

---

# **Rohde & Schwarz Instrument Control Pycharm Plugin**

*Release 3.1.0*

**Rohde & Schwarz**

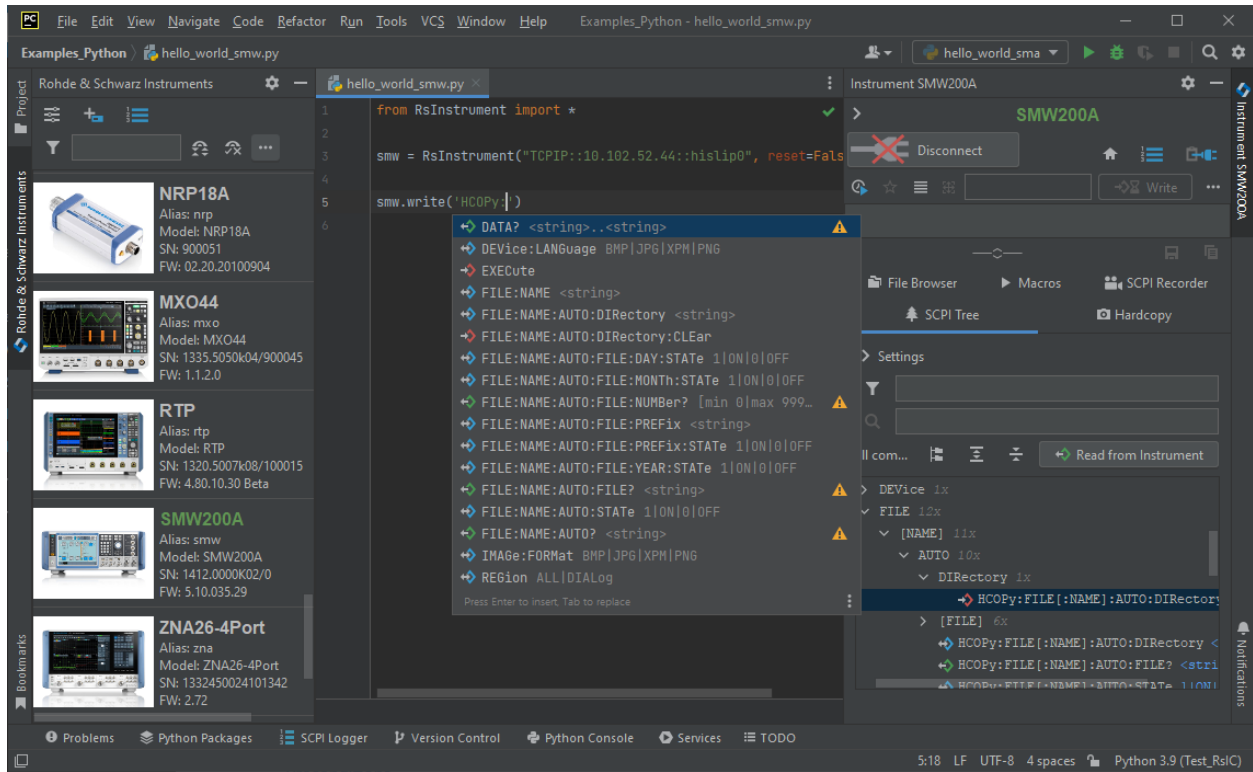
**Mar 22, 2024**



# CONTENTS

<b>1</b>	<b>Revision History</b>	<b>3</b>
<b>2</b>	<b>1. Introduction</b>	<b>7</b>
<b>3</b>	<b>2. Installation</b>	<b>9</b>
3.1	Preconditions . . . . .	9
3.2	Plugin Installation . . . . .	9
<b>4</b>	<b>3. Main Instruments Panel</b>	<b>11</b>
<b>5</b>	<b>4. Adding your first Instrument</b>	<b>13</b>
<b>6</b>	<b>5. Instruments Panel List</b>	<b>19</b>
<b>7</b>	<b>6. Instrument Tool Window</b>	<b>25</b>
7.1	Foldable Info Panel . . . . .	27
7.2	SCPI Communicator . . . . .	27
7.3	Commands History . . . . .	30
7.4	Favorite Commands . . . . .	32
7.5	Multi-line SCPI Editor . . . . .	33
<b>8</b>	<b>7. Function Panel - SCPI Tree</b>	<b>35</b>
8.1	7.1 Reading the SCPI Tree . . . . .	36
<b>9</b>	<b>8. Function Panel - Hardcopy</b>	<b>39</b>
<b>10</b>	<b>9. Function Panel - File Browser</b>	<b>43</b>
<b>11</b>	<b>10. Function Panel - Macros</b>	<b>47</b>
<b>12</b>	<b>11. Function Panel - SCPI Recorder</b>	<b>51</b>
<b>13</b>	<b>12. Function Panel - Curves</b>	<b>53</b>
13.1	12.1 Pre-defined Queries . . . . .	54
<b>14</b>	<b>13. Plain SCPI Scripts Editor</b>	<b>57</b>
14.1	13.1 Writing scripts - SCPI Communicator drag function . . . . .	62
14.2	13.2 Writing scripts - SCPI Auto-completion . . . . .	63
14.3	13.3 Writing scripts - Parsing from other formats . . . . .	64
14.4	13.4 Writing scripts - Searching for commands . . . . .	66
14.5	13.5 Executing scripts . . . . .	68
14.6	13.6 Checking the commands . . . . .	69

14.7	13.7 SCPI Editor color scheme . . . . .	70
14.8	13.8 SCPI Editor Flavors . . . . .	71
<b>15</b>	<b>14. Writing Python Scripts</b>	<b>75</b>
15.1	14.1 Writing scripts - SCPI Communicator drag function . . . . .	78
15.2	14.2 Writing scripts - Auto-completion . . . . .	79
15.3	14.3 Writing scripts - Searching for commands . . . . .	80
15.4	14.4 Writing scripts - Parsing from other formats . . . . .	82
15.5	14.5 Writing scripts - Drag & Drop from SCPI files . . . . .	85
<b>16</b>	<b>15. Advanced SCPI Parser</b>	<b>87</b>
<b>17</b>	<b>16. Plugin Settings</b>	<b>93</b>
17.1	16.1 General settings . . . . .	93
17.2	16.2 Instrument Panels . . . . .	94
17.3	16.3 Instrument Tool Windows . . . . .	95
17.4	16.4 SCPI Code Completion . . . . .	97
17.5	16.5 Code Inspections . . . . .	98
17.6	16.6 Templates . . . . .	99
17.7	16.7 Remote Display . . . . .	100
17.8	16.8 SCPI Editors . . . . .	101
17.9	16.9 Curves . . . . .	102
17.10	16.10 Advanced . . . . .	103
<b>18</b>	<b>Indices and tables</b>	<b>105</b>



See the SCPI Recorder feature in action:



## REVISION HISTORY

### Version 3.1.0 - 31.03.2024

- Added Curves Function panel to conveniently querying traces, waveforms and other array data.
- Added pre-defined curves to 3 different instrument groups: SignalAnalyzer, VectorSignalAnalyzer, Oscilloscope.
- Added support for M3SR radios - new Instrument Profile 'XK41'.
- Fixed NI VISA Parser where it ignored certain write commands.
- Fixed bug with Instrument List when you opened second project.
- Fixed ZNL SCPI Tree Groups.
- Fixed multiple Event Dispatch Thread exceptions.
- Fixed proper sorting of SCPI Groups with different priorities.
- Under the hood: updated JRsInstrument to 1.6.0.

### Version 3.0.0 - 02.02.2024

- Removed support for Pycharm older than 2023.3.
- Partial support for New UI.
- New Instruments List Toolbar for Actions and Filter.
- GUI toolbars changed to standard IntelliJ Action Toolbars.
- Fixed Instrument Tool Window Tabs where clicking directly on the icon did not work.
- Fixed many instances of non EDT-invoke exceptions due to the changes in the IntelliJ framework.
- Advanced SCPI Parser - improved performance and removed the reparse/auto-parse buttons. Added SMx Web Log Format recognition.
- Under the hood: updated org.jetbrains.intellij to 1.17.2, JRsInstrument to 1.5.0.

### Version 2.1.1 - 18.12.2023

- Fixed compatibility with Pycharm 2023.3.x

### Version 2.1.0 - 23.10.2023

- Corrected toggle and action buttons in New UI.
- Updated Pulse Sequencer SCPI Tree.
- Added SCPI code-completion for commands with simple array parameters.
- Improved error message when no VISA is installed.

- Added history function for Instrument List filter box.

