AFM Control API Documentation

API version 1.1

Release along with AFM Control application 2.0.2

Last edited: 04.10.2024

Changelog

Version	Changes
1.1	Added command object “DataSubscription” to manage all data subscriptions
	Added subscription to AFM system log
	Added command object “ActionMeasurementStop”
	Added data format for raw measurement data “float”, and scientific notation for the format “txt”
1.0	First release of the AFM Control API

Table of Contents

Introduction

This document provides the specification for the AFM Control API (Application Programming Interface). The API uses JSON (JavaScript Object Notation) commands that are sent between the AFM control app server and a client over the WebSocket protocol. Each API object allows specific operations, such as setting or getting values. The following sections detail the available API commands, including examples of JSON requests and the expected responses.

The API enables users to control the AFM control application remotely. This is especially useful for those who need to operate the AFM from their own host application, allowing for seamless integration with other equipment and centralized control from the host system.

Functionally, the system reacts the same way whether an operation is performed manually by a user or executed through an API command. This ensures consistent behavior and outcomes, regardless of the method used to interact with the AFM control application.

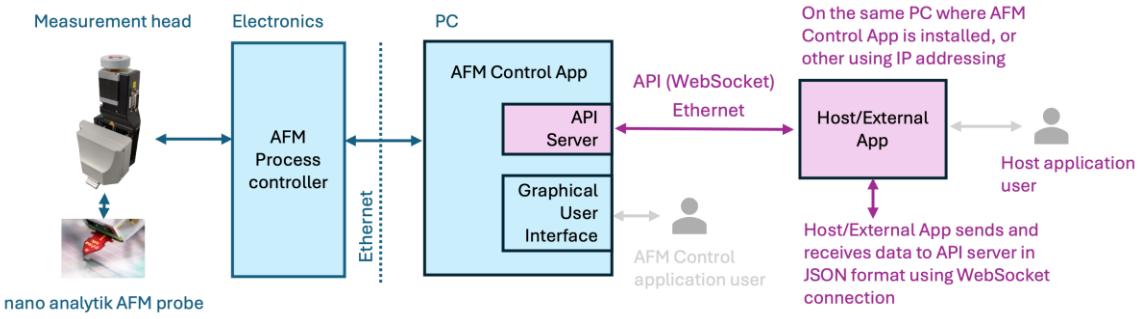

Figure 1: Schematic diagram of the AFM system with API connection

To control the system via the API server using WebSocket, client applications need to establish a WebSocket connection by sending a handshake request to the server. Once connected, clients can send control commands in the form of structured JSON messages, specifying desired actions such as data requests or hardware control instructions. The system will respond with real-time feedback, and clients can subscribe to specific data channels, like status updates or measurement results. Each client can individually manage subscriptions, choosing whether to receive data streams or single updates, allowing for efficient control and data monitoring.

API WebSocket Server

To use the API, the user must first start the server. This can be done either automatically by configuring the INI file or manually via the 'API Connection' option in the toolbar. Begin by entering the IP address of the PC where the AFM control application is installed, followed by the port number the API server will listen on. Press the 'Start Server' button to initiate the server, or press 'Stop Server' to halt it. When the server is stopped, all connected clients will be automatically disconnected.

For the client-server connection on the same machine, use IP address `127.0.0.1`. In case, the server and clients are on separate machines, enter the IP address, which is used for IP communication. If you are unsure what is the IP address of the server computer, open a Windows terminal or command

prompt and type **`ipconfig`**. Look for the IPv4 address of the machine's Ethernet/WiFi interface (e.g. `192.168.x.x`).

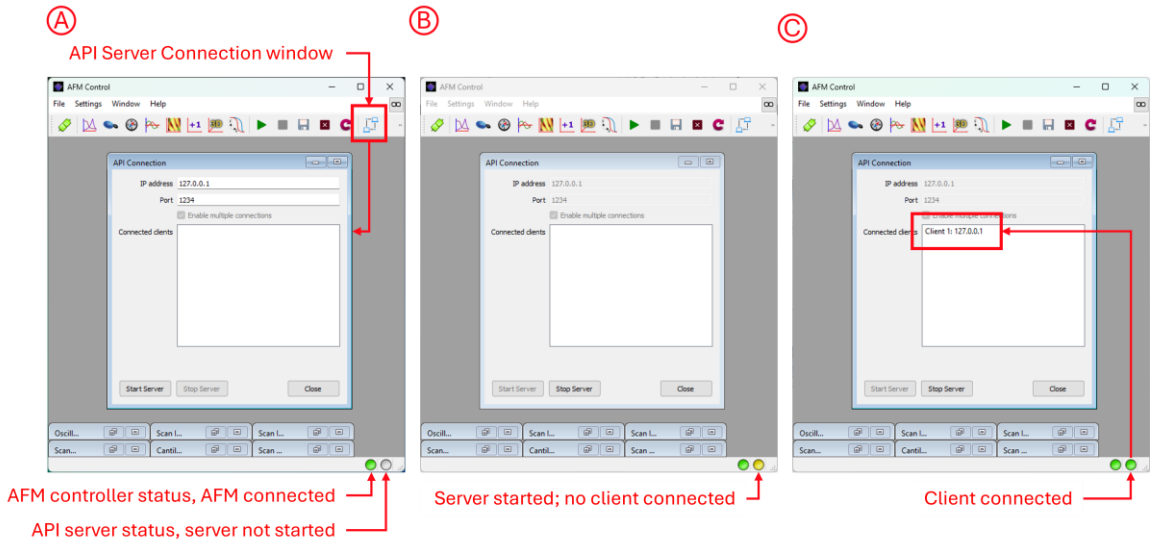

Figure 2: API Connection properties

An indicator displays the status of the API server (see Figure 2). When the server is not started, the API server indicator is gray (Fig. 2A). When the server is started but no authenticated clients are connected, the indicator is yellow (Fig. 2B). If at least one client is connected and authenticated, the indicator turns green (Fig. 2C).

