Manual: Aunis - Nanonis Scripting Interface

Version: 0.41 (28.08.2025)

Taner Esat

Contents

1) Interface Overview

1. Status
Displays whether the connection to the Nanonis software is established.
Details:

- If connected, the current **bias voltage** and **setpoint current** are shown.
- **Feedback control** can be toggled **ON/OFF** by pressing the button next to the *Feedback* label.
- To establish a connection, press the **Connect** button.
- TCP parameters can be configured under **Settings**.

2. Scripting
Provides a text area for entering and executing script commands.
Usage:

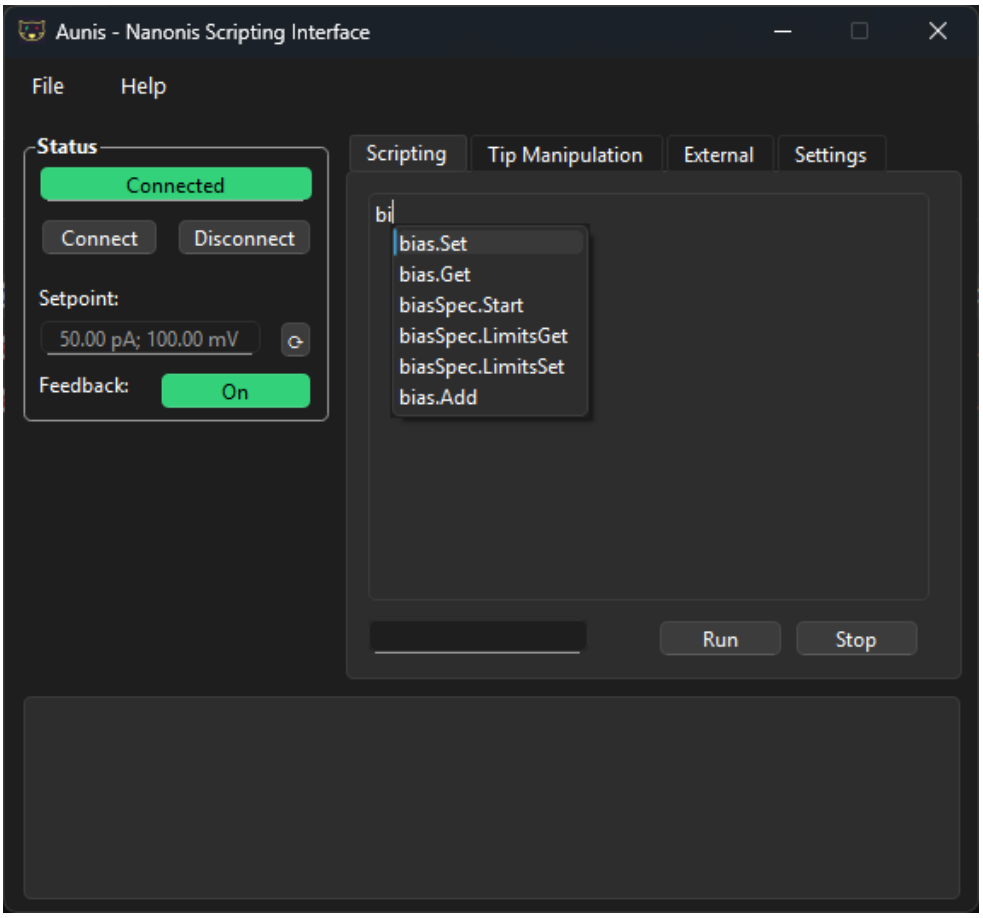
- Enter one or more commands (see *Scripting Commands* section).
- The system performs a syntax check before execution to ensure that all commands are valid.
- Press **Run** to execute commands sequentially.
- Press **Stop** to halt execution (note: the current command runs to completion).
- Use the **File** menu to save or load scripts.
- Use the **Autocomplete** feature:
  - Suggested commands appear while typing.
  - Use **arrow keys** and **Tab** to select a suggestion.

3. Tip Manipulation
Allows manual positioning of the tip along the **x**, **y**, and **z** directions.

- **Reference:** For detailed instructions, see section *Tip Manipulation*.

