandle, int axis, bool* present)		
Python		
present = **[dev].res.getSensorStatus**(axis)		
Matlab		
[present] = **res_getSensorStatus**(axis)		
C#		
bool value = [Device].**Res_GetSensorStatus**(int axis)		
LabVIEW		
getSensorStatus.vi		

setChainGain

Set signal chain gain to control overall power

Function specific parameters		
In	axis	number of axis
	gainconfig	0: 0dB (power 600mVpkpk^2/R), 1 : -10 dB , 2 : -15 dB , 3 : -20 dB
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.res.setChainGain		
params: [axis, gainconfig]		
Result: [errNo]		
C-DLL call		
int **AMC_res_setChainGain**(int deviceHandle, int axis, int gainconfig)		
Python		
[dev].res.setChainGain(axis, gainconfig)		
Matlab		
[] = **res_setChainGain**(axis, gainconfig)		
C#		
void value = [Device].**Res_SetChainGain**(int axis, int gainconfig)		
LabVIEW		
setChainGain.vi		

setConfigurationFile

Load configuration file which either contains a JSON dict with parameters for the positioner on the axis or the LUT file itself (as legacy support for ANC350 .aps files)

Function specific parameters		
In	axis	[0|1|2]
In	content	JSON Dictionary or .aps File. The JSON Dictonary can/must contain the following keys: 'type': mandatory This field has to be one of the positioner list (see getPositionersList) 'lut': optional, contains an array of 1024 LUT values that are a mapping between ratio of the RES element travelled (0 to 1) and the corresponding absolute value at this ratio given in [nm]. Note: when generating these tables with position data in absolute units, the scaling of the travel ratio with the current sensor range has to be reversed. 'lut_sn': optional, a string to uniquely identify the loaded LUT
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.res.setConfigurationFile		
params: [axis, content]		
Result: [errNo]		
C-DLL call		
int **AMC_res_setConfigurationFile**(int deviceHandle, int axis, const char* content)		
Python		
[dev].res.setConfigurationFile(axis, content)		
Matlab		
[] = **res_setConfigurationFile**(axis, content)		
C#		
void value = [Device].**Res_SetConfigurationFile**(int axis, string content)		
LabVIEW		
setConfigurationFile.vi		

setLinearization

Control if linearization is enabled or not

Function specific parameters		
In	axis	[0|1|2]
	enable	boolean (true: enable linearization)
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.res.setLinearization		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_res_setLinearization**(int deviceHandle, int axis, bool enable)		
Python		
[dev].res.setLinearization(axis, enable)		
Matlab		
[] = **res_setLinearization**(axis, enable)		
C#		
void value = [Device].**Res_SetLinearization**(int axis, bool enable)		
LabVIEW		
setLinearization.vi		

setMode

Sets the mode of the RES position measurement

This selects which frequency/ies are used for the lock-in measurement of the RES position, currently there are two possibilities:

1: Individual per axis: each axis is measured on a different frequency; this mode reduces noise coupling between axes, while requiring more wiring

2: Shared line/MIC-Mode: each axis is measured on the same frequency, which reduces the number of required wires while more coupling noise is excpected

3: Same as 1, but with overall lower frequencies. This reduces the influence on the measurement accuracy on longer cables

4: Same as 2, but with overall lower frequencies. This reduces the influence on the measurement accuracy on longer cables

Function specific parameters		
In	mode	1: Individual per axis, 2: Shared line mode, 3: Individual per axis (low frequency), 4: Shared line mode (low frequency)
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.res.setMode		
params: [mode]		
Result: [errNo]		
C-DLL call		
int **AMC_res_setMode**(int deviceHandle, int mode)		
Python		
[dev].res.setMode(mode)		
Matlab		
[] = **res_setMode**(mode)		
C#		
void value = [Device].**Res_SetMode**(int mode)		
LabVIEW		
setMode.vi		

3.8 Rotcomp

getControlTargetRanges

Checks if all three axis are in target range.

Function specific parameters		
Out	errNo	int32 Error code, if there was an error, otherwise 0 for ok
	in_target_range	boolean true all three axes are in target range, false at least one axis is not in target range
JSON Method		
method: com.attocube.amc.rotcomp.getControlTargetRanges		
params: []		
Result: [errNo, in_target_range]		
C-DLL call		
int **AMC_rotcomp_getControlTargetRanges**(int deviceHandle, bool* in_target_range)		
Python		
in_target_range = **[dev].rotcomp.getControlTargetRanges**()		
Matlab		
[in_target_range] = **rotcomp_getControlTargetRanges**()		
C#		
bool value = [Device].**Rotcomp_GetControlTargetRanges**()		
LabVIEW		
getControlTargetRanges.vi		

getEnabled

Gets the enabled status of the rotation compensation

Function specific parameters		
Out	errNo	int32 Error code, if there was an error, otherwise 0 for ok
	enabled	boolean true Rotation compensation is enabled, false Rotation compensation is disabled
JSON Method		
method: com.attocube.amc.rotcomp.getEnabled		
params: []		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_rotcomp_getEnabled**(int deviceHandle, bool* enabled)		
Python		
enabled = **[dev].rotcomp.getEnabled**()		
Matlab		
[enabled] = **rotcomp_getEnabled**()		
C#		
bool value = [Device].**Rotcomp_GetEnabled**()		
LabVIEW		
getEnabled.vi		