WebSocket Clients

Clients wishing to connect to the API server must provide the correct IP address and Port. After establishing a connection, clients must authenticate the session. The first command sent over the API must be the "authenticate" command (see the appropriate chapter for details). If a valid API-Key is not provided or the first command is not the authentication command, the server will disconnect the client. Authenticated clients will be listed in the "Connected clients" text box.

For localhost in the example above, the API server address shall look as follows `ws://127.0.0.1:1234`.

A convenient way to test the API communication is by using tools like Postman or in JavaScript in a Web browser. Postman allows you to simulate API requests and examine the responses without the need for custom client software. By configuring Postman with the appropriate WebSocket settings, IP address, Port, and authentication commands, users can efficiently validate their API setup and troubleshoot any issues in the communication process (see Figure 3). If you encounter any issues, make sure your antivirus software, VPN, browser or firewall isn't blocking WebSocket connections.

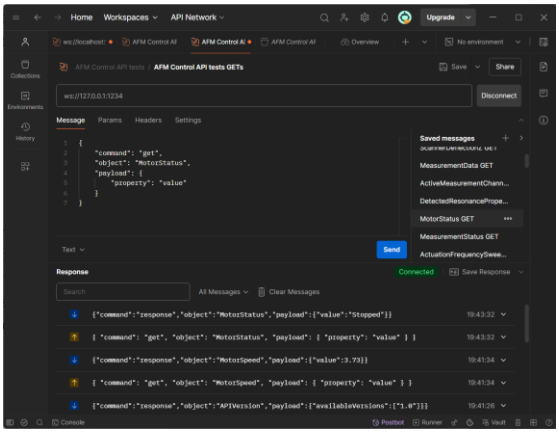

Figure 3: An exemplary Postman session with the AFM Control server below

Basic usage

The API operates based on simple logic: clients send commands in a structured JSON format and server responses accordingly. Each command sent by the client is processed by the API server, which then returns a response message, also in JSON format. The response will either confirm the successful execution of the command, provide the requested data, or include an error message if something went wrong.

A typical client command follows this structure:

```
{
  "command": "get",
  "object": " MotorStatus",
  "payload": {
        "property": "value"
   }
}
```

where **command**: Specifies the type of operation (e.g., "get", "set"), **object**: Identifies the API object to interact with, **payload**: Contains any additional data needed for the command, such as specific properties and their values. See further captions for detailed description.

When the coarse motor is not moving, the API response is:

```
{
  "command": "response",
  "object": " MotorStatus",
  "payload": {
        "value": "Stopped"
   }
}
```

Client subscription mode

The client subscription mechanism in the AFM Control application's API allows clients to receive near-real-time measurement data by subscribing to specific data types and channels. This system is designed to provide a flexible and efficient way for clients to access the measurement data as soon as they are ready.

Overview of the Subscription Mechanism

- **Data Types**: The API supports various types of data (API version 1.1 supports `"line"`, `"map"`, `"log"`)
- **Channels**: Measurement data is organized into channels
- **Subscriptions**: Clients can subscribe to one or more combinations of data types and channels to receive the corresponding measurement data.

Client subscription configuration

To receive measurement data, clients must send a subscription request specifying the data type and channel they wish to subscribe to.

```
Example:
{
    "command": "set",
    "object": "MeasurementDataSubscription",
    "payload": {
        "property": "type",
        "type": "line",
        "format": "txt",
        "channel": 0,
        "subscription": true
    }
}
```

In this case, a client will subscribe to channel 0 (main channel) to receive measurement data taken after each line.

Upon receiving a subscription request, the server:

1. Validates the Request:
2. Updates Subscription Records:
3. Acknowledges the Subscription

To unsubscribe from the given channel and data type, a client must send the the same command as for subscription, but with the parameter `subscription` set to `false` instead of `true.`

To list the valid subscriptions, a client sends a command

```
{
    "command": "get",
    "object": "MeasurementDataSubscription",
    "payload": {
        "property": "value"
    }
}
```

For the exemplary subscription, the server responses

```
{
    "command": "response",
    "object": "MeasurementDataSubscription",
    "payload": {
        "subscriptions": [
```

```
            {
                "channel": 0,
                "type": "line"
            }
        ]
    }
}
```

If a client has more subscriptions, more data entries will be listed under `subscription` property.

After the measurement is started, after completion of subscribed measurements, the server broadcasts the available data to the subscribers as soon as the data is ready. API server supports broadcasting the same data to multiple clients, as well as multiple channels to subscribed clients.

For more information, see the documentation for commands related to the object `DataSubscription` and `MeasurementDataSubscription`.

