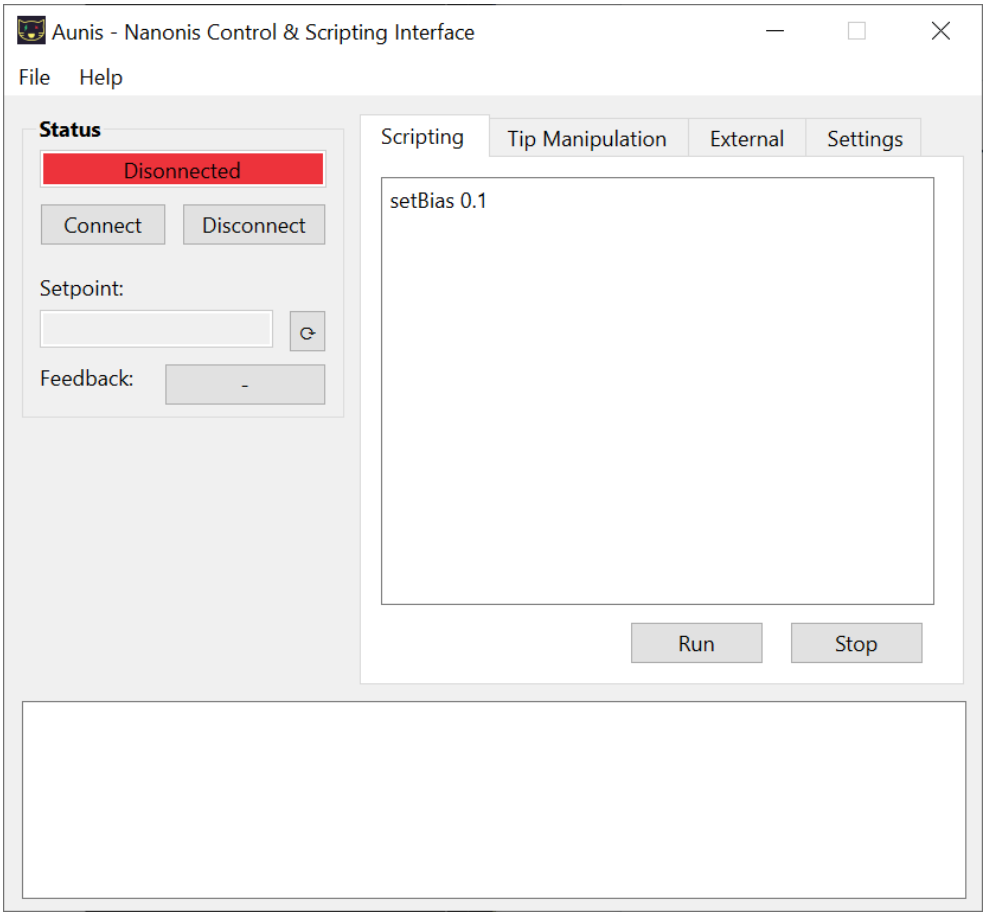
# Manual: Aunis - Nanonis Control & Scripting Interface

**Version: 0.32 (04.05.2025)**

**Dr. Taner Esat**

## Contents

# 1) Graphical User Interface (GUI)




- **Status:** Indicates whether the connection to the Nanonis software is established. If yes, it shows the bias voltage and setpoint current. Feedback can be turned OFF or ON by pressing the button next to the feedback label. The connection with the Nanonis software can be established by pressing the "*Connect*" button. The TCP parameters can be set under **Settings**.
- **Scripting:** A list of several commands (see Scripting Interface section) can be entered in this text area. The commands are executed one after the other by pressing the "*Run*" button. The execution can be stopped by pressing the "*Stop*" button. Note, however, that the current command will be executed to the end. Scripts can be saved or loaded using the *File* menu. An autocomplete feature predicts the rest of the commands a user is entering. The predicted command is selected with the arrow and tab keys.
- **Tip Manipulation:** Allows manual movement of the tip in x-, y- and z-directions. For more details see section Tip Manipulation.
- **External:** Provides an overview of the external TCP interfaces.
- **Settings:** Parameters for the TCP connection with the Nanonis software.