### Version 2.0.0 - 21.09.2023

- Removed compatibility with Pycharm 2022.x.
- Full integration of SCPI Language, including new SCPI Editor with customizable color schemes.
- SCPI Multi-liners are now native SCPI Editors with the capability of controlling multiple instruments.
- Template for SCPI files: Use the ProjectTree -> Project right-click -> New -> SCPI File.
- Added advanced SCPI parser - very useful for IO Traces.
- Fixed Communicator Write/Query functions.
- Fixed Communicator waiting Timeout.
- All VISA sessions are closed on Pycharm exit.
- Under the hood: updated org.jetbrains.intellij to 1.15.0, RsInstrument to 1.4.4

### Version 1.3.1 - 11.04.2023

- Added compatibility with Pycharm 2023.1+

### Version 1.3.0 - 01.02.2023

- Added compatibility with Pycharm 2022.3.1+
- Fixed error with calling VISA native DLLs in Pycharm version 2022.3.1+
- Added option to create a Hello-World Python script.
- Improved FSW SCPI Recorder, added SigGens SCPI Recorder.
- Added multi-line window for plain SCPI commands editing and execution.
- Multi-line window has a powerful SCPI parser from different text and log formats.
- Reworked and unified SCPI Drag & Drop actions.
- SCPI Tree: FSW now has an option to change drives.
- Connect Button: added menu item 'Set Error Checking ON/OFF'.
- Under the hood: updated org.jetbrains.intellij to 1.12.0

### Version 1.2.0 - 21.12.2022

- Added compatibility with Pycharm 2022.3.
- Added 1st version of FSW SCPI Interactive Recorder.
- Added VNC, RDP, Web-Control quick launch buttons.
- Added one-click to open the instrument's website (for LAN sessions).
- Added option to import settings XML file from the previous Pycharm Version.
- Fixed text wrapping in the SCPI Communicator for ASCII and BINary responses.
- Fixed SMA100B File Browser.
- Under the hood: updated org.jetbrains.intellij to 1.10.1, RsInstrument to 1.4.1

### Version 1.1.1 - 19.10.2022

- Changed GUIs to flat elements.
- Added support for the new MXO Oscilloscope and CMQ500 Shielding cube.



- Changed and improved instruments list filtering.
- Improved SCPI code-completion.
- Updated documentation.
- SCPI Communicator - improved handling of timeout, error message shows directly in the response field.
- SCPI Communicator - added automatic recognition of binary response.
- SCPI Communicator - added option to have Write / Query buttons visible simultaneously.
- SCPI Communicator - reworked History and added Favorite commands.
- SCPI Tree Function Panel - improved SCPI Tree group names, added items to the context-menu.
- SCPI Tree Function Panel - now it is possible to drag commands to your Python script.
- Macros - added buttons rearrangement possibility, new macro for NGx instruments VNC enabling.
- Templates - reworked handling single/double string quotes.
- Fixed several bugs.
- Fixed RTA/B/M Scpi Tree errors.
- Various improvements.
- Under the hood - updated org.jetbrains.intelliJ to 1.9.0, RsInstrument to 1.2.0

Version 1.0.1 - 01.09.2022

First Released Version



## 1. INTRODUCTION



Rohde & Schwarz Instrument Control Pycharm plugin (RsIC) is targeted for our customers that use Pycharm for writing python instrument remote-control scripts. It is optimized, although not exclusive to be used with our [RsInstrument Python package](#).

**The plugin offers the following key features:**

- Instruments management - you have list of your instruments with all their important information in one place.
- Interactive SCPI communicator GUI for each instrument.
- Plain SCPI commands editor, similar to R&S Forum, but with SCPI Syntax checks, code completion, and more interactive commands execution.
- SCPI Tree - you can read out all the SCPI commands supported by your instrument.
- SCPI Recorder - control you instrument locally, and get the SCPI commands immediately into your script. See <https://youtu.be/pURxIsOwRBI>
- SCPI commands auto-completion - the plugin helps you to write your SCPI commands faster and error-free.
- Hardcopy screenshot - quickly make screenshot of your instrument's screen and save it to your PC.
- File Browser - use CTRL+C / CTRL+V to seamlessly transfer files from / to instrument or between instruments.
- Macros - execute python scripts for different instruments with one click.
- More interesting features coming in the future...

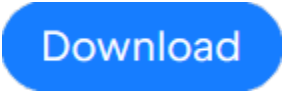


## 2. INSTALLATION

You can use the plugin on Windows, macOS, or Linux.

### 3.1 Preconditions

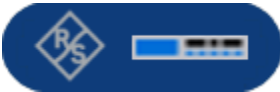
- Install *Pycharm* 2022.2 or newer for Windows, macOS or Linux:

A blue rounded rectangular button with the word "Download" in white text.

- Install Python Interpreter 3.7 or newer:



- Install RsInstrument Python package:



- Install Rohde & Schwarz VISA (optional):



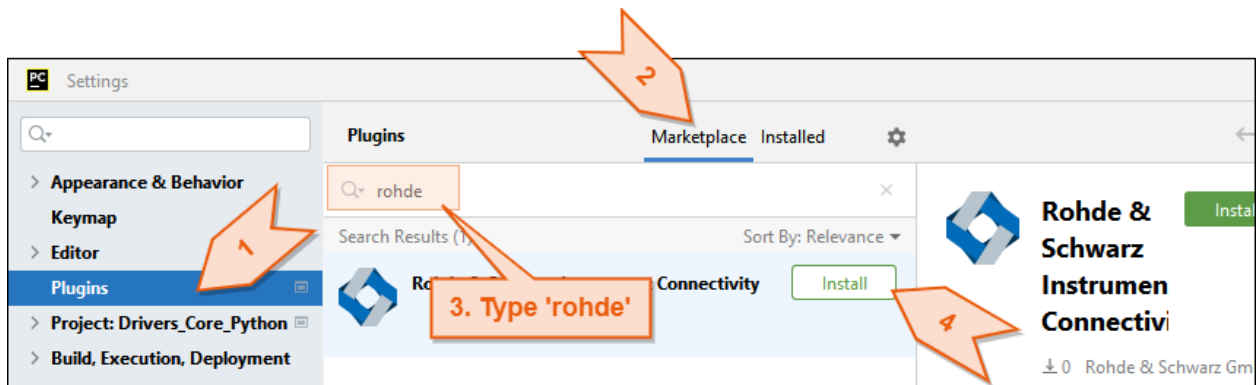
- Install the Pycharm RsIC Plugin:

### 3.2 Plugin Installation

**Installation with this button:**

**Installation from the Marketplace:**

Go to the menu *File -> Settings -> Plugins* and choose the Tab *Marketplace*:



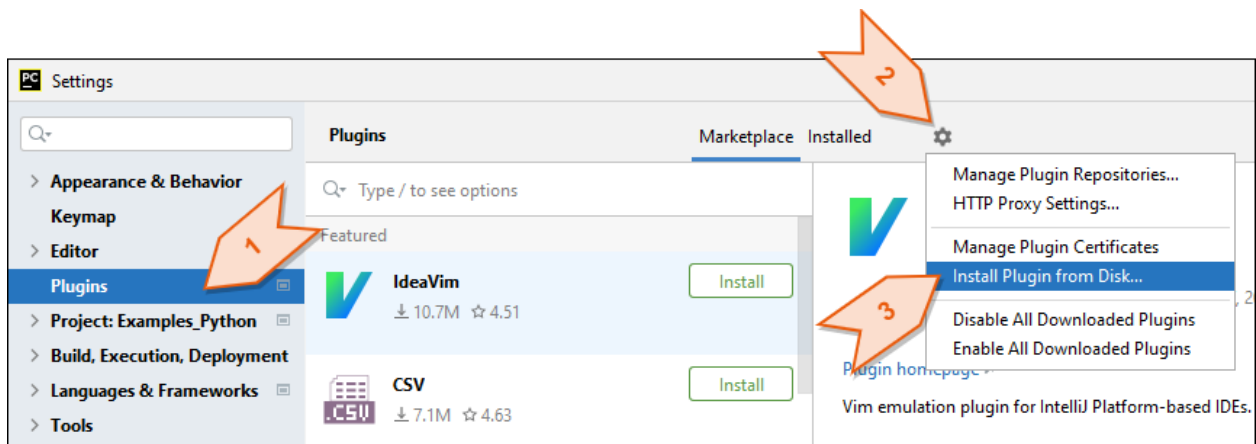
Type the search text, for example 'rohde', and hit install.

**Hint:** If you do not see the plugin listed, you have a Pycharm version older than **2023.1**. Update your Pycharm to the newest version, and repeat the process.

After that, you have to restart the Pycharm IDE.

### Installation from a local file:

In Pycharm, go to menu *File -> Settings -> Plugins* and choose *Install Plugin from Disk*:



Select the plugin **zip** file, e.g.: `RsConnectivityPycharmPlugin-2.0.0.zip`

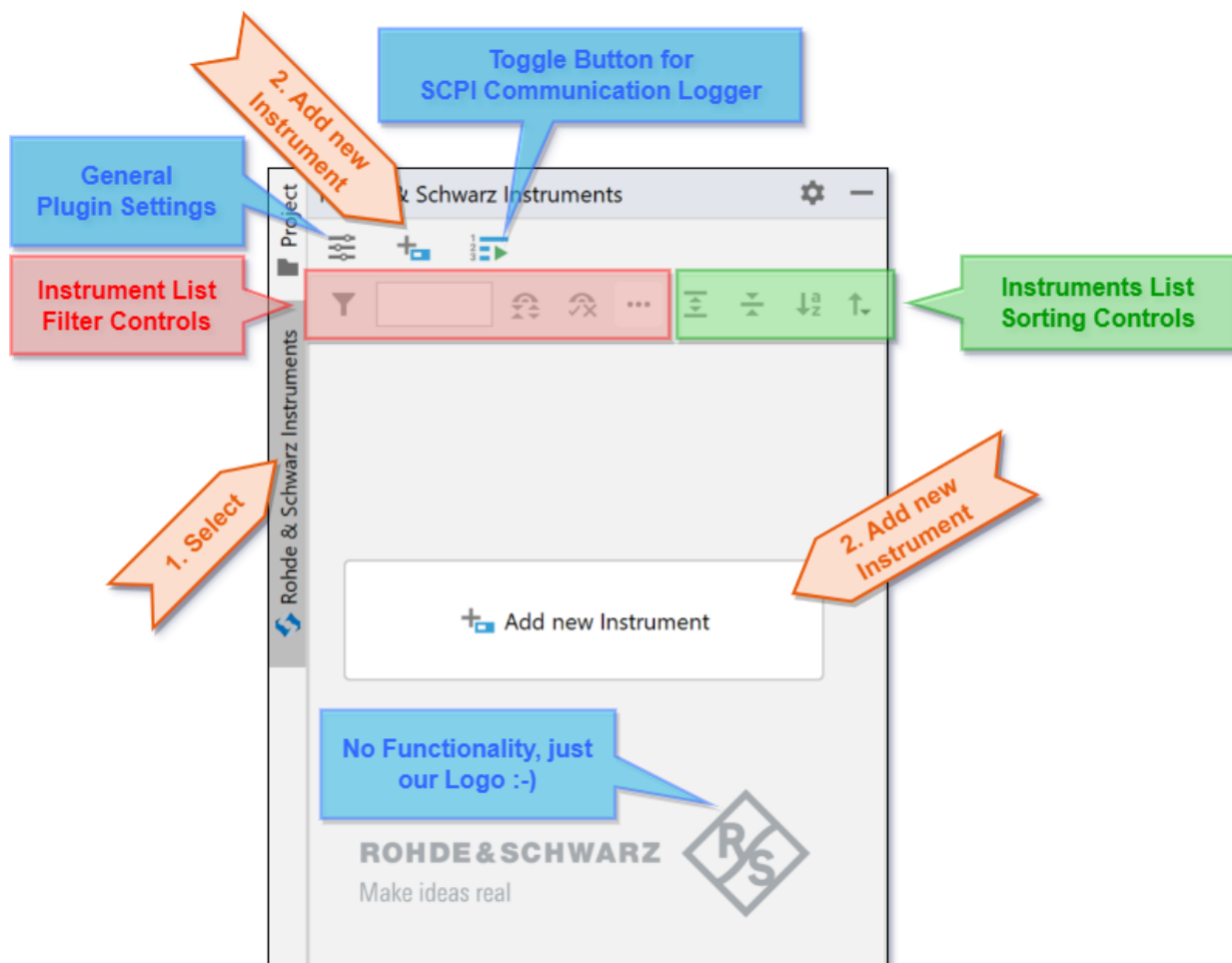
After that, you have to restart the Pycharm IDE.

**Warning:** Do not unpack the **.ZIP** file!!! Select the **.ZIP** file for the installation. Although Pycharm allows you to select the included **.JAR** files, the plugin will not work properly.

**Hint:** On **macOS**, try to avoid the offline installation. The reason is: Safari with its default settings **Open safe files** after downloading extracts the zip file and deletes it, which complicates the installation process - you have to pack the files again to a zip file or change the default Safari settings.

### 3. MAIN INSTRUMENTS PANEL

When you start the Pycharm IDE with the RsIC plugin installed, the IDE shows an additional Tool Window **Rohde & Schwarz Instruments**. Click on the Tool Window gutter to open it - see the orange arrows:



---

**Hint:** On rare occasions, the tab *Rohde & Schwarz Instruments* is not visible. In such cases, you can always open the window from the Pycharm menu *View->Tool Windows->Rohde & Schwarz Instruments*

---





## 4. ADDING YOUR FIRST INSTRUMENT

Click on + Add new Instrument button (either one), and the following dialog comes up:

PC Add New Instrument

HiSLIP

VXI-11

Socket

USB-TMC

GPIB

Serial

Custom

Search for LAN Instruments

IP Address or Hostname

10.102.52.42

Test

Ping

Instance

0

Instrument Active

Name

New

Alias (script variable name)

Family

Other

Result Resource Name

TCPIP::10.102.52.42::hislip0

Advanced Options ...

Select Icon ...

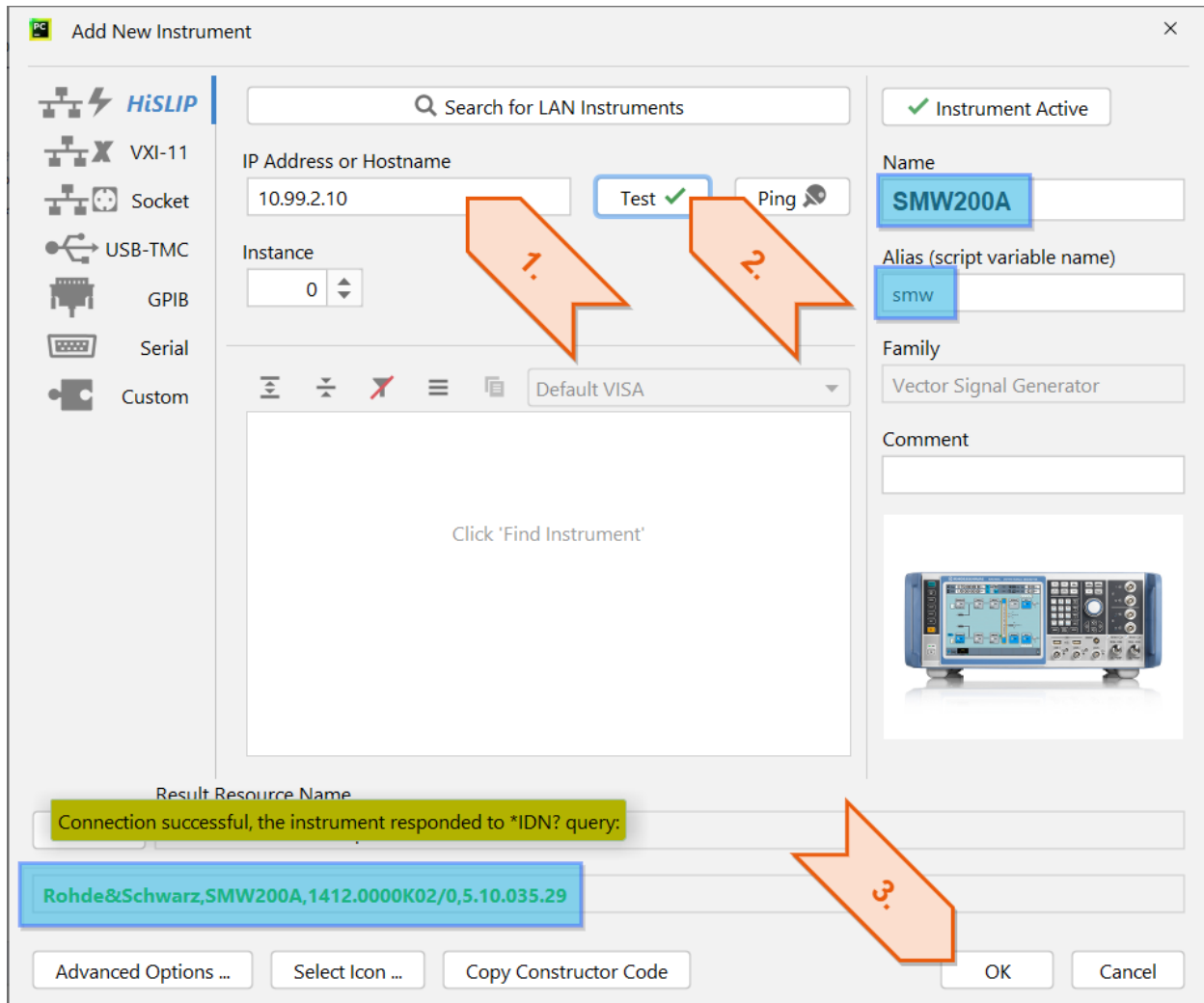
Copy Constructor Code

OK

Cancel

We add a typical LAN instrument with HiSLIP interface. If you wish to switch the session type to VXI-11 or USB-TMC, do so with the tab selector on the left side.

Enter the IP Address or the computer name and hit the **Test** button. We are lucky, our instrument connection worked, the plugin fills out all other fields for us:



**Name, Alias, Icon Picture** come pre-filled with the values parsed from the instrument's IDN response:

- **Name** - has to be unique in your list of instruments.
- **Alias** - in your case `smw`, you are going to reference your instrument from python scripts.
- **Family** - non-editable field, it is set according the internal table to specify the family of your instrument.
- **Comment** - add custom comment/note to this instrument.

**Hint:** You can use the same **Alias** by multiple instruments, but only one can be set to active. The reason is practical - you can quickly switch between different connection types for your `smw`, without changing anything in your python script. Change the active/inactive status with the button **Instrument Active**. Inactive instruments work just like the active ones, except of SCPI auto-completion in your python scripts. There, the SCPI auto-completion relates only to the currently active instruments.