Automatic start of the WebSocket server

You can configure the AFM Control application to automatically start the API WebSocket server by setting the appropriate options in the application's INI configuration file. This allows the server to begin listening for client connections as soon as the application starts, without manual intervention.

Steps to Configure the INI File

1. **Add or Modify the API WebSocket Server Section**
2. **Ensure that the following section is present in the INI file:**

```
[afmconstants/APIWebSocketServer]
```

If it doesn't exist, add it to the file.

3. **Configure the Server Settings**

Add or modify the following settings under the `[afmconstants/APIWebSocketServer]` section:

```
IP = "127.0.0.1"
Port = 1234
StartServerOnStartUp = 1
```

4. **Save the INI File**
   - After making the changes, save the file and close the text editor.
5. **Restart the AFM Control Application**

- For the new settings to take effect, restart the application if it's already running.
- Upon startup, the application will read the INI file and start the API WebSocket server based on the configured settings.

Setting Descriptions:

- **IP Address (`IP`)**:
  - Specifies the network interface the server will bind to.
  - `"127.0.0.1"` binds the server to localhost (the local machine).
  - In needed, replace with another valid IP address (`"XXX.XXX.XXX.XXX"`) to allow connections from other machines.
- **Port (`Port`)**:
  - The network port the server listens on.
  - Must be a number between 1 and 65535.
  - Ensure the chosen port is not used by another application.
- **Start Server on Startup (`StartServerOnStartUp`)**:
  - Determines whether the server starts automatically when the application launches.
  - `0`: The server will **not** start automatically.
  - `1`: The server **will** start automatically.

Commands list

This section lists all available AFM Control objects and commands supported by API version 1.0 in alphabetical order. Each chapter describes a given object name, a short function description, and provides exemplary API commands and responses. API users are free to compose sequences of commands. It is also recommended to experiment with API server operation using tools like Postman or Python scripts to fine-tune the AFM system to the particular AFM applications.

Please note that only authenticated API clients are allowed to exchange data with the AFM Control application server.

ActionActuationFrequencySweepStart

Description

Begin frequency sweep

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionActuationFrequencySweepStart",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

ActionActuationFrequencySweepStop

Description

End sweep frequency

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionActuationFrequencySweepStop",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

ActionAddMeasurementChannel

Description

Add a measurement channel

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionAddMeasurementChannel",
    "payload": {
        "property": "channel",
        "value": true
    }
}
```

Expected SET Values

true only

ActionMeasurementBufferClear

Description

Clear scanned picture buffer

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionMeasurementBufferClear",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

ActionMeasurementStart

Description

Begin surface scanning

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActionMeasurementStart",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true

Get Example

```
{
    "command": "get",
    "object": "ActionMeasurementStart",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

true, false

Notes

Value "true" in SET command triggers the measurement start. Value in the GET response returns true when system is measuring and false when is in idle mode.

ActionMeasurementStop

Description

Stop surface scanning

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActionMeasurementStop",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true

Get Example

```
{
    "command": "get",
    "object": " ActionMeasurementStop",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

true, false

Notes

Value "true" in SET command triggers the measurement stop. Value in the GET response returns true when system is in idle mode and false when is measuring.

ActionMotorApproachContinuous

Description

Approach Cantilever towards the surface

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionMotorRetractContinuous",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

ActionMotorApproachOnce

Description

Single approach step towards the surface

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionMotorRetractOnce",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Notes

One single step likely will be executed faster than getting the motor status. Therefore "false" response is to be expected in a typical case.

ActionMotorRetractContinuous

Description

Rectract Cantilever from the surface

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActionMotorRetractContinuous",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

Get Example

```
{
    "command": "get",
    "object": "ActionMotorRetractContinuous",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

true, false

ActionMotorRetractOnce

Description

Single retract step from the surface

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionMotorRetractOnce",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

ActionMotorSafeDistance

Description

Go to safe distance

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionMotorSafeDistance",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

ActionRemoveMeasurementChannel

Description

Remove measurement channel

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionRemoveMeasurementChannel",
    "payload": {
        "property": "index",
        "channel": "1"
    }
}
```

Expected SET Values

unsigned int

Notes

Removing the main channel (0) is not permitted.

ActionScannerReset

Description

Reset the scaning settings

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActionScannerReset",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Notes

Strictly technically, it is possible to get the ActionScannerReset value, but the respons gets the status whether the action is in the middle of operation, and not whether the scanner has been reset after the the set command was sent.

ActiveMeasurementChannels

Description

Returns number of active measurement channels

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "ActiveMeasurementChannels",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int

Notes

unsigned int between 1 and 4, which is the maximum number of channels supported for this software version.

ActuationAmplitude

Description

Cantilever actuation amplitude

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActuationAmplitude",
    "payload": {
        "property": "value",
        "value": 0.150
    }
}
```

Expected SET Values

floating-point value

Get Example

```
{
    "command": "get",
    "object": "ActuationAmplitude",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

floating-point value

ActuationFrequency

Description

Cantilever actuation frequency

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActuationFrequency",
    "payload": {
        "property": "value",
        "value": 32733
    }
}
```