getLUT

Gets the LUT file as JSON string

Function specific parameters		
Out	errNo	int32 Error code, if there was an error, otherwise 0 for ok
	lut	string JSON string of the LUT file for the rotation compensation
JSON Method		
method: com.attocube.amc.rotcomp.getLUT		
params: []		
Result: [errNo, lut]		
C-DLL call		
int **AMC_rotcomp_getLUT**(int deviceHandle, char* lut, int size0)		
Python		
lut = **[dev].rotcomp.getLUT**()		
Matlab		
[lut] = **rotcomp_getLUT**()		
C#		
string value = [Device].**Rotcomp_GetLUT**()		
LabVIEW		
getLUT.vi		

setEnabled

Enables and disables the rotation compensation

Function specific parameters		
In	enabled	true Rotation compensation is enabled, false Rotation compensation is disabled
Out	errNo	int32 Error code, if there was an error, otherwise 0 for ok
JSON Method		
method: com.attocube.amc.rotcomp.setEnabled		
params: [enabled]		
Result: [errNo]		
C-DLL call		
int **AMC_rotcomp_setEnabled**(int deviceHandle, bool enabled)		
Python		
[dev].rotcomp.setEnabled(enabled)		
Matlab		
[] = **rotcomp_setEnabled**(enabled)		
C#		
void value = [Device].**Rotcomp_SetEnabled**(bool enabled)		
LabVIEW		
setEnabled.vi		

setLUT

Sets the LUT file from a JSON string

Function specific parameters		
In	lut_string	JSON string of the LUT file for the rotation compensation
Out	errNo	int32 Error code, if there was an error, otherwise 0 for ok
JSON Method		
method: com.attocube.amc.rotcomp.setLUT		
params: [lut_string]		
Result: [errNo]		
C-DLL call		
int **AMC_rotcomp_setLUT**(int deviceHandle, const char* lut_string)		
Python		
[dev].rotcomp.setLUT(lut_string)		
Matlab		
[] = **rotcomp_setLUT**(lut_string)		
C#		
void value = [Device].**Rotcomp_SetLUT**(string lut_string)		
LabVIEW		
setLUT.vi		

updateOffsets

Updates the start offsets of the axes

Function specific parameters		
In	offset_axis0	Offset of axis 1 in [nm]
	offset_axis1	Offset of axis 2 in [nm]
	offset_axis2	Offset of axis 3 in [nm]
Out	errNo	int32 Error code, if there was an error, otherwise 0 for ok
JSON Method		
method: com.attocube.amc.rotcomp.updateOffsets		
params: [offset_axis0, offset_axis1, offset_axis2]		
Result: [errNo]		
C-DLL call		
int **AMC_rotcomp_updateOffsets**(int deviceHandle, int offset_axis0, int offset_axis1, int offset_axis2)		
Python		
[dev].rotcomp.updateOffsets(offset_axis0, offset_axis1, offset_axis2)		
Matlab		
[] = **rotcomp_updateOffsets**(offset_axis0, offset_axis1, offset_axis2)		
C#		
void value = [Device].**Rotcomp_UpdateOffsets**(int offset_axis0, int offset_axis1, int offset_axis2)		
LabVIEW		
updateOffsets.vi		

3.9 Rtin

apply

Apply all realtime input function

Function specific parameters		
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.apply		
params: []		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_apply**(int deviceHandle)		
Python		
[dev].rtin.apply()		
Matlab		
[] = **rtin_apply**()		
C#		
void value = [Device].**Rtin_Apply**()		
LabVIEW		
rtin_apply.vi		

discard

Discard all values beting set and not yet applieds

Function specific parameters		
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.discard		
params: []		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_discard**(int deviceHandle)		
Python		
[dev].rtin.discard()		
Matlab		
[] = **rtin_discard**()		
C#		
void value = [Device].**Rtin_Discard**()		
LabVIEW		
rtin_discard.vi		

getControlAQuadBInResolution

This function gets the AQuadB input resolution for setpoint parameter.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	resolution	in nm
JSON Method		
method: com.attocube.amc.rtin.getControlAQuadBInResolution		
params: [axis]		
Result: [errNo, resolution]		
C-DLL call		
int **AMC_rtin_getControlAQuadBInResolution**(int deviceHandle, int axis, int* resolution)		
Python		
resolution = **[dev].rtin.getControlAQuadBInResolution**(axis)		
Matlab		
[resolution] = **rtin_getControlAQuadBInResolution**(axis)		
C#		
int value = [Device].**Rtin_GetControlAQuadBInResolution**(int axis)		
LabVIEW		
rtin_getControlAQuadBInResolution.vi		