# 2) Scripting Interface

## Commands General

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **repeat [number]**<br>**…**<br>**end** | [number]: (none) | Executes the commands between "repeat" and "end" [number] times.<br><br>**Example:**<br>repeat 10<br>addBias 0.1<br>doScan<br>waitEndScan<br>end |
| **wait [time]** | [time]: (s) | Waits [time] seconds before executing the next command.<br><br>**Example:**<br>wait 10 |

## Commands Feedback

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **getFeedback** | - | Returns the status of the Z-Controller.<br>1: Feedback loop ON<br>0: Feedback loop OFF<br><br>**Example:**<br>getFeedback |
| **setFeedback [status]** | [status]: (none) | Sets the status of the Z-Controller to [status].<br>[status]: 1 – Feedback loop ON<br>[status]: 0 – Feedback loop OFF<br><br>**Example:**<br>setFeedback 1 |

## Commands Bias and Current

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **getBias** | - | Returns the bias voltage in Volts.<br><br>**Example:**<br>getBias |

| | | |
|---|---|---|
| **setBias [bias voltage]** | [bias voltage]: (V) | Sets the bias voltage to [bias voltage].<br><br>**Example:**<br>setBias 0.5 |
| **addBias [bias voltage]** | [bias voltage]: (V) | Adds [bias voltage] to the current bias voltage.<br><br>**Example:**<br>addBias 0.1 |
| **getCurrent** | - | Returns the setpoint current in Ampere.<br><br>**Example:**<br>getCurrent 0.1 |
| **setCurrent [current]** | [current]: (A) | Sets the setpoint current to [current]<br><br>**Example:**<br>setCurrent 200e-12 |
| **addCurrent [current]** | [current]: (A) | Adds [current] to the setpoint current.<br><br>**Example:**<br>addCurrent 100e-12 |

## Commands Scan

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **doScan** | - | Starts a new scan of an image.<br>Note: Next command in the list is executed directly, it does not wait until the scan is finished.<br><br>**Example:**<br>doScan |
| **waitEndScan** | - | Waits until the current scan is finished. Next command in the list is not executed until the scan is finished.<br><br>**Example:**<br>waitEndScan |

## Commands Spectroscopy and Lock-In

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **doBiasSpec** | - | Performs bias spectroscopy.<br><br>**Example:**<br>doBiasSpec |
| **getBiasSpecLimits** | - | Returns the start and end values for the bias spectroscopy.<br><br>**Example**:<br>getBiasSpecLimits |
| **setBiasSpecLimits [start] [end]** | [start]: (V)<br>[end]: (V) | Sets the start and end values for bias spectroscopy to [start] and [end] respectively.<br><br>**Example**:<br>setBiasSpecLimits -250e-3 50e-3 |
| **getLockinFrequency** | - | Returns the frequency of the Lock-In (demod. number 1).<br><br>**Example:**<br>getLockinFrequency |
| **setLockinFrequency [frequency]** | [frequency]: (Hz) | Sets the frequency of the Lock-In (demod. number 1) to [frequency].<br><br>**Example:**<br>setLockinFrequency 187.0 |
| **getLockinAmplitude** | - | Returns the amplitude of the Lock-In (demod. number 1).<br><br>**Example:**<br>getLockinAmplitude |
| **setLockinAmplitude [amplitude]** | [amplitude]: (V) | Sets the amplitude of the Lock-In (demod. number 1) to [amplitude].<br><br>**Example:**<br>setLockinAmplitude 5e-3 |
| **getLockinPhase** | - | Returns the phase of the Lock-In (demod. number 1).<br><br>**Example:**<br>getLockinPhase |
| **setLockinPhase [phase]** | [phase]: (°) | Sets the phase of the Lock-In (demod. number 1) to [phase].<br><br>**Example:**<br>setLockinPhase 55.8 |

## Commands Atom tracking

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **setAtomTrackingModulation [status]** | [status]: (none) | Turns the atom tracking modulation on or off.<br>[status]: 1 – Modulation ON<br>[status]: 0 – Modulation OFF<br><br>**Example:**<br>setAtomTrackingModulation 1 |
| **setAtomTrackingController [status]** | [status]: (none) | Turns the atom tracking controller on or off.<br>[status]: 1 – Controller ON<br>[status]: 0 – Controller OFF<br><br>**Example:**<br>setAtomTrackingController 1 |