**Tip:** If your LAN connection does not work, try to use the **Ping** button to see whether the instrument is available on LAN. Searching for LAN instruments only works in certain scenarios. If your instrument does not appear in the found list, it does not necessary mean that it is not available. The Search feature is very helpful with USB-TMC instruments, where it finds all of them:

PC

Add New Instrument

×

HiSLIP

VXI-11

Socket

**USB-TMC**

GPIB

Serial

Custom

Search for USB-TMC Instruments

Vendor ID

0x0AAD

Product ID

0x014E

Test

✓

Serial Number

900051

Default VISA

RSNRP::0x014e::900051::INSTR

USB 0xAAD (Rohde & Schwarz)

USB::0x0AAD::0x014E::900051::INSTR - NRP18A

✓ Instrument Active

Name

NRP18A


Alias (script variable name)

nrp

Family

Power Sensor

Comment



Result Resource Name

USB::0x0AAD::0x014E::900051::INSTR

ROHDE&SCHWARZ,NRP18A,900051,02.20.20100904

Advanced Options ...

Select Icon ...

Copy Constructor Code

OK

Cancel

**Advanced Options** button allows you select additional settings for your session, like VISA selection (we hope you use R&S VISA :-), VISA Timeout, Logging mode, etc... :

15

PC

Configure Advanced Options

✕

Visa Selection:

☐ Default

☒ Rohde & Schwarz

☐ No Visa (Socket IO)

Termination Character:

☒ LF (default)

☐ CR

☐ 0x00

☒ Simulate (No)

☒ \*STB? in Error Checking (Yes)

☒ Query Instrument Status (Yes)

☒ Skip Status System Settings (No)

☒ Write with Term Char (No)

☒ Disable OPC Query (No)

☒ Add Term Char to Write Bin (No)

☒ Clear Status (No)

☒ \*OPC? after each Write (No)

☒ ...-capable (interface dependent)

Visa Timeout in ms:

Logging Mode:

Off

▼

OPC Timeout in ms:

Logging Alias Name:

Write Delay in ms:

☒ Log to Console (0)

Read Delay in ms:

☒ Log to UDP (0)

Data Chunk Size in Bytes:

☒ Log global target (0)

Profile Selection:

None

▼

Additional Settings

Result Settings String

VisaTimeout = 3000, SelectVisa = rs

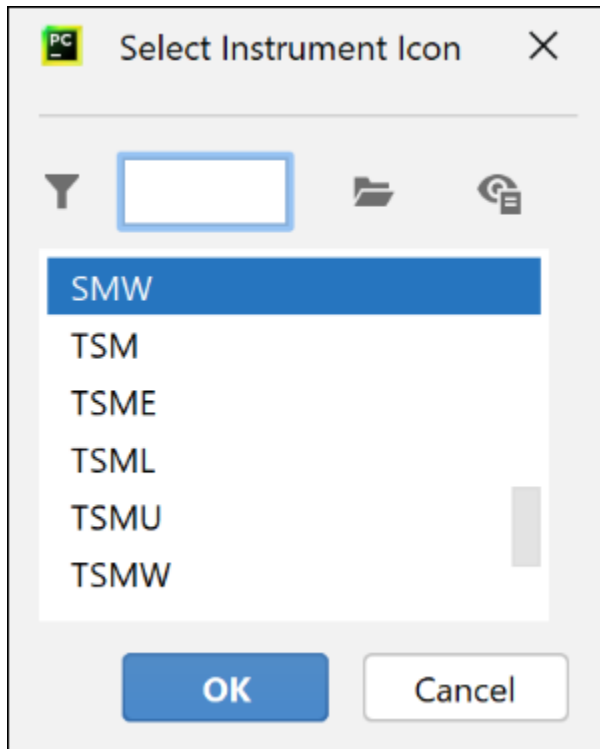
Result Options Settings String

Set to Default

OK

Cancel

You can also adjust the icon picture or even add your own custom picture with the button **Select Icon** (or single-click on the current icon picture). If possible, use the format 640x480:



**Note:** Settings in this configuration dialog results in a Python code snippet for RsInstrument that you can copy to clipboard by hitting the button **Copy Constructor Code**. Then, use it in your python script to initialize connection to your instrument:

```
smw = RsInstrument('TCPIP::10.102.52.42::hislip0', reset=False, options='SelectVisa = rs,  
↳ VisaTimeout = 3000')
```



## 5. INSTRUMENTS PANEL LIST

Instruments Panels List (**IPL**) serves as an overview of all your instrument, and as a launchpad for dedicated Instrument Tool Windows for each instrument:

Example of the IPL:

The screenshot shows the Rohde & Schwarz Instruments Panel List (IPL) interface. The sidebar on the left contains 'Project', 'Rohde & Schwarz Instruments', and 'Bookmarks'. The main area displays three instruments with their details and icons for various actions. A list of six actions is shown on the right:

1. Collapse the Instrument Panel
2. Open Instrument Tool Window
3. Test the Connection
4. Configure the Instrument
5. Remote Display Access
6. Remove the Instrument

Instrument	Model	Family	Alias	Rsrc	SN	FW
SMW200A	SMW200A	Vector Signal Generator	smw	TCPIP::10.99.2.10::hislip0	1412.0000K02/0	5.10.035.29
ZNA26 4-Port	ZNA26-4Port	Vector Signal Analyzer	zna	TCPIP::10.205.0.60::hislip0	1332450024101342	2.41
MXO44	MXO44	Oscilloscope	mxo	TCPIP::10.112.0.157::hislip0	1335.5050k04/900045	1.1.2.0

Description of the controls:

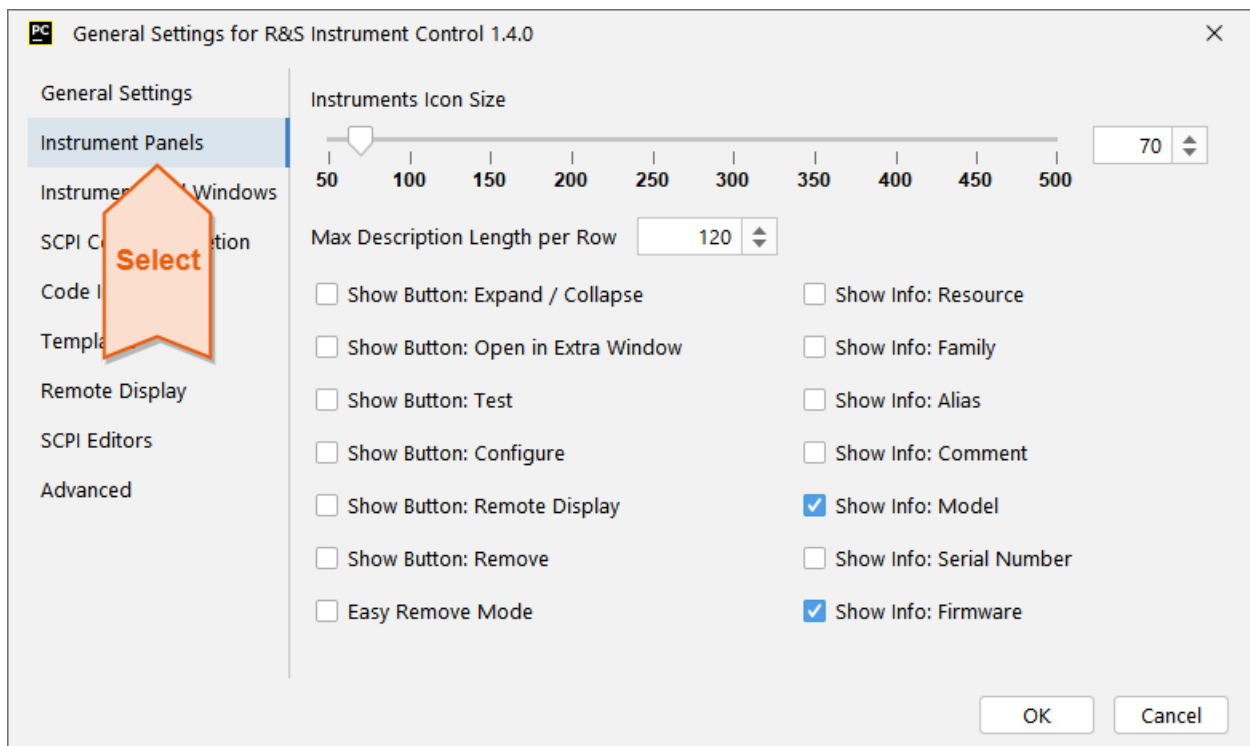
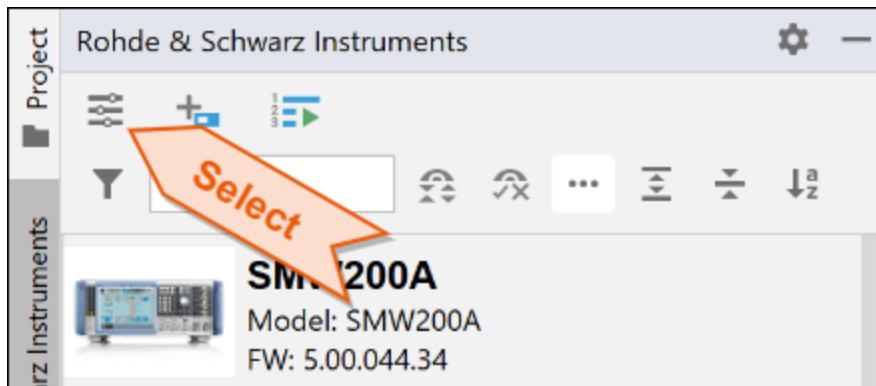
1. **Collapse the Instrument Panel** - you can minimize the Instrument Panel to a small icon and the name.
2. **Open Instrument Tool Window** - opens dedicated Instrument Tool Window (ITW) with all the functions for the instrument. See the [6. Instrument Tool Window](#).
3. **Test the Connection** - tests, whether the connection to your instrument can be established. The connection is closed afterwards.
4. **Configure the Instrument** - opens the instrument's configuration dialog described in [4. Adding your first Instrument](#).
5. **Remote Display Access** - quickly opens Remote Operation either with VNC, Remote Desktop, or Web-control. Configure it with the [context-menu item Remote Display Settings](#).
6. **Remove the Instrument** - removes the instrument from the list.