getControlMoveGPIO

This function gets the status for real time input on the selected axis in closed-loop mode.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enable	boolean true: approach enabled, false: approach disabled
JSON Method		
method: com.attocube.amc.rtin.getControlMoveGPIO		
params: [axis]		
Result: [errNo, enable]		
C-DLL call		
int **AMC_rtin_getControlMoveGPIO**(int deviceHandle, int axis, bool* enable)		
Python		
enable = **[dev].rtin.getControlMoveGPIO**(axis)		
Matlab		
[enable] = **rtin_getControlMoveGPIO**(axis)		
C#		
bool value = [Device].**Rtin_GetControlMoveGPIO**(int axis)		
LabVIEW		
rtin_getControlMoveGPIO.vi		

getGpioMode

get the GPIO mode for Mic Mode feature

Function specific parameters		
Out	errNo	errNo
	gpio_mode	gpio_mode: 0: Standard GPIO 1: NSL-/Mic-Mode
JSON Method		
method: com.attocube.amc.rtin.getGpioMode		
params: []		
Result: [errNo, gpio_mode]		
C-DLL call		
int **AMC_rtin_getGpioMode**(int deviceHandle, int* gpio_mode)		
Python		
gpio_mode = **[dev].rtin.getGpioMode**()		
Matlab		
[gpio_mode] = **rtin_getGpioMode**()		
C#		
int value = [Device].**Rtin_GetGpioMode**()		
LabVIEW		
rtin_getGpioMode.vi		

getNslMux

get the axis the NSL multiplexer is set to

Function specific parameters		
Out	errNo	errNo
	mux_mode	0: Off 1: Axis 1 2: Axis 2 3: Axis 3

JSON Method
method: com.attocube.amc.rtin.getNslMux
params: []
Result: [errNo, mux_mode]
C-DLL call
int **AMC_rtin_getNslMux**(int deviceHandle, int* mux_mode)
Python
mux_mode = **[dev].rtin.getNslMux**()
Matlab
[mux_mode] = **rtin_getNslMux**()
C#
int value = [Device].**Rtin_GetNslMux**()
LabVIEW
rtin_getNslMux.vi

getRealTimeInChangePerPulse

This function gets the change per pulse for the selected axis under real time input in the closed-loop mode.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	resolution	to be added in current pos in nm
JSON Method		
method: com.attocube.amc.rtin.getRealTimeInChangePerPulse		
params: [axis]		
Result: [errNo, resolution]		
C-DLL call		
int **AMC_rtin_getRealTimeInChangePerPulse**(int deviceHandle, int axis, int* resolution)		
Python		
resolution = **[dev].rtin.getRealTimeInChangePerPulse**(axis)		
Matlab		
[resolution] = **rtin_getRealTimeInChangePerPulse**(axis)		
C#		
int value = [Device].**Rtin_GetRealTimeInChangePerPulse**(int axis)		
LabVIEW		
rtin_getRealTimeInChangePerPulse.vi		

getRealTimeInFeedbackLoopMode

Get if the realtime function must operate in close loop operation or open loop operation

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	mode	0: open loop, 1 : close-loop
JSON Method		
method: com.attocube.amc.rtin.getRealTimeInFeedbackLoopMode		
params: [axis]		
Result: [errNo, mode]		
C-DLL call		
int **AMC_rtin_getRealTimeInFeedbackLoopMode**(int deviceHandle, int axis, int* mode)		
Python		
mode = **[dev].rtin.getRealTimeInFeedbackLoopMode**(axis)		
Matlab		
[mode] = **rtin_getRealTimeInFeedbackLoopMode**(axis)		
C#		
int value = [Device].**Rtin_GetRealTimeInFeedbackLoopMode**(int axis)		
LabVIEW		
rtin_getRealTimeInFeedbackLoopMode.vi		

getRealTimeInMode

This function sets or gets the real time input mode for the selected axis.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	RT_IN_MODE	Please refer to "setRealTimeInMode" for information on possible values.
JSON Method		
method: com.attocube.amc.rtin.getRealTimeInMode		
params: [axis]		
Result: [errNo, RT_IN_MODE]		
C-DLL call		
int **AMC_rtin_getRealTimeInMode**(int deviceHandle, int axis, int* RT_IN_MODE)		
Python		
RT_IN_MODE = **[dev].rtin.getRealTimeInMode**(axis)		
Matlab		
[RT_IN_MODE] = **rtin_getRealTimeInMode**(axis)		
C#		
int value = [Device].**Rtin_GetRealTimeInMode**(int axis)		
LabVIEW		
rtin_getRealTimeInMode.vi		

getRealTimeInStepsPerPulse

Get the change in step per pulse of the realtime input when trigger and stepper mode is used.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	steps	number of steps to applied
JSON Method		
method: com.attocube.amc.rtin.getRealTimeInStepsPerPulse		
params: [axis]		
Result: [errNo, steps]		
C-DLL call		
int **AMC_rtin_getRealTimeInStepsPerPulse**(int deviceHandle, int axis, int* steps)		
Python		
steps = **[dev].rtin.getRealTimeInStepsPerPulse**(axis)		
Matlab		
[steps] = **rtin_getRealTimeInStepsPerPulse**(axis)		
C#		
int value = [Device].**Rtin_GetRealTimeInStepsPerPulse**(int axis)		
LabVIEW		
rtin_getRealTimeInStepsPerPulse.vi		

setControlAQuadBInResolution

This function sets the AQuadB input resolution for setpoint parameter.