## Commands Tip position

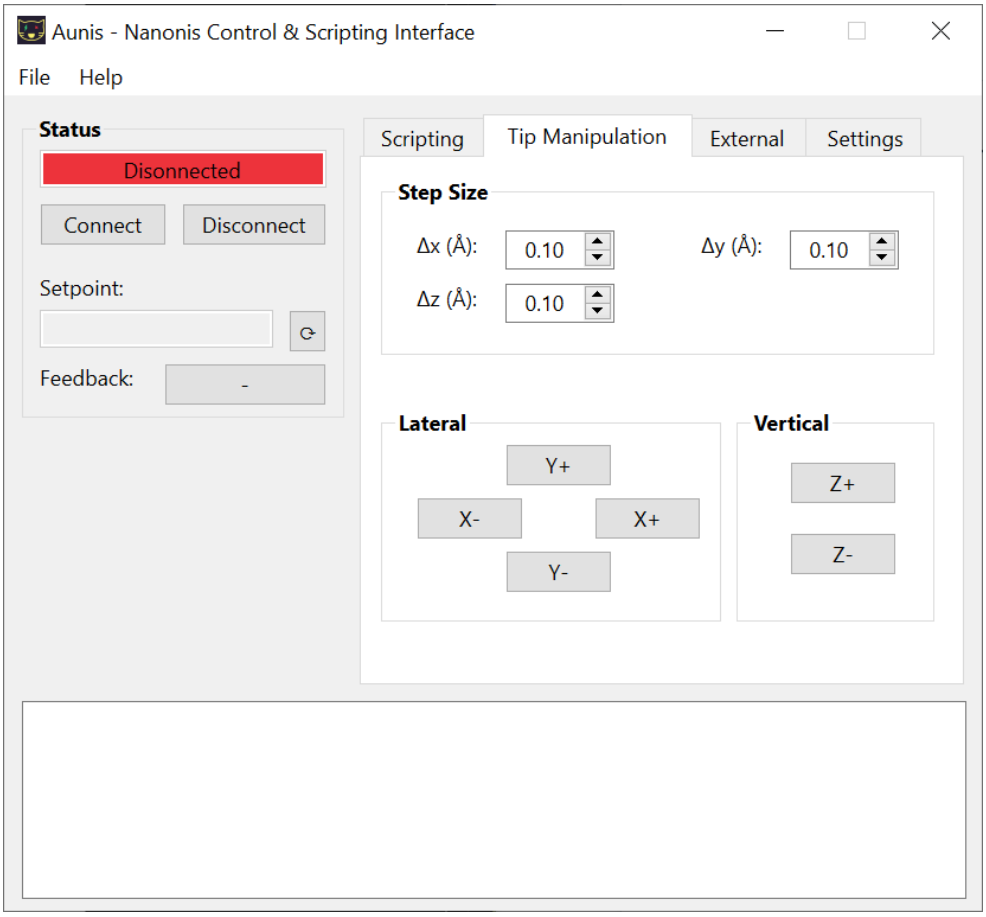
| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **getZ** | - | Returns the Z position of the tip in meters.<br><br>**Example:**<br>getZ |
| **setZ [Z]** | [Z]: (m) | Sets the Z position of the tip to [Z].<br><br>**Example:**<br>setZ 50e-9 |
| **getXY** | - | Returns the X and Y position of the tip in meters.<br><br>**Example:**<br>getXY |
| **setXY [X] [Y]** | [X]: (m)<br>[Y]: (m) | Moves the tip laterally to [X] in x-direction and [Y] in y-direction.<br><br>**Example:**<br>setXY 5e-9 -30e-9 |
| **addX [dx]** | [dx]: (m) | Moves the tip relative to the current position in x-direction by [dx].<br><br>**Example:**<br>addX 100e-12 |

| | | |
|---|---|---|
| **addY [dy]** | [dy]: (m) | Moves the tip relative to the current position in y-direction by [dy].<br><br>**Example:**<br>addY -50e-12 |
| **addZ [dz]** | [dz]: (m) | Moves the tip relative to the current position in z-direction by [dz].<br><br>**Example:**<br>addZ 10e-12 |
| **withdraw** | - | Withdraws the tip.<br><br>**Example:**<br>withdraw |

## Commands Drift compensation

| Command | Parameter: (Unit) | Description and Example |
|---|---|---|
| **getDriftComp** | - | Returns the status and drift compensation values for x-, y- and z-direction.<br><br>**Example:**<br>getDriftComp |
| **setDriftComp [status] [Vx] [Vy] [Vz]** | [status]: (none)<br>[Vx]: (m/s)<br>[Vy]: (m/s)<br>[Vz]: (m/s) | Sets the values for the drift compensation in x-, y- and z-direction.<br>[status]: 1 – compensation ON, 0 – compensation OFF<br>[Vx]: x-direction<br>[Vy]: y-direction<br>[Vz]: z-direction<br><br>**Example:**<br>setDriftComp 1 10e-15 0 30e-15 |
| **correctZDrift [time]** | [time]: (s) | Estimates and corrects the drift in z-direction by measuring drift for [time] seconds.<br><br>**Example:**<br>correctZDrift 120 |