4. External
Displays an overview of the external TCP interfaces.

5. Settings
Provides configuration options for the TCP connection with the Nanonis software.


Aunis - Nanonis Scripting Interface
File
Help
Status
Connected
Connect
Disconnect
Setpoint:
50.00 pA; 100.00 mV
Feedback:
On
Scripting
Tip Manipulation
External
Settings
bi
bias.Set
bias.Get
biasSpec.Start
biasSpec.LimitsGet
biasSpec.LimitsSet
bias.Add
Run
Stop

2) Scripting Commands

General

Command	Parameter: (Unit)	Description and Example
loop [number] **…** **end**	[number]: (none)	Executes the commands between “loop” and “end” [number] times. Nested loops are also possible. **Example:** loop 10 … end
wait [time]	[time]: (s)	Waits [time] seconds before executing the next command. **Example:** wait 10

Feedback

Command	Parameter: (Unit)	Description and Example
fb.Get	-	Returns the status of the Z-Controller. 1: Feedback loop ON 0: Feedback loop OFF **Example:** fb.Get
fb.Set [status]	[status]: (none)	Sets the status of the Z-Controller to [status]. [status]: 1 – Feedback loop ON [status]: 0 – Feedback loop OFF **Example:** fb.Set 1

Bias and Current

Command	Parameter: (Unit)	Description and Example
bias.Get	-	Returns the bias voltage in Volts. **Example:** bias.Get
bias.Set [bias voltage]	[bias voltage]: (V)	Sets the bias voltage to [bias voltage]. **Example:** bias.Set 0.5

bias.Add [bias voltage]	[bias voltage]: (V)	Adds [bias voltage] to the current bias voltage. **Example:** bias.Add 0.1
current.Get	-	Returns the setpoint current in Ampere. **Example:** current.Get 0.1
current.Set [current]	[current]: (A)	Sets the setpoint current to [current] **Example:** current.Set 200e-12
current.Add [current]	[current]: (A)	Adds [current] to the setpoint current. **Example:** current.Add 100e-12

Scan

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

Spectroscopy and Lock-In

Command	Parameter: (Unit)	Description and Example
biasSpec.Start	-	Performs bias spectroscopy. **Example:** biasSpec.Start
biasSpec.LimitsGet	-	Returns the start and end values for the bias spectroscopy. **Example**: biasSpec.LimitsGet

biasSpec.LimitsSet [start] [end]	[start]: (V) [end]: (V)	Sets the start and end values for bias spectroscopy to [start] and [end] respectively. **Example**: biasSpec.LimitsSet -250e-3 50e-3
lockin.FreqGet	-	Returns the frequency of the Lock-In (demod. number 1). **Example:** lockin.FreqGet
lockin.FreqSet [frequency]	[frequency]: (Hz)	Sets the frequency of the Lock-In (demod. number 1) to [frequency]. **Example:** lockin.FreqSet 187.0
lockin.AmplGet	-	Returns the amplitude of the Lock-In (demod. number 1). **Example:** lockin.AmplGet
lockin.AmplSet [amplitude]	[amplitude]: (V)	Sets the amplitude of the Lock-In (demod. number 1) to [amplitude]. **Example:** lockin.AmplSet 5e-3
lockin.PhaseGet	-	Returns the phase of the Lock-In (demod. number 1). **Example:** lockin.PhaseGet
lockin.PhaseSet [phase]	[phase]: (°)	Sets the phase of the Lock-In (demod. number 1) to [phase]. **Example:** lockin.PhaseSet 55.8

Atom tracking

Command	Parameter: (Unit)	Description and Example
atomtrack.ModSet [status]	[status]: (none)	Turns the atom tracking modulation on or off. [status]: 1 – Modulation ON [status]: 0 – Modulation OFF **Example:** atomtrack.ModSet 1

atomtrack.TrackSet [status]	[status]: (none)	Turns the atom tracking controller on or off. [status]: 1 – Controller ON [status]: 0 – Controller OFF **Example:** atomtrack.TrackSet 1