Function specific parameters		
In	axis	[0|1|2]
	resolution	in nm
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setControlAQuadBInResolution		
params: [axis, resolution]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setControlAQuadBInResolution**(int deviceHandle, int axis, int resolution)		
Python		
[dev].rtin.setControlAQuadBInResolution(axis, resolution)		
Matlab		
[] = **rtin_setControlAQuadBInResolution**(axis, resolution)		
C#		
void value = [Device].**Rtin_SetControlAQuadBInResolution**(int axis, int resolution)		
LabVIEW		
rtin_setControlAQuadBInResolution.vi		

setControlMoveGPIO

This function sets the status for real time input on the selected axis in closed-loop mode.

Function specific parameters		
In	axis	[0|1|2]
	enable	boolean true: eanble the approach, false: disable the approach
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setControlMoveGPIO		
params: [axis, enable]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setControlMoveGPIO**(int deviceHandle, int axis, bool enable)		
Python		
[dev].rtin.setControlMoveGPIO(axis, enable)		
Matlab		
[] = **rtin_setControlMoveGPIO**(axis, enable)		
C#		
void value = [Device].**Rtin_SetControlMoveGPIO**(int axis, bool enable)		
LabVIEW		
rtin_setControlMoveGPIO.vi		

setGpioMode

set the GPIO mode for Mic Mode feature

Function specific parameters		
In	gpio_mode	[0|1] 0: Standard GPIO 1: NSL-/Mic-Mode
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setGpioMode		
params: [gpio_mode]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setGpioMode**(int deviceHandle, int gpio_mode)		
Python		
[dev].rtin.setGpioMode(gpio_mode)		
Matlab		
[] = **rtin_setGpioMode**(gpio_mode)		
C#		
void value = [Device].**Rtin_SetGpioMode**(int gpio_mode)		
LabVIEW		
rtin_setGpioMode.vi		

setNslMux

set the axis the NSL multiplexer is set to

Function specific parameters		
In	mux_mode	[0|1|2|3] 0: Off 1: Axis 1 2: Axis 2 3: Axis 3
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setNslMux		
params: [mux_mode]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setNslMux**(int deviceHandle, int mux_mode)		
Python		
[dev].rtin.setNslMux(mux_mode)		
Matlab		
[] = **rtin_setNslMux**(mux_mode)		
C#		
void value = [Device].**Rtin_SetNslMux**(int mux_mode)		
LabVIEW		
rtin_setNslMux.vi		

setRealTimeInChangePerPulse

This function sets the change per pulse for the selected axis under real time input in the closed-loop mode. Only used in closed loop operation.

Function specific parameters		
In	axis	[0|1|2]
	delta	to be added to current position in nm
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setRealTimeInChangePerPulse		
params: [axis, delta]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setRealTimeInChangePerPulse**(int deviceHandle, int axis, int delta)		
Python		
[dev].rtin.setRealTimeInChangePerPulse(axis, delta)		
Matlab		
[] = **rtin_setRealTimeInChangePerPulse**(axis, delta)		
C#		
void value = [Device].**Rtin_SetRealTimeInChangePerPulse**(int axis, int delta)		
LabVIEW		
rtin_setRealTimeInChangePerPulse.vi		

setRealTimeInFeedbackLoopMode

Set if the realtime function must operate in close loop operation or open loop operation

Function specific parameters		
In	axis	[0|1|2]
	mode	0: open loop, 1 : close-loop
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setRealTimeInFeedbackLoopMode		
params: [axis, mode]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setRealTimeInFeedbackLoopMode**(int deviceHandle, int axis, int mode)		
Python		
[dev].rtin.setRealTimeInFeedbackLoopMode(axis, mode)		
Matlab		
[] = **rtin_setRealTimeInFeedbackLoopMode**(axis, mode)		
C#		
void value = [Device].**Rtin_SetRealTimeInFeedbackLoopMode**(int axis, int mode)		
LabVIEW		
rtin_setRealTimeInFeedbackLoopMode.vi		

setRealTimeInMode

This function sets the real time input mode for the selected axis.

Function specific parameters		
In	axis	[0|1|2]
	RT_IN_MODE	0: AquadB (LVTTL) 1: AquadB (LVDS) 8: Stepper (LVTTL) 9: Stepper (LVDS) 10: Trigger (LVTTL) 11: Trigger (LVDS) 15: Off
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setRealTimeInMode		
params: [axis, RT_IN_MODE:]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setRealTimeInMode**(int deviceHandle, int axis, int RT_IN_MODE:)		
Python		
[dev].rtin.setRealTimeInMode(axis, RT_IN_MODE:)		
Matlab		
[] = **rtin_setRealTimeInMode**(axis, RT_IN_MODE:)		
C#		
void value = [Device].**Rtin_SetRealTimeInMode**(int axis, int RT_IN_MODE:)		
LabVIEW		
rtin_setRealTimeInMode.vi		

setRealTimeInStepsPerPulse

Set the change in step per pulse of the realtime input when trigger and stepper mode is used. Only used in open loop operation.