Expected SET Values

floating-point value

Get Example

```
{
    "command": "get",
    "object": "ActuationFrequency",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

floating-point value

ActuationFrequencySweepStart

Description

Start sweep frequency value

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActuationFrequencySweepStart",
    "payload": {
        "property": "value",
        "value": 10000
    }
}
```

Expected SET Values

floating-point value

ActuationFrequencySweepStop

Description

End sweep frequency value

Set/Get Information

set

Set Example

```
{
    "command": "set",
    "object": "ActuationFrequencySweepStop",
    "payload": {
        "property": "value",
        "value": 40000
    }
}
```

Expected SET Values

floating-point value

ActuationHalfResonanceFrequency

Description

Half omega actuation of the cantilever resonance frequency

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActuationHalfResonanceFrequency",
    "payload": {
        "property": "state",
        "value": true
    }
}
```

Expected SET Values

true, false

Get Example

```
{
    "command": "get",
    "object": "ActuationHalfResonanceFrequency",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

true, false

ActuationOutput

Description

Switch on/off the actuation signal ouput

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ActuationOutput",
    "payload": {
        "property": "triggered",
        "value": true
    }
}
```

Expected SET Values

true, false

Get Example

```
{
    "command": "get",
    "object": "ActuationOutput",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

true, false

AFMAmplitudeSetPoint

Description

Cantilever amplitude contact amplitude

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "AFMAmplitudeSetPoint",
    "payload": {
        "property": "value",
        "value": 0.521
    }
}
```

Expected SET Values

floating-point value

Get Example

```
{
    "command": "get",
    "object": "AFMAmplitudeSetPoint",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

floating-point value

AFMPIDConstantI

Description

I constant in the Z PID controller

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "AFMPIDConstantI",
    "payload": {
        "property": "value",
        "value": 20
    }
}
```

Expected SET Values

unsigned int

Get Example

```
{
    "command": "get",
    "object": "AFMPIDConstantI",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int

AFMPIDConstantP

Description

P constant in the Z PID controller

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "AFMPIDConstantP",
    "payload": {
        "property": "value",
        "value": 1000
    }
}
```

Expected SET Values

unsigned int

Get Example

```
{
    "command": "get",
    "object": "AFMPIDConstantP",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int

APIVersion

Description

Manage API version

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "APIVersion",
    "payload": {
        "property": "value",
        "value": "1.0"
    }
}
```

Expected SET Values

String values from the list of available server API versions

Get Example

```
{
    "command": "get",
    "object": "APIVersion",
    "payload": {
        "property": "value",
        "value": "current"
    }
}
```

Expected GET Values

String with a API version

Notes

Use "current", "available" for the "value" field to get the corresponding data

Examplary response:

```
{
    "command": "get",
    "object": "APIVersion",
    "payload": {
        "property": "value",
        "value": "current"
    }
}
```

authenticate

Description

Authenticate client for APIcommunication

Set/Get Information

set

Set Example

```
{
    "command": "authenticate",
    "apikey": "d1f89a72-3f0b-4d57-b3a9-0f7c63a2e914"
}
```

Notes

Do not reveal the API-Key to unauthorized people!

DataSubscription

Description

Subscription to data steaming

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "DataSubscription",
    "payload": {
        "property": "type",
        "type": "log",
        "subscription": true
    }
}
```

Expected SET Values

String values supported for property "`type`" are “line” (cross-section for each measured line) ,“map” (whole scanned area maps) or “log” (system log messages).

Get Example

```
{
    "command": "get",
    "object": "DataSubscription",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

String “value” for "property".

Notes

An exemplary response for a client, who subscribed to “line” measurement for channels 0 and 1, and to the system log messages:

```
{
    "command": "response",
    "object": "DataSubscription",
    "payload": {
        "subscriptions": [
            {
                "channel": 0,
                "format": "txt",
                "type": "line"
            },
```

```
            {
                "channel": 1,
                "format": "txt",
                "type": "line"
            },
            {
                "format": "txt",
                "type": "log"
            }

        ]
    }
}
```

`"subscription": true` subscribes to the data subscription system, `"subscription": false` unsubscribes from the data subscription system.

When a clients requests a subscription to the system log, “channel” and “format” fields are ignored.

For more information about the subscription to the measurement data, see MeasurementDataSubscription.

FoundResonanceProperties

Description

Resonance parameters provided by the automatic sweep procedure.

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": " FoundResonanceProperties ",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

JSON structure with cantilever resonance parameters

Notes

When after automatic frequency sweeping, a proper resonance is found, API returns the JSON structure as in this example:

```
{
    "command": "response",
    "object": "FoundResonanceProperties",
    "payload": {
        "Ivalue": 290,
        "LITimeConstant_ms": 5.99708,
        "Pvalue": 10,
        "QFactor": 3926.04,
        "peakBandwidth_Hz": 8.33739,
        "resonanceAmplitude_V": 0.393794,
        "resonanceFrequencyActuation_Hz": 32733.0,
        "resonanceFrequencyVibration_Hz": 16366.5
    }
}
```

When the probe actuation is set to half-Omega (see AFM user manual and `ActuationHalfResonanceFrequency` for reference) the actuation resonance frequency is a half of actual mechanical probe resonance frequency. Based on found probe's resonance properties, AFM Control application proposes initial P ("Pvalue") and I ("Ivalue") constants for the control tip-sample feedback loop. `"LITimeConstant_ms"` indicates proposed Lock-In amplifier time constant for balanced performance in terms of response speed and noise damping.

FrequencySweepStatus

Description

Actuation frequency sweep status

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "FrequencySweepStatus",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

"Sweep", "Idle"

Notes

"Sweep" = actuation frequency sweep in progress, "Idle"= AFM is not sweeping actuation frequency

MeasurementData

Description

Get the scanned data from the buffer

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "MeasurementData",
    "payload": {
        "property": "value",
        "format": "txt",
        "type": "image",
        "channel": "0"
    }
}
```