---

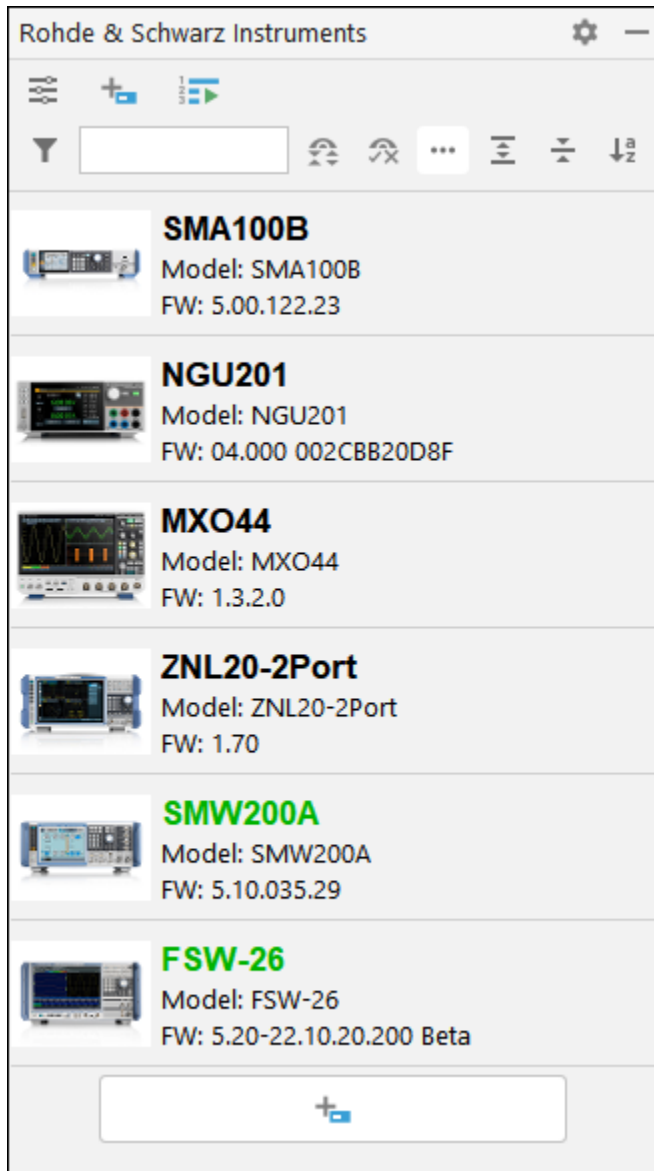
**Tip:** You can adjust the order by drag-and-drop:



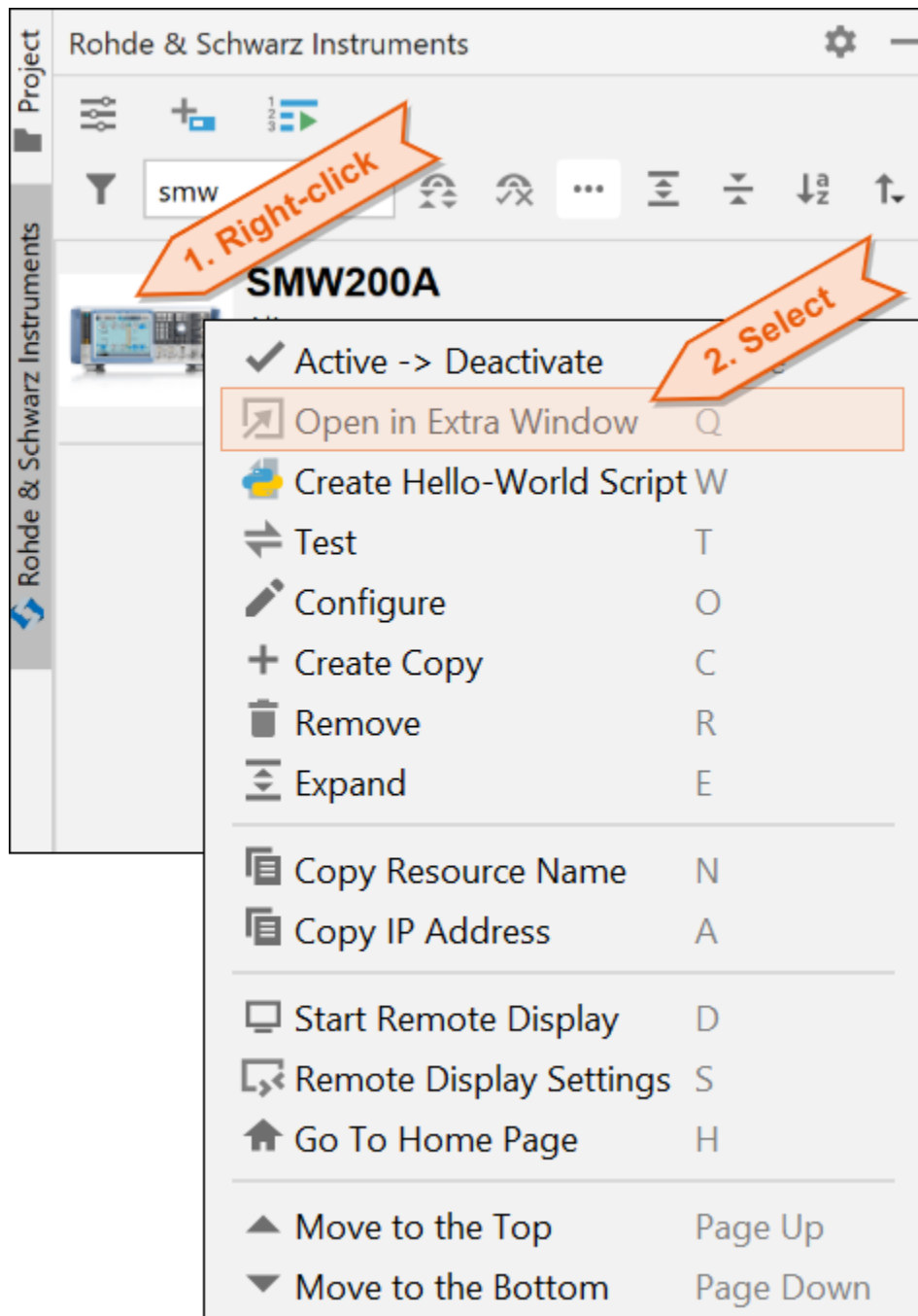
21



This makes your list look more compact:



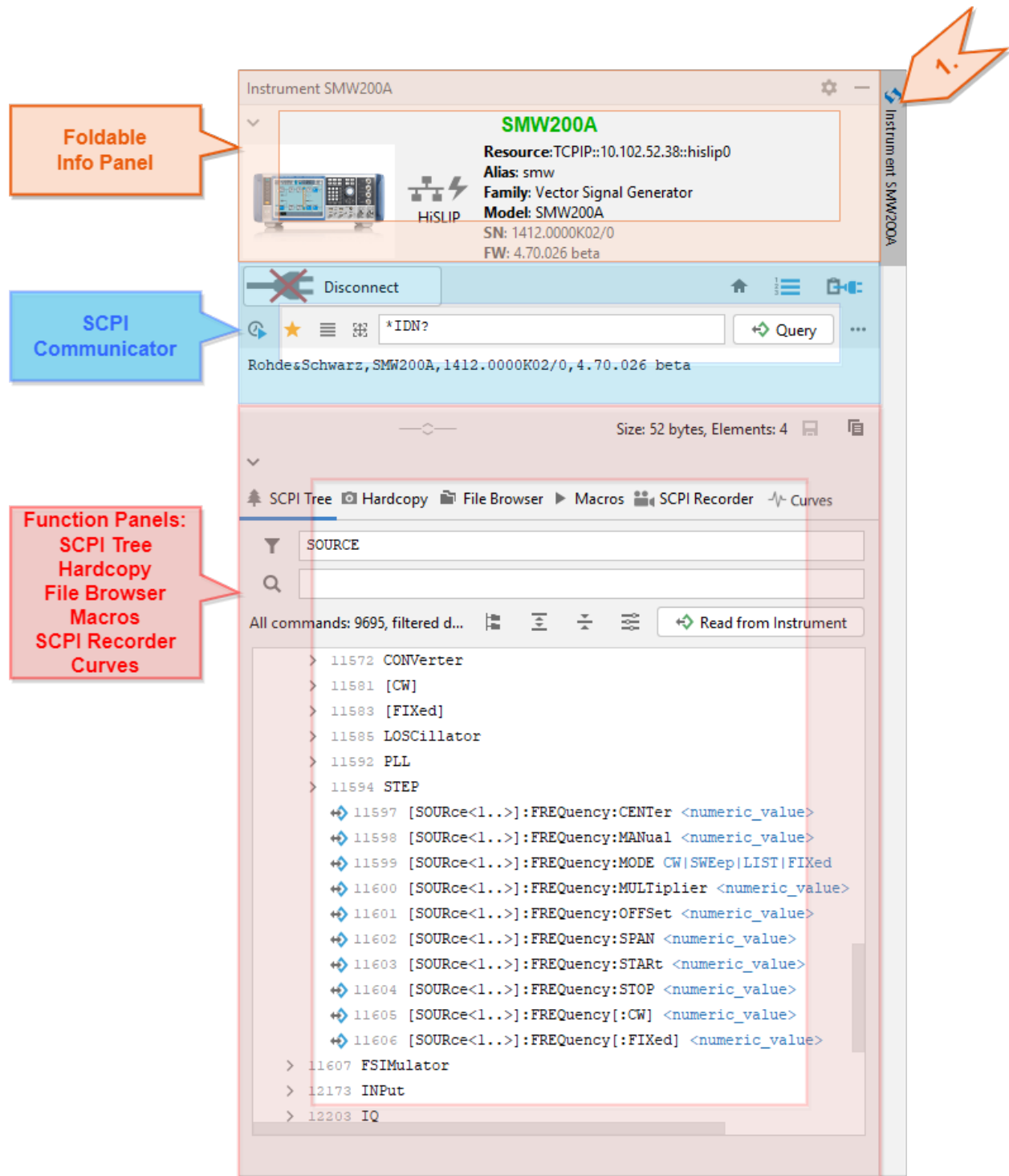
**Tip:** You can reach all the functionalities (and some more) through the context menu: right-click on the instrument picture. Select the **Open in Extra Window** or press 'Q'