Function specific parameters		
In	axis	[0|1|2]
	steps	number of steps to applied
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtin.setRealTimeInStepsPerPulse		
params: [axis, steps]		
Result: [errNo]		
C-DLL call		
int **AMC_rtin_setRealTimeInStepsPerPulse**(int deviceHandle, int axis, int steps)		
Python		
[dev].rtin.setRealTimeInStepsPerPulse(axis, steps)		
Matlab		
[] = **rtin_setRealTimeInStepsPerPulse**(axis, steps)		
C#		
void value = [Device].**Rtin_SetRealTimeInStepsPerPulse**(int axis, int steps)		
LabVIEW		
rtin_setRealTimeInStepsPerPulse.vi		

3.10 Rtout

apply

Apply for all rtout function

Function specific parameters		
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.apply		
params: []		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_apply**(int deviceHandle)		
Python		
[dev].rtout.apply()		
Matlab		
[] = **rtout_apply**()		
C#		
void value = [Device].**Rtout_Apply**()		
LabVIEW		
rtout_apply.vi		

applyAxis

Apply for rtout function of specific axis

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.applyAxis		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_applyAxis**(int deviceHandle, int axis)		
Python		
[dev].rtout.applyAxis(axis)		
Matlab		
[] = **rtout_applyAxis**(axis)		
C#		
void value = [Device].**Rtout_ApplyAxis**(int axis)		
LabVIEW		
rtout_applyAxis.vi		

discard

Discard all rtout value set by the set function(not applied yet)

Function specific parameters		
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.discard		
params: []		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_discard**(int deviceHandle)		
Python		
[dev].rtout.discard()		
Matlab		
[] = **rtout_discard**()		
C#		
void value = [Device].**Rtout_Discard**()		
LabVIEW		
rtout_discard.vi		

discardAxis

Discard rtout value of specific axis set by the set function(not applied yet)

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.discardAxis		
params: [axis]		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_discardAxis**(int deviceHandle, int axis)		
Python		
[dev].rtout.discardAxis(axis)		
Matlab		
[] = **rtout_discardAxis**(axis)		
C#		
void value = [Device].**Rtout_DiscardAxis**(int axis)		
LabVIEW		
rtout_discardAxis.vi		

discardSignalMode

Discard value set by setSignalMode

Function specific parameters		
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.discardSignalMode		
params: []		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_discardSignalMode**(int deviceHandle)		
Python		
[dev].rtout.discardSignalMode()		
Matlab		
[] = **rtout_discardSignalMode**()		
C#		
void value = [Device].**Rtout_DiscardSignalMode**()		
LabVIEW		
rtout_discardSignalMode.vi		

getControlAQuadBOut

This function gets if of AQuadB output for position indication is enabled

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	enabled	boolean
JSON Method		
method: com.attocube.amc.rtout.getControlAQuadBOut		
params: [axis]		
Result: [errNo, enabled]		
C-DLL call		
int **AMC_rtout_getControlAQuadBOut**(int deviceHandle, int axis, bool* enabled)		
Python		
enabled = **[dev].rtout.getControlAQuadBOut**(axis)		
Matlab		
[enabled] = **rtout_getControlAQuadBOut**(axis)		
C#		
bool value = [Device].**Rtout_GetControlAQuadBOut**(int axis)		
LabVIEW		
rtout_getControlAQuadBOut.vi		

getControlAQuadBOutClock

This function gets the clock for AQuadB output.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	clock_in_ns	Clock in multiples of 20ns. Minimum 2 (40ns), maximum 65535 (1,310700ms)
JSON Method		
method: com.attocube.amc.rtout.getControlAQuadBOutClock		
params: [axis]		
Result: [errNo, clock_in_ns]		
C-DLL call		
int **AMC_rtout_getControlAQuadBOutClock**(int deviceHandle, int axis, int* clock_in_ns)		
Python		
clock_in_ns = **[dev].rtout.getControlAQuadBOutClock**(axis)		
Matlab		
[clock_in_ns] = **rtout_getControlAQuadBOutClock**(axis)		
C#		
int value = [Device].**Rtout_GetControlAQuadBOutClock**(int axis)		
LabVIEW		
rtout_getControlAQuadBOutClock.vi		

getControlAQuadBOutResolution

This function gets the AQuadB output resolution for position indication.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	resolution	in nm
JSON Method		
method: com.attocube.amc.rtout.getControlAQuadBOutResolution		
params: [axis]		
Result: [errNo, resolution]		
C-DLL call		
int **AMC_rtout_getControlAQuadBOutResolution**(int deviceHandle, int axis, int* resolution)		
Python		
resolution = **[dev].rtout.getControlAQuadBOutResolution**(axis)		
Matlab		
[resolution] = **rtout_getControlAQuadBOutResolution**(axis)		
C#		
int value = [Device].**Rtout_GetControlAQuadBOutResolution**(int axis)		
LabVIEW		
rtout_getControlAQuadBOutResolution.vi		

getMode

Get Mode

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	RT_OUT_MODE	For the meaning of the values, please refer to the setMode-function.
JSON Method		
method: com.attocube.amc.rtout.getMode		
params: [axis]		
Result: [errNo, RT_OUT_MODE]		
C-DLL call		
int **AMC_rtout_getMode**(int deviceHandle, int axis, int* RT_OUT_MODE)		
Python		
RT_OUT_MODE = **[dev].rtout.getMode**(axis)		
Matlab		
[RT_OUT_MODE] = **rtout_getMode**(axis)		
C#		
int value = [Device].**Rtout_GetMode**(int axis)		
LabVIEW		
rtout_getMode.vi		

getSignalMode

This function gets the real time output mode for the selected axis.