Tip position

Command	Parameter: (Unit)	Description and Example
z.Get	-	Returns the Z position of the tip in meters. **Example:** z.Get
z.Set [Z]	[Z]: (m)	Sets the Z position of the tip to [Z]. **Example:** z.Set 50e-9
xy.Get	-	Returns the X and Y position of the tip in meters. **Example:** xy.Get
xy.Set [X] [Y]	[X]: (m) [Y]: (m)	Moves the tip laterally to [X] in x-direction and [Y] in y-direction. **Example:** xy.Set 5e-9 -30e-9
x.Add [dx]	[dx]: (m)	Moves the tip relative to the current position in x-direction by [dx]. **Example:** x.Add 100e-12
y.Add [dy]	[dy]: (m)	Moves the tip relative to the current position in y-direction by [dy]. **Example:** y.Add -50e-12
z.Add [dz]	[dz]: (m)	Moves the tip relative to the current position in z-direction by [dz]. **Example:** z.Add 10e-12

withdraw	-	Withdraws the tip. **Example:** withdraw

Drift compensation

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

3) Tip Manipulation

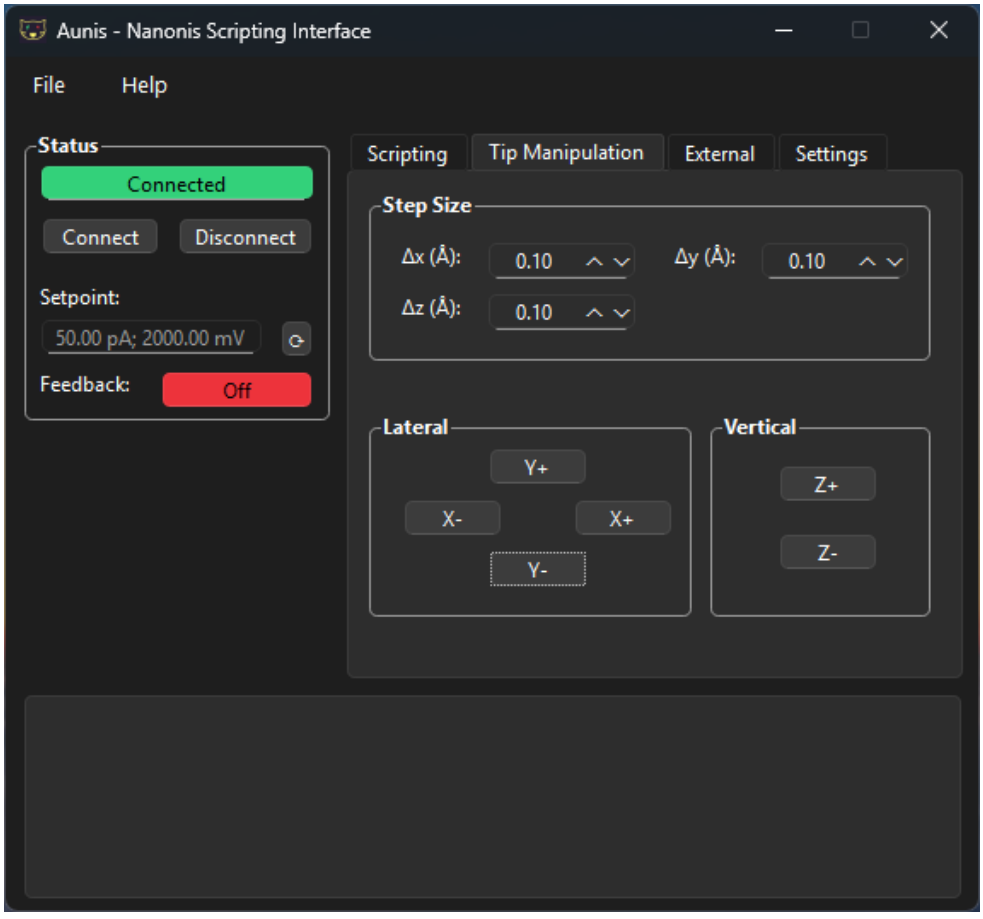
Function: Enables manual movement of the tip along the **x**, **y**, and **z** directions.