**Open in Extra Window** is only available if you have made at least one successful connection to your instrument. This is because the plugin needs to know the instrument's identity.

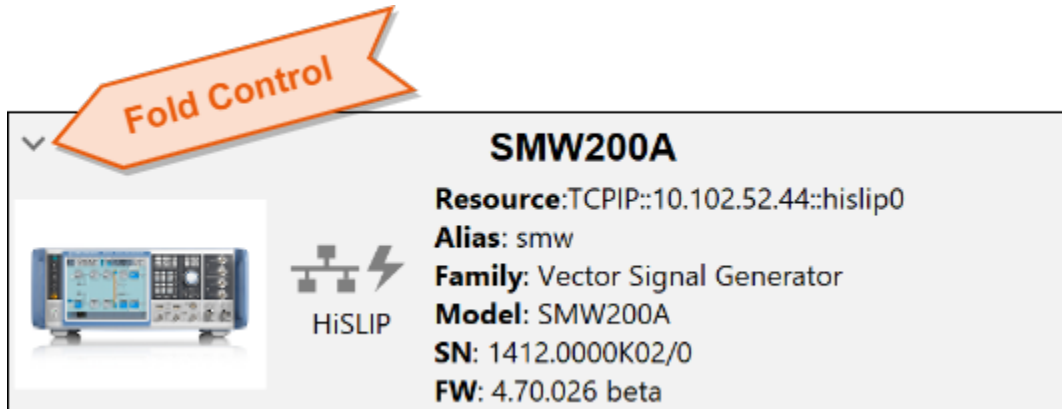
## 6. INSTRUMENT TOOL WINDOW

Instrument Tool Window (**ITW**) provides all the features available for the instrument:



## 7.1 Foldable Info Panel

The **Info Panel** is a header for the Instrument Tool Window. It contains all the important information about your instrument. Fold it to win more vertical space:

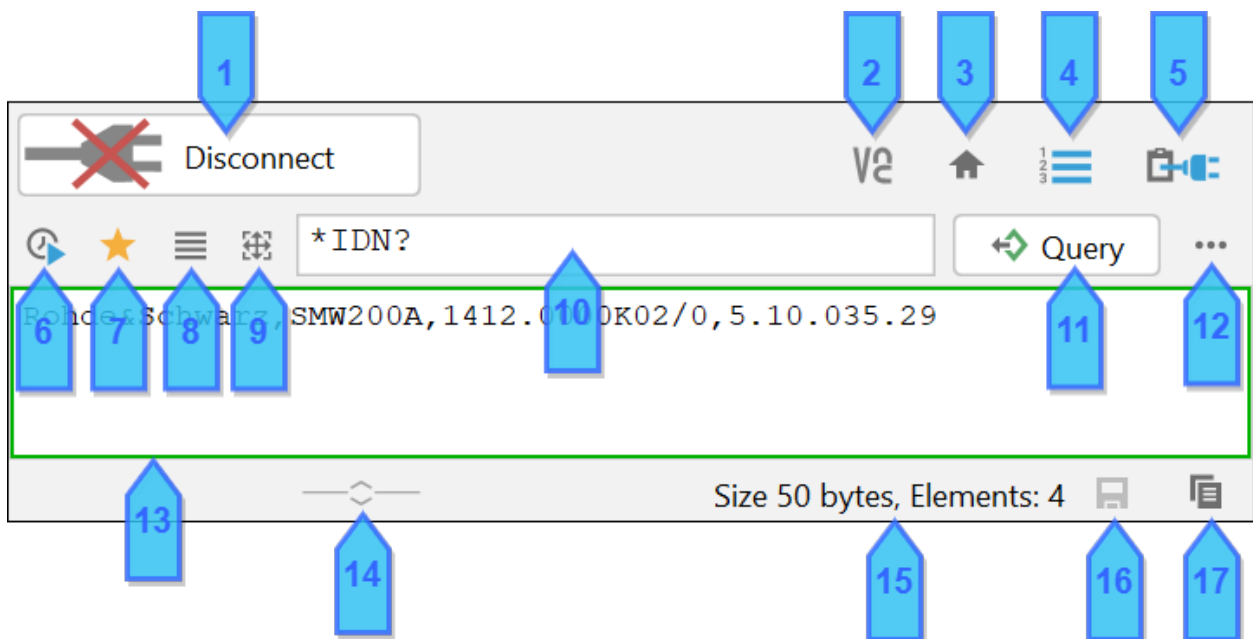


**Hint:**

- **Right-click** on the instrument's icon to get its context-menu, same as in the overview list.
- **Left-click** on the on the instrument's icon to open its Configuration Window.

## 7.2 SCPI Communicator

SCPI Communicator contains controls for connecting and communicating with your instrument. The connection status is valid for all other functionalities in the ITW.



Description of the controls:

1. **Connection Button** - toggle button for connecting and disconnecting to the instrument. When connected, multiple functions of the ITW become available. Right-click allows for sending GoToLocal or GoToRemote signal, Reset or setting VISA Timeout.
2. **VNC Start Button** - start your VNC application with one click. You have to configure the VNC connection for your instrument to see this button.
3. **Visit Home Page Button** - go to your instrument's home web address. Available for instruments with LAN connection and working web-server.
4. **SCPI Logger Toggle** - opens/closes SCPI Logger Tool Window. The Logger logs entire communication with your instrument coming from the plugin, and optionally from your python script (for that, you need to [switch on logging to UDP](#))
5. **Paste import and constructor snippet** - pastes an RsInstrument import statement, plus the instrument's constructor snippet to your currently opened python script.
6. **Commands History** - keeps list of all the commands used in the past. Invoke the quick version of the window by pressing **UP** key in the SCPI Command Field. The full version you can invoke by pressing **DOWN** in the SCPI Command Field (10).
7. **Favorite Commands** - mark your commands as favorite with the left-click, or open the Favorites window with the right-click.
8. **Multi-line SCPI Editor (F6)** - open this popup window to manage multiple lines of SCPI commands. [See below](#).
9. **SCPI Command Drag Button** - drag the SCPI command to your script or to another editor. On drop, you can decide to insert plain string or SCPI call snippet.
10. **SCPI Command Field with Auto-completion** - write your SCPI command here. If your instrument has a SCPI Tree available (see [7. Function Panel - SCPI Tree](#)), this field offers you auto-completion for SCPI commands.
11. **Communication Action Button** - sends/queries the current SCPI command (Field 10) to the instrument. The button changes its function based on the command type (write/query). The response is automatically recognized as binary, and the result display changes accordingly. You can also select the mode with the Field 11 - see the tip below.
12. **More Options** - select the type of write/query action - standard, with OPC, query binary data. The setting is persistent. You can also select whether you want to see two separate buttons for Write and Query.
13. **SCPI Response Field** - text field that contains responses received from the instrument.
14. **SCPI Response Field Resizer** - change the height of the response field to fit your needs.
15. **Response Info** - shows additional response information. For text, it shows its length and if commas are detected, it counts number of elements.
16. **Save Binary Response to File** - this button is enabled, when you have received a binary response from the instrument.
17. **Copy Response to Clipboard** - enabled when the SCPI Response Field (Field 12) is non-empty.