Expected GET Values

“format”: “txt ” – measurement map stored as ASCII text

“type”: “image” – full measurement data including metadata and measurement map; “metadata” – metadata only; “map” – measurement map only

“channel”: measurement data channel from 0 to MaxAvailableChannel -1

Notes

The measurement data from the most recent completed measurement is transmitted via the API. The data structure consists of metadata, which describes the measurement conditions and settings, and the measurement data itself in an NxN matrix, where NxN represents the measurement resolution in pixels. The data format is very similar to Gwyddion files, with the key difference being that data points are transferred as ASCII values.

Please note that transferring full high-resolution images (above 256x256) may overload the WebSocket connection and cause instabilities in the Client-Server communication. In this case, it is recommended to use the subscription model to transmit measurement data in line by line (see `MeasurementDataSubscription` chapter for more information).

MeasurementDataActiveChannel

Description

Signal to display or transmit

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "MeasurementDataActiveChannel",
    "payload": {
        "property": "index",
        "value": 1,
        "channel": "0"
    }
}
```

Expected SET Values

unsigned int type as "value" in the SET command corresponding with the measurement channel set to extract data from

Get Example

```
{
    "command": "get",
    "object": "MeasurementDataActiveChannel",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int for "index", string for "text"

Notes

Examplary response:

```
{
    "command": "response",
    "object": "MeasurementDataActiveChannel",
    "payload": {
        "value": {
            "index": 0,
            "text": "topography"
        }
    }
}
```

index 0 = "topography", index 1 = "phase"

MeasurementDataCorrectionMode

Description

Line correction method

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "MeasurementDataCorrectionMode",
    "payload": {
        "property": "index",
        "value": 1,
        "channel": "1"
    }
}
```

Expected SET Values

unsigned int type as "value" in the SET command to set the corresponsdix index in the modes collection

Get Example

```
{
    "command": "get",
    "object": "MeasurementDataCorrectionMode",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int for "index", string for "text"

Notes

index 0 = "none", index 1 = "line" correction, index 2 = "Plane", index 3 = "Paraboloid", index 4 = "Cubic surface"

MeasurementDataDirectionMode

Description

Forward or backward signal directon to display or transmit

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "MeasurementDataDirectionMode",
    "payload": {
        "property": "index",
        "value": "0",
        "channel": "1"
    }
}
```

Expected SET Values

unsigned int type as "value" in the SET command corresponding with the measurement channel set to extract data from

Get Example

```
{
    "command": "get",
    "object": "MeasurementDataDirectionMode",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int for "index", string for "text"

Notes

Examplary response:

```
{
    "command": "response",
    "object": "MeasurementDataDirectionMode",
    "payload": {
        "value": {
            "index": 0,
            "text": "forward"
        }
    }
}
```

index 0 = "forward", index 1 = "backward" direction

MeasurementDataSubscription

Description

Subscription to data steaming

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "MeasurementDataSubscription",
    "payload": {
        "property": "type",
        "type": "line",
        "format": "txt",
        "channel": 0,
        "subscription": true
    }
}
```

Expected SET Values

String values supported for property "`type`" are “line” (cross-section for each measured line) and “map” (whole scanned area maps). "`format`" can be “float”, “txt” or “base64” coded in binary base64 format. “`channel`” contains int value between 0 and 3, where “channel” corresponds to the GUI data channel. Channel 0 represents the main AFM measurement channel.
“subscription” is a bool value. `true` means to subscribe, and `false` means to unsubscribe to the data channel.

Get Example

```
{
    "command": "get",
    "object": "MeasurementDataSubscription",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

String “value” for "property".

Notes

An exemplary response for a client, who subscribed to “line” measurement for channels 0 and 1:

```
{
    "command": "response",
    "object": "MeasurementDataSubscription",
```

```
    "payload": {
        "subscriptions": [
            {
                "channel": 0,
                "format": "txt",
                "type": "line"
            },
            {
                "channel": 1,
                "format": "txt",
                "type": "line"
            }
        ]
    }
}
```

Each line data response consists of the JSON structure as follows:

```
{
    "command": "response",
    "object": "MeasurementDataSubscription",
    "payload": {
        "channel": 0,
        "format": "float",
        "signal": "phase",
        "type": "line",
        "value": {
            "x": [
                0,
                0.04093853011727333,
                0.08187706023454666,
...
```

In the "value" structure, there are 3 sets of floating-point data for x (`"x"`) position in µm, measured signals in forward (`"y_forward"`) and backward (`"y_backward"`) direction. The length of each vector corresponds to the `ScannerResolution` setting; when set to 128x128, the data vector is 128 data-point long. Property "`y_position`" show the integer value of pixel position in Y direction. Y position pixel value is between 0 and scan resolution – 1. Due to the limited refresh rate of the scan lines, some measurement lines may be omitted for display, when scanning in high speed > 2 lines/second, but they are recorded in the measurement data.