Function specific parameters		
Out	errNo	errNo
	mode	0: TTL, 1: LVDS
JSON Method		
method: com.attocube.amc.rtout.getSignalMode		
params: []		
Result: [errNo, mode]		
C-DLL call		
int **AMC_rtout_getSignalMode**(int deviceHandle, int* mode)		
Python		
mode = **[dev].rtout.getSignalMode**()		
Matlab		
[mode] = **rtout_getSignalMode**()		
C#		
int value = [Device].**Rtout_GetSignalMode**()		
LabVIEW		
rtout_getSignalMode.vi		

getTriggerConfig

Get the real time output trigger config

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	higher	upper limit in nm / µdeg
	lower	lower limit in nm / µdeg
	epsilon	hysteresis in nm / µdeg
	polarity	0: active high, 1: active low
JSON Method		
method: com.attocube.amc.rtout.getTriggerConfig		
params: [axis]		
Result: [errNo, higher, lower, epsilon, polarity]		
C-DLL call		
int **AMC_rtout_getTriggerConfig**(int deviceHandle, int axis, int* higher, int* lower, int* epsilon, int* polarity)		
Python		
higher, lower, epsilon, polarity = **[dev].rtout.getTriggerConfig**(axis)		
Matlab		
[higher, lower, epsilon, polarity] = **rtout_getTriggerConfig**(axis)		
C#		
Tuple<int,int,int,int> value = [Device].**Rtout_GetTriggerConfig**(int axis)		
LabVIEW		
rtout_getTriggerConfig.vi		

setControlAQuadBOutClock

This function sets the clock for AQuadB output.

Function specific parameters		
In	axis	[0|1|2]
	clock	Clock in multiples of 20ns. Minimum 2 (40ns), maximum 65535 (1,310700ms)
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.setControlAQuadBOutClock		
params: [axis, clock]		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_setControlAQuadBOutClock**(int deviceHandle, int axis, int clock)		
Python		
[dev].rtout.setControlAQuadBOutClock(axis, clock)		
Matlab		
[] = **rtout_setControlAQuadBOutClock**(axis, clock)		
C#		
void value = [Device].**Rtout_SetControlAQuadBOutClock**(int axis, int clock)		
LabVIEW		
rtout_setControlAQuadBOutClock.vi		

setControlAQuadBOutResolution

This function sets the AQuadB output resolution for position indication.

Function specific parameters		
In	axis	[0|1|2]
	resolution	in nm; range [1 ... 16777]
Out	errNo	errNo

JSON Method
method: com.attocube.amc.rtout.setControlAQuadBOutResolution
params: [axis, resolution]
Result: [errNo]
C-DLL call
int **AMC_rtout_setControlAQuadBOutResolution**(int deviceHandle, int axis, int resolution)
Python
[dev].rtout.setControlAQuadBOutResolution(axis, resolution)
Matlab
[] = **rtout_setControlAQuadBOutResolution**(axis, resolution)
C#
void value = [Device].**Rtout_SetControlAQuadBOutResolution**(int axis, int resolution)
LabVIEW
rtout_setControlAQuadBOutResolution.vi

setMode

Set the real time output signal mode. To set either LVTTL or LVDS, please use setSignalMode-function.

Function specific parameters		
In	axis	[0|1|2]
	RT_OUT_MODE	0: Off, 1: AquadB, 2: Trigger
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.setMode		
params: [axis, RT_OUT_MODE]		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_setMode**(int deviceHandle, int axis, int RT_OUT_MODE)		
Python		
[dev].rtout.setMode(axis, RT_OUT_MODE)		
Matlab		
[] = **rtout_setMode**(axis, RT_OUT_MODE)		
C#		
void value = [Device].**Rtout_SetMode**(int axis, int RT_OUT_MODE)		
LabVIEW		
rtout_setMode.vi		

setSignalMode

This function sets the real time output mode for the selected axis.