Usage:

- Set the desired **step size** in the *Step Size* box.
- Move the tip by pressing the directional buttons:
  - **X+ / X–** → Moves the tip in the x-direction.
  - **Y+ / Y–** → Moves the tip in the y-direction.
  - **Z+ / Z–** → Moves the tip in the z-direction.
- Each button press adjusts the tip position by the preset step size.

Important Note:

- Movement in the **z-direction** is only possible when the **feedback loop is switched OFF**.

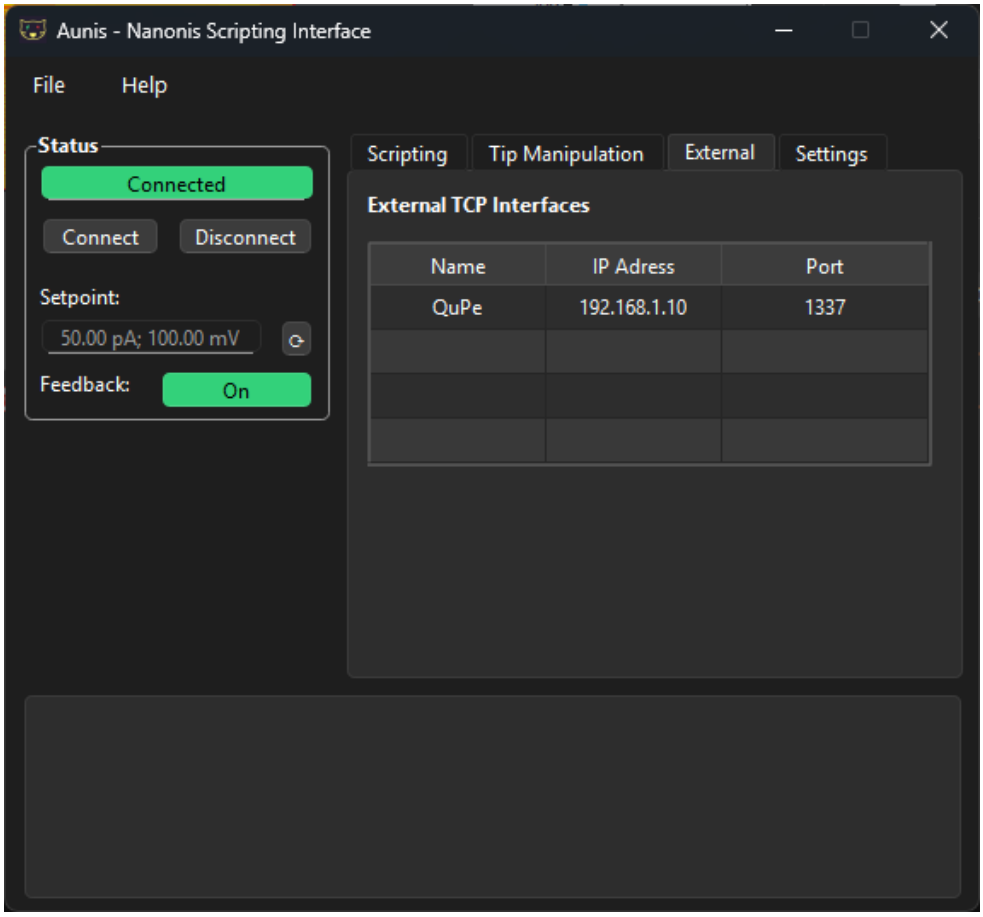


4) External TCP Interface

Function: Provides an overview of the external TCP interfaces. The TCP interface allows to **send commands to other devices** over a network using the TCP protocol. This enables to mix **local function calls** with **remote device control** inside the same script.

Usage:

- Remote commands are added in the same way as normal commands (Adding New Functions to the Command System).



5) Adding New Functions to the Command System

Structure of a Registry Entry

The **command system** works by mapping **command names** (the text the user writes in the script) to Python functions or class methods. These can be functions or class methods provided by the **Python Interface Package for Nanonis (nanonis_spm)**, or custom, self-implemented functions for more complex tasks.