# 3) Tip Manipulation

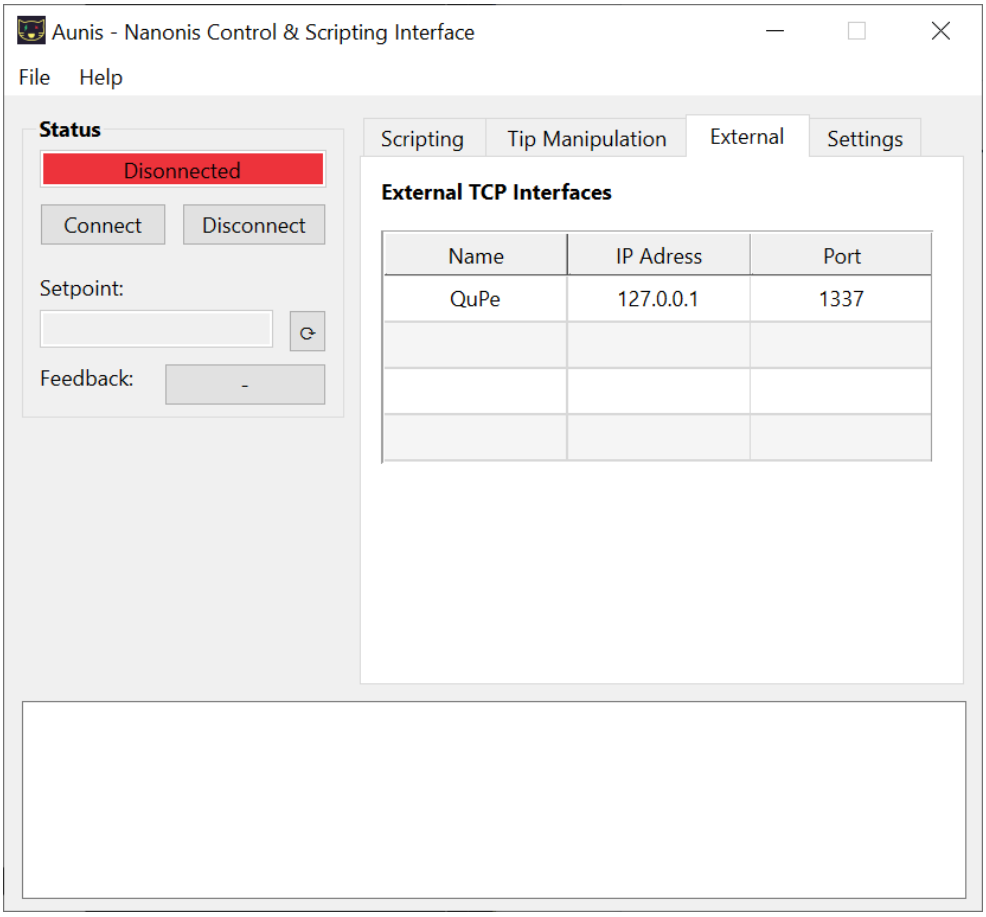



**Tip Manipulation:** Allows manual movement of the tip in x-, y- and z-directions.

The step size in the corresponding direction can be set in the *Step Size* box. By pressing the X+ or X- buttons, the preset step size is added to or subtracted from the current position of the tip. The buttons X+/- correspond to the movement in x-direction, Y+/- to the y-direction and Z+/- to the z-direction.

Note that the tip can only be moved in the z-direction when the feedback loop is switched off.

# 4) External




**External:** Provides an overview of the external TCP interfaces. New TCP interfaces can be added by creating a new JSON file in the "/cmds/external" folder. The structure of the file follows the command structure of the normal commands. Additionally, the entry "Interface" must be created. This contains the parameters for the TCP connection. For the specification of the commands see the section Adding new command - External.

Here is an example to illustrate the syntax of the “Interface” entry:

```
"Interface": {
        "Name": "QuPe",
        "IP-Adress": "127.0.0.1",
        "Port": 1337
    }
```

# 5) Adding new commands

## Normal command

The functionality of Aunis can be easily extended by adding new commands via the JSON file "commands.json" in the folder "/cmds". For this purpose, in addition to the name/alias of the command, the specific arguments for the command must also be defined. This includes the type as well as a default value for the argument. Furthermore, it is possible to specify which arguments can be set via the scripting interface and for which the default values should be used.