When `"format"` is set to `"txt"`, values are rounded to 5 digits after coma and sent as strings in scientific notation (e.g. 0.08283889… will be changed to "8.2839e-02"). When `"format"` is set to `"base64"`, values are rounded to 5 digits after coma and sent base64 binary coded.

MeasurementStatus

Description

Get AFM scan state

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "MeasurementStatus",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

Measurement, Idle

Notes

"Measurement" = measurement in progress, "Idle"= AFM is not measuring

MotorApproachMode

Description

Manual or auto approach method

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "MotorApproachMode",
    "payload": {
        "property": "index",
        "value": 0
    }
}
```

Expected SET Values

unsigned int

Get Example

```
{
    "command": "get",
    "object": "MotorApproachMode",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int for "index", string for "text"

Notes

index 0 = "manual" approach, index 1 = "auto, fast" approach

MotorPosition

Description

Return motor position with reliability status

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "MotorPosition",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type in micrometers for "value", string value for “reliability” property (“reliable” or “unreliable”).

Notes

Exemplary response for the position just after starting the application. In this case, similarly to the `ScannerPosition`, AFM system has not reached the tip-sample contact to determine the motor position as reliable.

```
{
    "command": "response",
    "object": "MotorPosition",
    "payload": {
        "reference": "",
        "reliability": "unreliable",
        "value": 0
    }
}
```

MotorSpeed

Description

Motor speed of the Z positioner

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "MotorSpeed",
    "payload": {
        "property": "value",
        "value": 500
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "MotorSpeed",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

Notes

Min value around 3.73 um/s (depends on the motor type), max value 1000 um/s

MotorStatus

Description

Return the motor status

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "MotorStatus",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

"None", "Stopped", "Retracting", "Approaching", "Approached", "Unknown"

Notes

"None" when motor not available, "Stopped" when motor is not moving, "Retracting" when motor is retracting from the surface, "Approaching" when motor is approaching to the surface, "Approached" when the probe is in the contact with the surface, "Unknown" when state cannot be determined

ScannerCenterX

Description

Scan centre in the x scanning direction

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerCenterX",
    "payload": {
        "property": "value",
        "value": 12.3456
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerCenterX",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

ScannerCenterY

Description

Scan centre in the y scanning direction

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerCenterY",
    "payload": {
        "property": "value",
        "value": 7.6543
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerCenterY",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

ScannerDeflectionZ

Description

Z piezo deflection

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "ScannerDeflectionZ",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

Notes

Floating point values between 0 and 100 percent

ScannerLimitZ

Description

Z piezo limit

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerLimitZ",
    "payload": {
        "property": "value",
        "value": 50.21
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerLimitZ",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

Notes

Floating point values between 0 and 100 percent

ScannerLinesPerSecond

Description

Lines per second

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerLinesPerSecond",
    "payload": {
        "property": "value",
        "value": 2.5
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerLinesPerSecond",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

Notes

The higher value, the faster measurement is carried out, but the quality of measurement data may be reduced. See the manual for more information.

ScannerMode

Description

Scan mode

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerMode",
    "payload": {
        "property": "index",
        "value": 1
    }
}
```

Expected SET Values

unsigned int type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerMode",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int for "index", string for "text"

Notes

index 0 = "signle frame", index 1 = "continuous" measurements

ScannerPosition

Description

Return tip-sample distance with reliability status

Set/Get Information

get

Get Example

```
{
    "command": "get",
    "object": "ScannerPosition",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type in micrometers for "value", string value for “reliability” property (“reliable” or “unreliable”).

Notes

Exemplary response for the position just after starting the application. In this case, AFM system has not reached the tip-sample contact to determine the scanner position as reliable.

```
{
    "command": "response",
    "object": "ScannerPosition",
    "payload": {
        "reliability": "unreliable",
        "value": 8162.79
    }
}
```

ScannerRange

Description

Scan range

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerRange",
    "payload": {
        "property": "value",
        "value": 10.9977
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerRange",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

Notes

Range is limmited to the calibrated values stored in the ini file

ScannerResolution

Description

Scanning resolution in pixels

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerResolution",
    "payload": {
        "property": "index",
        "value": 1
    }
}
```

Expected SET Values

unsigned int type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerResolution",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

unsigned int for "index", string for "text"

Notes

Exemplary response:

```
{
    "command": "response",
    "object": "ScannerResolution",
    "payload": {
        "value": {
            "index": 1,
            "text": "128x128"
        }
    }
}
```

The returned index is the position in the available resolution colletion and the corresponding resolution in pixels

ScannerRotation

Description

Scan rotation value

Set/Get Information

set / get

Set Example

```
{
    "command": "set",
    "object": "ScannerRotation",
    "payload": {
        "property": "value",
        "value": 12.34
    }
}
```

Expected SET Values

float type as "value" in the SET command

Get Example

```
{
    "command": "get",
    "object": "ScannerRotation",
    "payload": {
        "property": "value"
    }
}
```

Expected GET Values

float type as "value" in the GET response

APPENDIX A:
Real-Time Line Data Plotting with Python, tkinter and matplotlib

```
"""
Real-Time Measurement Data Plotting Script

This script connects to a WebSocket server, authenticates using an API key,
subscribes to measurement data on a specified channel, and plots the data
in real-time using Tkinter and Matplotlib.