This mapping is stored in the **FUNCTION_REGISTRY**, where each entry describes:

- **Which function to call**
- **What arguments it expects**
- **Which arguments are user-provided vs. default**
- **How arguments should be converted (type casting)**

```
FUNCTION_REGISTRY = {
    "command_name": {
        "func": <function_or_method_reference>,
        "args": {
            "arg1_name": {
                "type": <Python_type>,
                "default": <default_value>,
                "user": <True_or_False>
            },
            "arg2_name": {
                "type": <Python_type>,
                "default": <default_value>,
                "user": <True_or_False>
            },
            ...
        }
    }
```

Explanation of fields:

command_name:

- The text the user must type in the script to call this function.

func:

- A reference to the function or method to be executed.
- Nanonis commands are made available via the **Python Interface Package for Nanonis**.
- For standalone functions: just use the function name (e.g. complexTask).
- For class methods: use the **unbound method** (e.g. nanonis_spm.Nanonis.Bias_Set).
  - The system will automatically inject the correct instance from INSTANCE_MAP. The INSTANCE_MAP is a lookup table that maps class names to specific pre-created objects, so that the parser knows which instance to use when executing class methods.

args: (dictionary of arguments)
Defines the **arguments** required by the function and their respective **data types**.
Each argument must define:

- **"type"**: Python data (numpy) type used to convert the user input (e.g. np.int, np.float32).
- **"default"**: Value used if this argument is not provided by the user.
- **"user"**:
  - ❖ True → user must provide this argument in the script.
  - ❖ False → argument is filled automatically with the default.

Functions and argument definitions for Nanonis commands follow the **Nanonis TCP protocol**.
Arguments are always read **in the dictionary order**.

Examples

These commands are based on Nanonis functions that are provided via the TCP protocol.

```
"lockin.PhaseSet": {
        "func": nanonis_spm.Nanonis.LockIn_DemodPhasSet,
        "args": {
            "Demodulator number": {"type": np.int32, "default": 1, "user":
False},
            "Phase (deg)": {"type": np.float32, "default": 0.0, "user": True},
        },
    },
```

```
"drift.Set": {
        "func": nanonis_spm.Nanonis.Piezo_DriftCompSet,
        "args": {
            "Compensation status": {"type": np.uint32, "default": 1, "user":
True},
            "Vx (m/s)": {"type": np.float32, "default": 0.0, "user": True},
            "Vy (m/s)": {"type": np.float32, "default": 0.0, "user": True},
            "Vz (m/s)": {"type": np.float32, "default": 0.0, "user": True},
            "Saturation limit (%)": {"type": np.float32, "default": 10.0, "user":
False},
        },
    }
```

This command is based on a custom, self-implemented function.

```
    "bias.Add": {
        "func": ScriptingInterface.addBias,
        "args": {
            "Bias value (V)": {"type": np.float32, "default": 0.1, "user": True},
        },
    },
```

Adding a Remote Command

All external devices are defined in the **TCP_INTERFACES dictionary**.

```
TCP_INTERFACES = {
    "QuPe": {"host": "192.168.1.10", "port": 1337},
}
```

Multiple TCP interfaces can be managed via **TCP_INTERFACES**. Each remote command specifies which interface to use.

The **TCPClient class** handles communication. It has two methods:

1. **send(command, args)**
    - Sends a command with arguments to the device.
    - Fire-and-forget (does not wait for a response).
2. **query(command, args)**
    - Sends a command with arguments to the device and waits for a response.
    - Expected format from the device: error|response|variable1,variable2
        - "None" for empty values

Remote commands are added to the **FUNCTION_REGISTRY** just like local ones. Functions are implemented as lambda callings.

1. Decide a **command name** (e.g., " QuPe.FreqSet").
2. Create a **lambda function** that calls **TCPClient**(interface_name).send(...).
3. Define **arguments** in the same style as local commands.

Example:

```
"QuPe.FreqSet": {
        "func": lambda freq: TCPClient("QuPe").send("setFrequency", [freq]),
        "args": {
            "Frequency (Hz)": {"type": np.float32, "default": 1e9, "user": True},
        },
    }
```

- **freq** is the user-supplied argument.
- Defaults are used if user=False.
- The TCP client sends the command **”setFrequency freq” (setFrequency 1e9)** to the interface **QupE**.