Here is an example to illustrate the syntax and structure of a new entry/command:

```
"setXY": {
    "cmdName": "FolMe.XYPosSet",
    "argTypes": {
        "X (m)": "d",
        "Y (m)": "d",
        "Wait end of move": "I"
    },
    "argValues": {
        "X (m)": 0,
        "Y (m)": 0,
        "Wait end of move": 1
    },
    "args": [
        "X (m)",
        "Y (m)"
    ],
    "respTypes": {}
}
```

- The name of the new command is **setXY**.
- The entry **cmdName** specifies the function that is made available via the TCP Programming Interface and is to be executed. In this case the function "FolMe.XYPosSet".
- The entry **argTypes** defines the arguments and their types. These follow directly from the Nanonis TCP protocol. The function "FolMe.XYPosSet" expects the three arguments "X (m)" [type: float64], "Y (m)" [type: float64] and "Wait end of move" [type: unsigned int32]. The types are specified according to the definition in the struct module (see https://docs.python.org/3/library/struct.html#format-characters), i.e. in this case "d" (float64) and "I" (unsigned int32). See also below for an overview of the main data types.
- The entry **argValues** defines default values for the individual arguments. These are only used if the respective argument is not made available via the scripting interface.
- The entry **args** specifies which arguments and in which order they can be set from the scripting interface. In this example only the arguments "X (m)" and "Y (m)" can be set. For the argument "Wait end of move" the default value is always used.

- The last entry **respTypes** defines the return values. The syntax for the last entry is similar to that of **argTypes**. The arguments and their types can be taken from the Nanonis TCP protocol. In this example, no return values are specified. See the next example for the use of return values.

The new command **setXY** then has the following syntax within the scripting interface: **setXY [X] [Y]** Example of a call would be: **setXY 5e-9 -30e-9**

Here is another example to illustrate the handling of return values:

```
"getZ": {
    "cmdName": "ZCtrl.ZPosGet",
    "argTypes": {},
    "argValues": {},
    "args": [],
    "respTypes": {
        "Z position (m)": "f"
    }
```

- The last entry **respTypes** defines the return values. The syntax for the last entry is similar to that of **argTypes**. The arguments and their types can be taken from the Nanonis TCP protocol. In this case there is only the return value "Z position (m)" [type: float32].

Here is an overview of the main data types:

| Data type | Format character | Description |
|---|---|---|
| **string** | s | This is an array of characters, where every character has a size of one byte. |
| **int** | i | 32 bit signed integer. Its range is -2147483648 to 2147483647. |
| **unsigned int16** | H | 16 bit unsigned integer. Its range is 0 to 65535. |
| **unsigned int32** | I | 32 bit unsigned integer. Its range is 0 to 4294967295. |
| **float32** | f | 32 bit (single precision) floating point number. |
| **float64** | d | 64 bit (double precision) floating point number. |

## Special command

Special commands go beyond the capabilities of the normal commands provided through the Nanonis TCP interface and their functionality must be implemented in Python. First, the special commands have to be created in the same way as the normal commands via the JSON file "special_commands.json" in the folder "/cmds". The syntax and structure follows that of the normal commands. However, only the name/alias and **argTypes** and **args** need to be specified in more detail. All other entries are omitted. Furthermore,

the functionality of the special commands must be hardcoded in "PyNanonis.py" in the function **specialCommand(self, cmdAlias, cmdArgs)**.

Here is an example to illustrate the syntax and structure of a new entry/command:

```
    "wait": {
        "argTypes": {
            "Time (s)": "I"
        },
        "args": [
            "Time (s)"
        ]
    }
```

## External command

External commands are created in the same way as normal commands. For each new TCP interface, a separate JSON file must be created in the folder "/cmds/external". These must also contain the entry "Interface" as described in section External. The entry **respTypes** can be omitted.

Here is an example to illustrate the syntax and structure of a new entry/command:

```
"doRFSweep": {
        "cmdName": "constantAmplitudeSweep",
        "argTypes": {
            "StartFreq (Hz)": "s",
            "EndFreq (Hz)": "s",
            "Vrf (V)": "s"
        },
        "argValues": {
            "StartFreq (Hz)": "200e6",
            "EndFreq (Hz)": "800e6",
            "Vrf (V)": "5e-3"
        },
        "args": [
            "StartFreq (Hz)",
            "EndFreq (Hz)",
            "Vrf (V)"
        ]
    }
```

Note that for the **argTypes** entry only the type string "s" is supported, since all commands are sent as bytes representation of the Unicode string. The conversion to the correct data types must be done on the server side.

The example shown here can be executed by calling "**doRFSweep 400e6 800e6 10e-3**" via the scripting interface. Then the string "**constantAmplitudeSweep 400e6 800e6 10e-3**" would be sent in bytes representation to the specified TCP interface.