Function specific parameters		
In	mode	0: TTL, 1: LVDS
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.setSignalMode		
params: [mode]		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_setSignalMode**(int deviceHandle, int mode)		
Python		
[dev].rtout.setSignalMode(mode)		
Matlab		
[] = **rtout_setSignalMode**(mode)		
C#		
void value = [Device].**Rtout_SetSignalMode**(int mode)		
LabVIEW		
rtout_setSignalMode.vi		

setTriggerConfig

Control the real time output trigger config

Function specific parameters		
In	axis	[0|1|2]
	higher	upper limit in nm / µdeg
	lower	lower limit in nm / µdeg
	epsilon	hysteresis in nm / µdeg
	polarity	0: active high, 1: active low
Out	errNo	errNo
JSON Method		
method: com.attocube.amc.rtout.setTriggerConfig		
params: [axis, higher, lower, epsilon, polarity]		
Result: [errNo]		
C-DLL call		
int **AMC_rtout_setTriggerConfig**(int deviceHandle, int axis, int higher, int lower, int epsilon, int polarity)		
Python		
[dev].rtout.setTriggerConfig(axis, higher, lower, epsilon, polarity)		
Matlab		
[] = **rtout_setTriggerConfig**(axis, higher, lower, epsilon, polarity)		
C#		
void value = [Device].**Rtout_SetTriggerConfig**(int axis, int higher, int lower, int epsilon, int polarity)		
LabVIEW		
rtout_setTriggerConfig.vi		

3.11 Status

getFullCombinedStatus

Get the full combined status of a positioner axis and return the status as a string (to be used in the Webapplication)

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	value_string1	string can be "moving","in target range", "backward limit reached", "forward limit reached", "positioner not connected", "grounded" (only AMC300), "output not enabled"
JSON Method		
method: com.attocube.amc.status.getFullCombinedStatus		
params: [axis]		
Result: [errNo, value_string1]		
C-DLL call		
int **AMC_status_getFullCombinedStatus**(int deviceHandle, int axis, char* value_string1, int size0)		
Python		
value_string1 = **[dev].status.getFullCombinedStatus**(axis)		
Matlab		
[value_string1] = **status_getFullCombinedStatus**(axis)		
C#		
string value = [Device].**Status_GetFullCombinedStatus**(int axis)		
LabVIEW		
getFullCombinedStatus.vi		

getOlStatus

Get the Feedback status of the positioner

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	sensorstatus	as integer 0: NUM Positioner connected 1: OL positioner connected 2: No positioner connected , 3: RES positione connected, 4: Positioner with IDS-CL connected
JSON Method		
method: com.attocube.amc.status.getOlStatus		
params: [axis]		
Result: [errNo, sensorstatus]		
C-DLL call		
int **AMC_status_getOlStatus**(int deviceHandle, int axis, int* sensorstatus)		
Python		
sensorstatus = **[dev].status.getOlStatus**(axis)		
Matlab		
[sensorstatus] = **status_getOlStatus**(axis)		
C#		
int value = [Device].**Status_GetOlStatus**(int axis)		
LabVIEW		
getOlStatus.vi		

getStatusConnected

This function gets information about the connection status of the selected axis' positioner.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	connected	If true, the actor is connected
JSON Method		
method: com.attocube.amc.status.getStatusConnected		
params: [axis]		
Result: [errNo, connected]		
C-DLL call		
int **AMC_status_getStatusConnected**(int deviceHandle, int axis, bool* connected)		
Python		
connected = **[dev].status.getStatusConnected**(axis)		
Matlab		
[connected] = **status_getStatusConnected**(axis)		
C#		
bool value = [Device].**Status_GetStatusConnected**(int axis)		
LabVIEW		
getStatusConnected.vi		

getStatusEot

Retrieves the status of the end of travel (EOT) detection in backward direction or in forward direction.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	detected	true when EoT in either direction was detected
JSON Method		
method: com.attocube.amc.status.getStatusEot		
params: [axis]		
Result: [errNo, detected]		
C-DLL call		
int **AMC_status_getStatusEot**(int deviceHandle, int axis, bool* detected)		
Python		
detected = **[dev].status.getStatusEot**(axis)		
Matlab		
[detected] = **status_getStatusEot**(axis)		
C#		
bool value = [Device].**Status_GetStatusEot**(int axis)		
LabVIEW		
getStatusEot.vi		

getStatusEotBkwd

This function gets the status of the end of travel detection on the selected axis in backward direction.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	detected	true when EoT was detected
JSON Method		
method: com.attocube.amc.status.getStatusEotBkwd		
params: [axis]		
Result: [errNo, detected]		
C-DLL call		
int **AMC_status_getStatusEotBkwd**(int deviceHandle, int axis, bool* detected)		
Python		
detected = **[dev].status.getStatusEotBkwd**(axis)		
Matlab		
[detected] = **status_getStatusEotBkwd**(axis)		
C#		
bool value = [Device].**Status_GetStatusEotBkwd**(int axis)		
LabVIEW		
getStatusEotBkwd.vi		

getStatusEotFwd

This function gets the status of the end of travel detection on the selected axis in forward direction.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	detected	true when EoT was detected
JSON Method		
method: com.attocube.amc.status.getStatusEotFwd		
params: [axis]		
Result: [errNo, detected]		
C-DLL call		
int **AMC_status_getStatusEotFwd**(int deviceHandle, int axis, bool* detected)		
Python		
detected = **[dev].status.getStatusEotFwd**(axis)		
Matlab		
[detected] = **status_getStatusEotFwd**(axis)		
C#		
bool value = [Device].**Status_GetStatusEotFwd**(int axis)		
LabVIEW		
getStatusEotFwd.vi		