Features:
- Connects to a WebSocket server and authenticates using an API key.
- Subscribes to measurement data of types 'line' and 'map' on a specified channel.
- Receives and processes incoming measurement data in real-time.
- Plots line data (forward and backward scans) and map data using Matplotlib embedded in
Tkinter.
- Dynamically updates the plots as new data arrives.
- Adjusts the map plot to display the data correctly.

Requirements:
- Python 3.7 or higher.
- Packages: websockets, numpy, matplotlib, tkinter.

Configuration:
- `SERVER_IP`: The IP address of the WebSocket server.
- `SERVER_PORT`: The port number of the WebSocket server.
- `API_KEY`: Your API key for authentication with the server.
- `CHANNEL`: The channel number to subscribe to for measurement data.

Notes:
- Ensure that the server is running and accessible.
- Verify that your API key has the necessary permissions.
- The script includes error handling to assist with troubleshooting.
- Mind the license requirements of the used 3rd-party software packages
"""

import asyncio
import json
import tkinter as tk
from datetime import datetime
import websockets
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

# Replace these with your server's IP and port
SERVER_IP = "127.0.0.1"  # Update with your server IP
SERVER_PORT = 1234       # Update with your server port
WEBSOCKET_URI = f"ws://{SERVER_IP}:{SERVER_PORT}"

# API Key for authentication
API_KEY = "your-api-key"  # Replace with your API key

# Specify the channel you want to subscribe to
CHANNEL = 0  # Change this to the desired channel number


class PlotWindow:
    def __init__(self, root):
```

```
        self.root = root
        self.root.title('Real-time Data Plots')

        # Create the line plot figure and canvas
        self.line_fig = Figure(figsize=(5, 4), dpi=100)
        self.line_ax = self.line_fig.add_subplot(111)
        self.line_ax.set_title('Line Data')
        self.line_ax.set_xlabel('X')
        self.line_ax.set_ylabel('Y')
        self.line_forward_line, = self.line_ax.plot([], [], 'r', label='Forward Scan')
        self.line_backward_line, = self.line_ax.plot([], [], 'b', label='Backward Scan')
        self.line_ax.legend()

        self.line_canvas = FigureCanvasTkAgg(self.line_fig, master=self.root)
        self.line_canvas.draw()
        self.line_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        # Create the map plot figure and canvas
        self.map_fig = Figure(figsize=(5, 4), dpi=100)
        self.map_ax = self.map_fig.add_subplot(111)
        self.map_ax.set_title('Map Data')
        self.map_image = self.map_ax.imshow(np.zeros((10, 10)), aspect='auto',
cmap='viridis')
        self.map_canvas = FigureCanvasTkAgg(self.map_fig, master=self.root)
        self.map_canvas.draw()
        self.map_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        # Data queues
        self.line_data_queue = []
        self.map_data_queue = []

        # Start the update loop for tkinter
        self.root.after(100, self.update_plots)

    async def connect_and_subscribe(self):
        try:
            async with websockets.connect(WEBSOCKET_URI) as websocket:
                print("Connected to the WebSocket server.")

                # Authenticate with the API key
                auth_request = {
                    "command": "authenticate",
                    "apikey": API_KEY
                }
                await websocket.send(json.dumps(auth_request))
                response = await websocket.recv()
                response_data = json.loads(response)
                if response_data.get("command") == "error":
                    print("Authentication failed:", response_data)
                    return
                else:
                    print("Authentication successful.")

                # Subscribe to line data
                subscribe_line_request = {
                    "command": "set",
                    "object": "MeasurementDataSubscription",
                    "payload": {
                        "property": "type",
                        "type": "line",
                        "format": "float",
                        "channel": CHANNEL,
                        "subscription": True
```

```
                }
            }
            await websocket.send(json.dumps(subscribe_line_request))
            response = await websocket.recv()
            response_data = json.loads(response)
            if response_data.get("command") == "error":
                print("Line data subscription failed:", response_data)
                return
            else:
                print(f"Subscribed to line data on channel {CHANNEL}.")

            # Subscribe to map data
            subscribe_map_request = {
                "command": "set",
                "object": "MeasurementDataSubscription",
                "payload": {
                    "property": "type",
                    "type": "map",
                    "format": "float",
                    "channel": CHANNEL,
                    "subscription": True
                }
            }
            await websocket.send(json.dumps(subscribe_map_request))
            response = await websocket.recv()
            response_data = json.loads(response)
            if response_data.get("command") == "error":
                print("Map data subscription failed:", response_data)
                return
            else:
                print(f"Subscribed to map data on channel {CHANNEL}.")

            # Listen for incoming measurement data
            async for message in websocket:
                await self.process_message(message)
    except Exception as e:
        print("Error in connect_and_subscribe:", e)

async def process_message(self, message):
    try:
        data = json.loads(message)
        if (data.get("command") == "response" and
            data.get("object") == "MeasurementDataSubscription"):
            payload = data.get("payload", {})
            channel = payload.get("channel")
            if channel != CHANNEL:
                return  # Ignore data from other channels

            data_type = payload.get("type", "")
            value = payload.get("value", {})
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            signal_name = payload.get("signal", "Measurement Data")

            if data_type == "line":
                # Process line data
                x_values = value.get("x", [])
                y_forward = value.get("y_forward", [])
                y_backward = value.get("y_backward", [])

                # Add the data to the line data queue
                self.line_data_queue.append({
                    'x_values': x_values,
                    'y_forward': y_forward,
```

```
                        'y_backward': y_backward,
                        'signal_name': signal_name,
                        'channel': channel,
                        'timestamp': timestamp
                    })

                    # Print timestamp and data receipt message
                    print(f"[{timestamp}] Received line data on channel {channel}.")

                elif data_type == "map":
                    # Process map data
                    map_values = value.get("imageData", [])
                    if not map_values:
                        print("No 'imageData' found in the map data.")
                        return

                    # Add map data to queue
                    self.map_data_queue.append({
                        'map_values': map_values,
                        'signal_name': signal_name,
                        'channel': channel,
                        'timestamp': timestamp
                    })

                    print(f"[{timestamp}] Received map data on channel {channel}.")

                else:
                    print(f"Received unknown data type '{data_type}' on channel
{channel}.")

        except json.JSONDecodeError as e:
            print("Failed to decode JSON:", e)
        except Exception as e:
            print("Error processing message:", e)

    def update_plots(self):
        # Update line plot
        if self.line_data_queue:
            data = self.line_data_queue.pop(0)
            x_values = data['x_values']
            y_forward = data['y_forward']
            y_backward = data['y_backward']
            signal_name = data['signal_name']
            channel = data['channel']
            timestamp = data['timestamp']

            # Update forward scan data
            self.line_forward_line.set_data(x_values, y_forward)
            # Update backward scan data
            self.line_backward_line.set_data(x_values, y_backward)

            # Redraw the line plot
            self.line_ax.relim()
            self.line_ax.autoscale_view()
            self.line_canvas.draw()

        # Update map plot
        if self.map_data_queue:
            data = self.map_data_queue.pop(0)
            map_values = data['map_values']

            # Check if map_values is a valid 2D array
            if isinstance(map_values, list) and len(map_values) > 0:
```

```
                # Assuming map_values is a flat list, reshape it into a 2D array
                map_array = np.array(map_values)

                # Ensure map_array is 2D, and reshape if necessary
                if map_array.ndim == 1:
                    # Assuming it's square or nearly square, reshape to a 2D array
                    size = int(np.sqrt(map_array.size))  # Calculate approximate size
                    map_array = map_array[:size * size].reshape((size, size))

                # Update the image in the map plot
                self.map_image.set_data(map_array)
                self.map_image.set_clim(np.min(map_array), np.max(map_array))  # Set color
limits

                # Redraw the map plot
                self.map_ax.relim()
                self.map_ax.autoscale_view()
                self.map_canvas.draw()

            else:
                print("Map data is not in the expected format.")

        # Schedule next update
        self.root.after(100, self.update_plots)


def run_async_tasks(loop):
    """Run asyncio event loop tasks periodically."""
    loop.call_soon(loop.stop)
    loop.run_forever()


def main():
    root = tk.Tk()
    window = PlotWindow(root)

    # Get the event loop
    loop = asyncio.get_event_loop()

    # Start the WebSocket connection in the background
    loop.create_task(window.connect_and_subscribe())

    # Run asyncio loop periodically in tkinter's mainloop
    def periodic_asyncio():
        run_async_tasks(loop)
        root.after(100, periodic_asyncio)

    # Start tkinter's mainloop and periodically process asyncio tasks
    root.after(100, periodic_asyncio)
    root.mainloop()


if __name__ == '__main__':
    main()
```

APPENDIX B:
Plotting of base64 binary coded line measurement data

```
import json
import base64
import matplotlib.pyplot as plt

# The provided JSON message
json_message = '''paste_your_JSON_message'''


def decode_base64_json_array(b64_string):
    b64_string_clean = ''.join(b64_string.split())
    decoded_bytes = base64.b64decode(b64_string_clean)
    decoded_str = decoded_bytes.decode('utf-8')
    json_data = json.loads(decoded_str)
    return json_data


def main():

    # Parse the JSON message
    data = json.loads(json_message)
    payload = data['payload']
    value = payload['value']

    # Decode the Base64-encoded data arrays
    x_data_json = decode_base64_json_array(value['x'])
    y_forward_data_json = decode_base64_json_array(value['y_forward'])
    y_backward_data_json = decode_base64_json_array(value['y_backward'])

    # Extract the numerical arrays
    x_array = [float(v) for v in x_data_json['x']]
    y_forward_array = [float(v) for v in y_forward_data_json['y_forward']]
    y_backward_array = [float(v) for v in y_backward_data_json['y_backward']]

    # Plot the data using matplotlib
    plt.plot(x_array, y_forward_array, label='y_forward', color='red')
    plt.plot(x_array, y_backward_array, label='y_backward', color='blue')

    # Add legend and labels
    plt.legend()
    plt.xlabel('X Value')
    plt.ylabel('Y Value')
    plt.title('Line Data Plot')

    # Show the plot
    plt.show()

if __name__ == '__main__':
    main()
```