getStatusMoving

This function gets information about the status of the stage output.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	status	0: Idle, i.e. within the noise range of the sensor, 1: Moving, i.e the actor is actively driven by the output stage either for closed-loop approach or continous/single stepping and the output is active. 2 : Pending means the output stage is driving but the output is deactivated
JSON Method		
method: com.attocube.amc.status.getStatusMoving		
params: [axis]		
Result: [errNo, status]		
C-DLL call		
int **AMC_status_getStatusMoving**(int deviceHandle, int axis, int* status)		
Python		
status = **[dev].status.getStatusMoving**(axis)		
Matlab		
[status] = **status_getStatusMoving**(axis)		
C#		
int value = [Device].**Status_GetStatusMoving**(int axis)		
LabVIEW		
getStatusMoving.vi		

getStatusReference

This function gets information about the status of the reference position.
It can only be used in conjunction with NUM.

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	valid	true = valid, false = not valid
JSON Method		
method: com.attocube.amc.status.getStatusReference		
params: [axis]		
Result: [errNo, valid]		
C-DLL call		
int **AMC_status_getStatusReference**(int deviceHandle, int axis, bool* valid)		
Python		
valid = **[dev].status.getStatusReference**(axis)		
Matlab		
[valid] = **status_getStatusReference**(axis)		
C#		
bool value = [Device].**Status_GetStatusReference**(int axis)		
LabVIEW		
getStatusReference.vi		

getStatusTargetRange

This function gets information about whether the selected axis' positioner is in target range or not.

The detection only indicates whether the position is within the defined range. This status is updated periodically but currently not in real-time.

If a fast detection is desired, please check the position in a loop

Function specific parameters		
In	axis	[0|1|2]
Out	errNo	errNo
	in_range	true within the target range, false not within the target range
JSON Method		
method: com.attocube.amc.status.getStatusTargetRange		
params: [axis]		
Result: [errNo, in_range]		
C-DLL call		
int **AMC_status_getStatusTargetRange**(int deviceHandle, int axis, bool* in_range)		
Python		
in_range = **[dev].status.getStatusTargetRange**(axis)		
Matlab		
[in_range] = **status_getStatusTargetRange**(axis)		
C#		
bool value = [Device].**Status_GetStatusTargetRange**(int axis)		
LabVIEW		
getStatusTargetRange.vi		

3.12 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

3.13 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		

3.14 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

<table>
<tr><th colspan="3">Function specific parameters</th></tr>
<tr><td rowspan="2">Out</td><td>errNo</td><td>errorCode</td></tr>
<tr><td>Proxy</td><td>Server String, empty for no proxy</td></tr>
<tr><th colspan="3">JSON Method</th></tr>
<tr><td colspan="3">method: com.attocube.system.network.getProxyServer</td></tr>
<tr><td colspan="3">params: []</td></tr>
<tr><td colspan="3">Result: [errNo, Proxy]</td></tr>
<tr><th colspan="3">C-DLL call</th></tr>
<tr><td colspan="3">int system_network_getProxyServer(int deviceHandle, char* Proxy, int size0)</td></tr>
<tr><th colspan="3">Python</th></tr>
<tr><td colspan="3">Proxy = [dev].network.getProxyServer()</td></tr>
<tr><th colspan="3">Matlab</th></tr>
<tr><td colspan="3">[Proxy] = system_network_getProxyServer()</td></tr>
<tr><th colspan="3">C#</th></tr>
<tr><td colspan="3">string value = [Device].Network_GetProxyServer()</td></tr>
<tr><th colspan="3">LabVIEW</th></tr>
<tr><td colspan="3">getProxyServer.vi</td></tr>
</table>

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		

3.15 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		

setTime

Set system time manually

Function specific parameters		
In	day	integer: Day (1-31)
	month	integer: Day (1-12)
	year	integer: Day (eg. 2021)
	hour	integer: Day (0-23)
	minute	integer: Day (0-59)
	second	integer: Day (0-59)
Out	errNo	errorCode
JSON Method		
method: com.attocube.system.setTime		
params: [day, month, year, hour, minute, second]		
Result: [errNo]		
C-DLL call		
int **system_setTime**(int deviceHandle, int day, int month, int year, int hour, int minute, int second)		
Python		
[dev].system_service.setTime(day, month, year, hour, minute, second)		
Matlab		
[] = **system_setTime**(day, month, year, hour, minute, second)		
C#		
void value = [Device].**SetTime**(int day, int month, int year, int hour, int minute, int second)		
LabVIEW		
setTime.vi		

softReset

Performs a soft reset (Reset without deleting the network settings). Please reboot the device directly afterwards.

Function specific parameters		
Out	errNo	errorCode
JSON Method		
method: com.attocube.system.softReset		
params: []		
Result: [errNo]		
C-DLL call		
int **system_softReset**(int deviceHandle)		
Python		
[dev].system_service.softReset()		
Matlab		
[] = **system_softReset**()		
C#		
void value = [Device].**SoftReset**()		
LabVIEW		
softReset.vi		

updateTimeFromInternet

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		

3.16 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