---

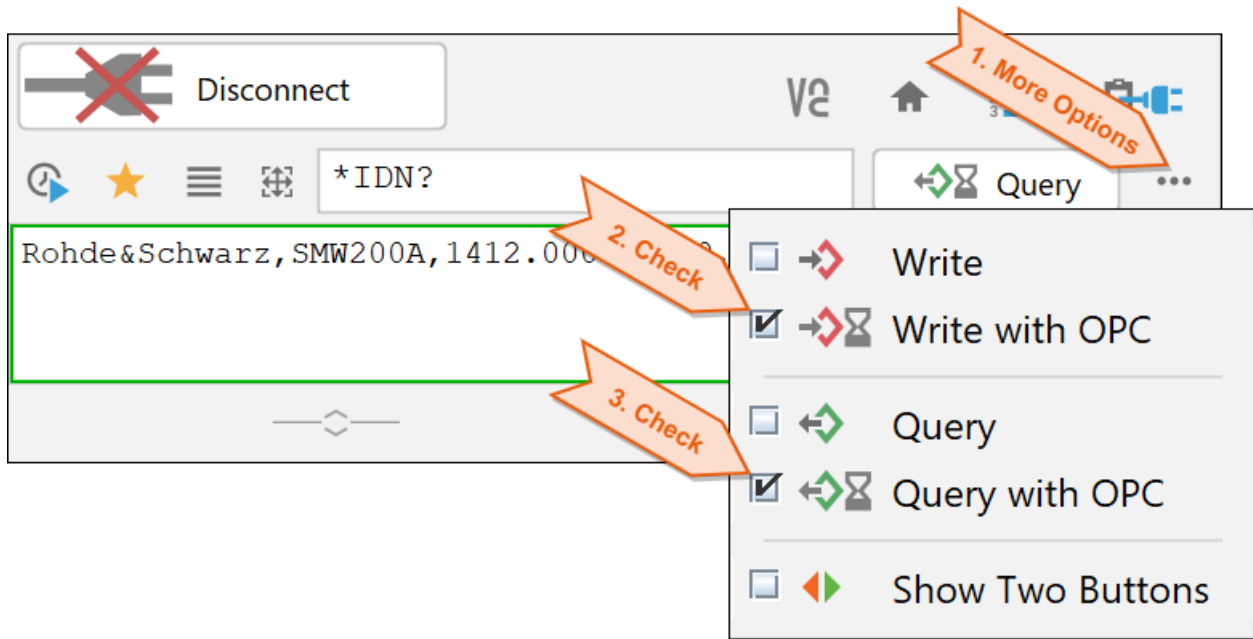
**Tip:** You can configure the look-and-feel of the Communicator in [16.3 Instrument Tool Windows](#)

---

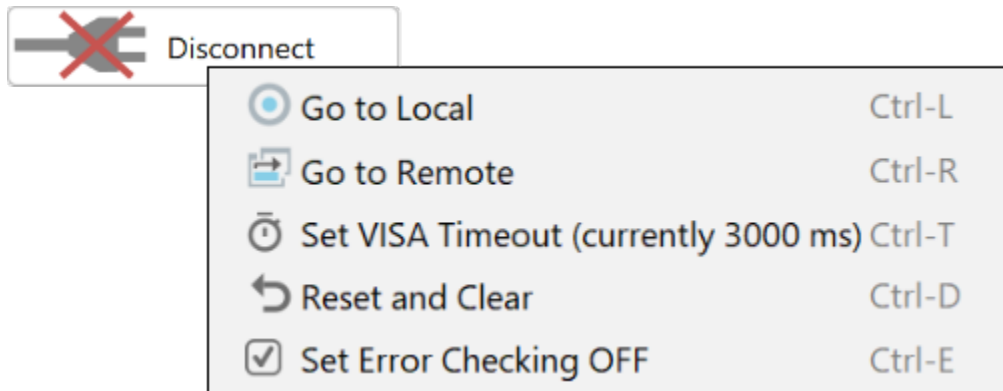
---

**Tip:** If you use HiSlip or VXI-11, switch to the mode write/query with OPC (Field 11). This is very convenient, since in case of an SCPI error, you do not have to wait for VISA timeout:

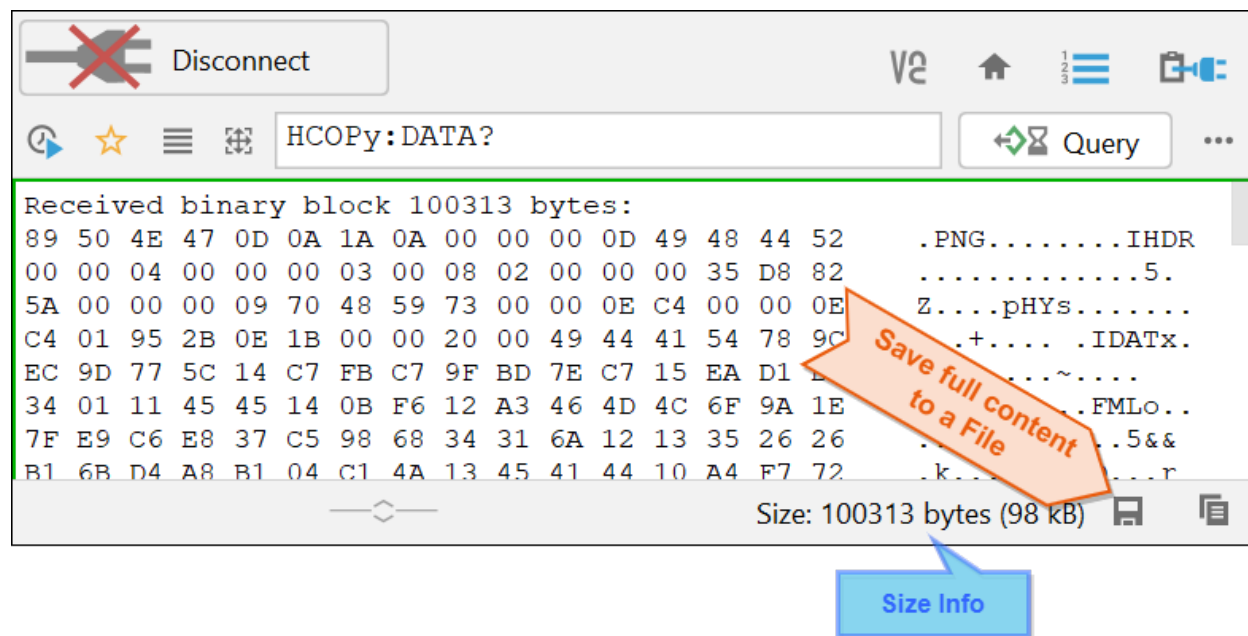




**Hint:** The **Connect/Disconnect Button** (1) has a right-click context menu with some useful features:

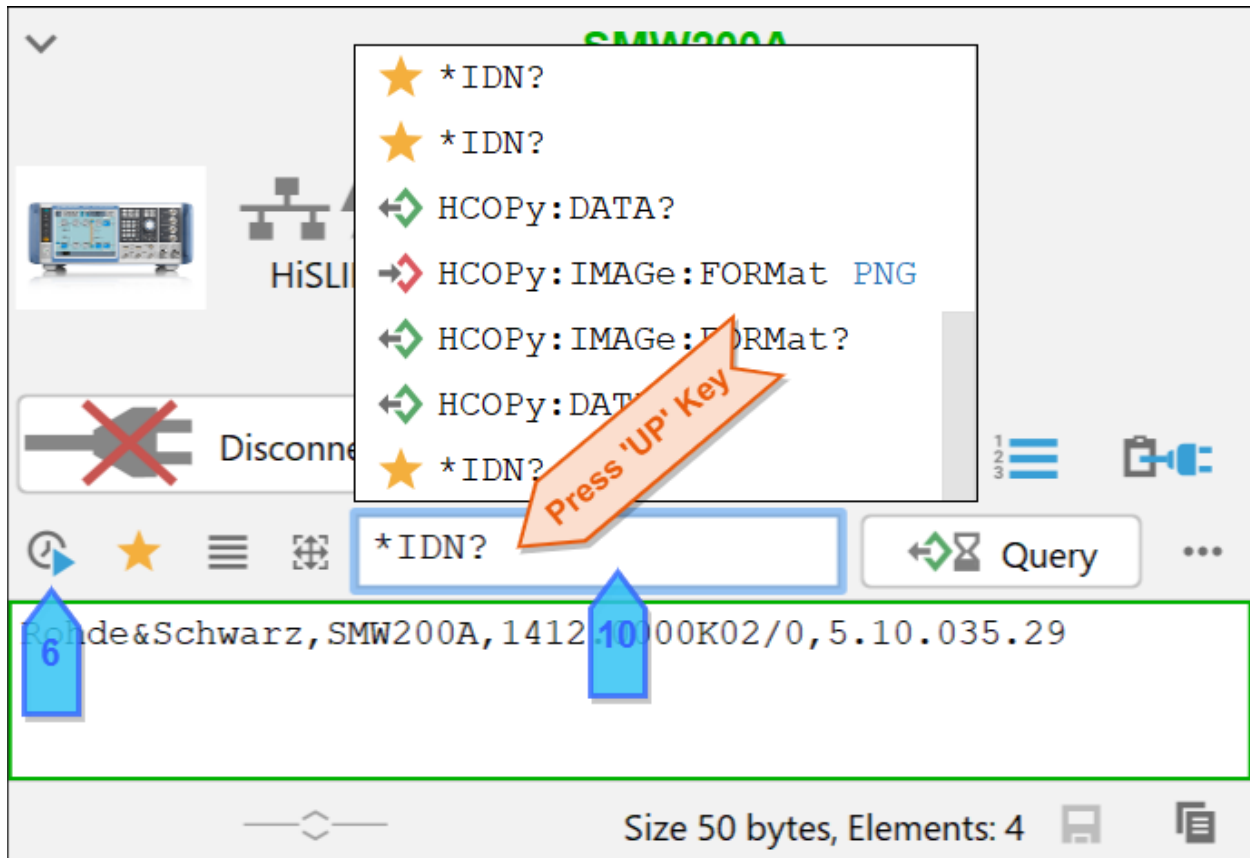


**Note:** If your SCPI Query reads a binary block, the response display changes to show binary data:

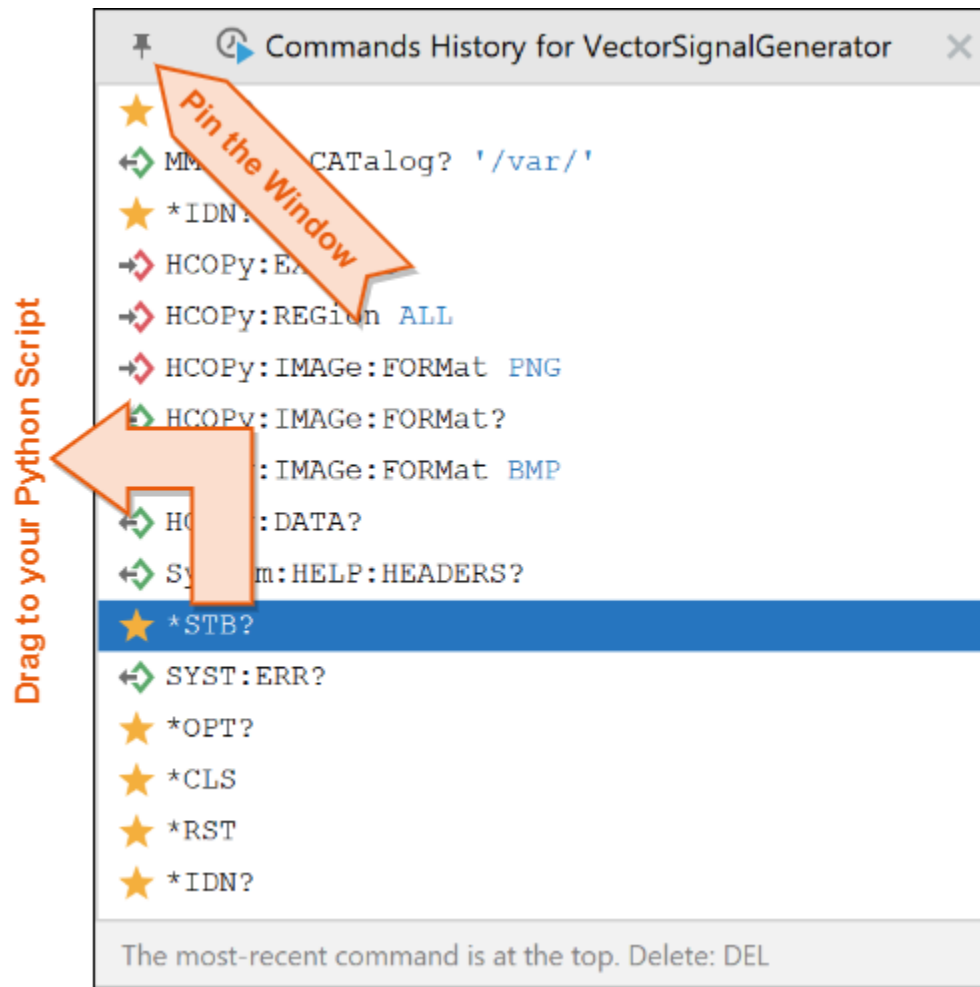


## 7.3 Commands History

Each command you send to your instrument is stored in a persistent history, which is grouped by the instrument's family. That means, that for example, both SMW200A and SMM100A share the same history and favorite commands, because they both belong to the family of Vector Signal Generators. The Commands History window has two versions: **Quick** and **Full**. You can invoke the quick version by pressing **UP** key while typing in the **SCPI Command Field** (10). Quick version has the last command sent at the bottom:

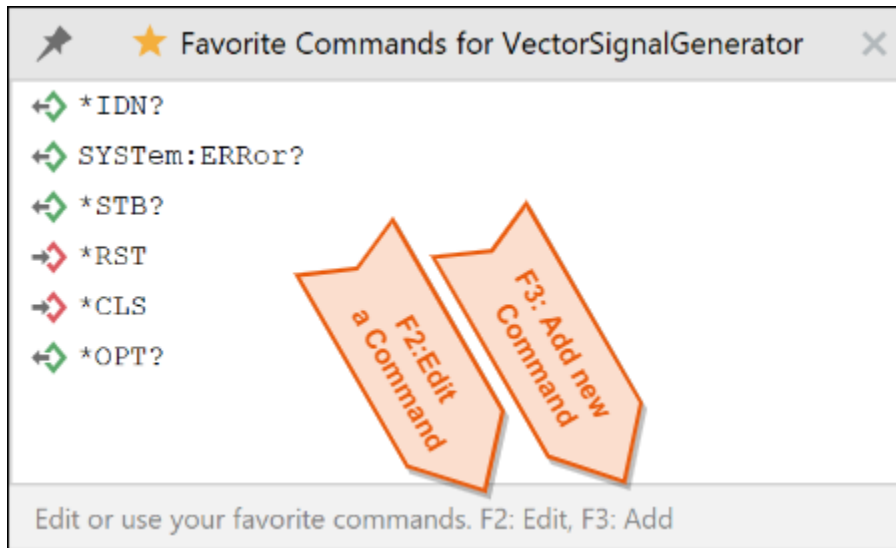


Full Version is invoked with **DOWN**, or clicking on **Commands History** button (6). The last command sent is at the top. Full version allows you to edit the list with drag and drop or **DELETE** key. Pressing **ENTER** or double-clicking sends the command to the instrument. Dragging a command to you opened Python script inserts the call snippet. If you want to drag more than one command, use the **Right Mouse Button**. You can pin the window to prevent it from auto-closing:



## 7.4 Favorite Commands

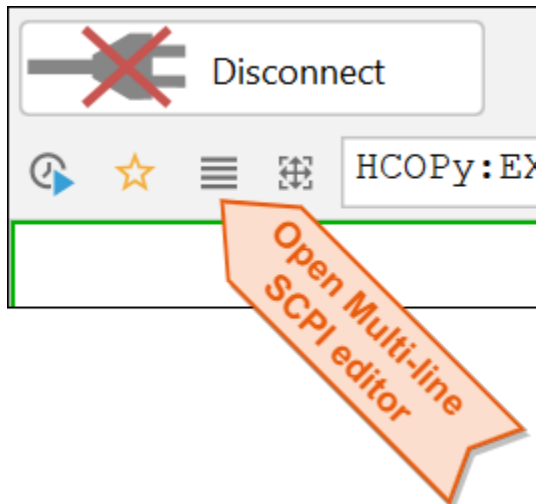
You can mark any command of the instrument as favorite to have a quick access to it by clicking the star icon on the communication panel (7). To open the Favorites Window, right-click the star icon. Same as for the Commands History, the Favorites are stored for the entire family of instruments. Here you also have an option to edit or manually add commands:



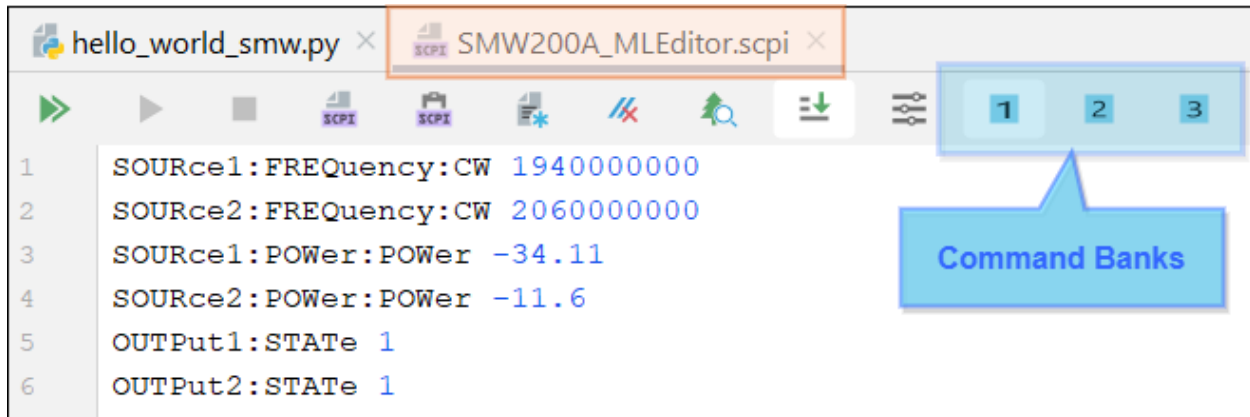
## 7.5 Multi-line SCPI Editor

Since Version 2.0.0, the Multi-line SCPI Editor is a standard Pycharm Text editor, but with a special top panel for execution and parse controls. It overcomes the limitation of a single-line SCPI Command field. The SCPI Editor is in detailed described in [13. Plain SCPI Scripts Editor](#).

Open the Multi-Line SCPI Editor:



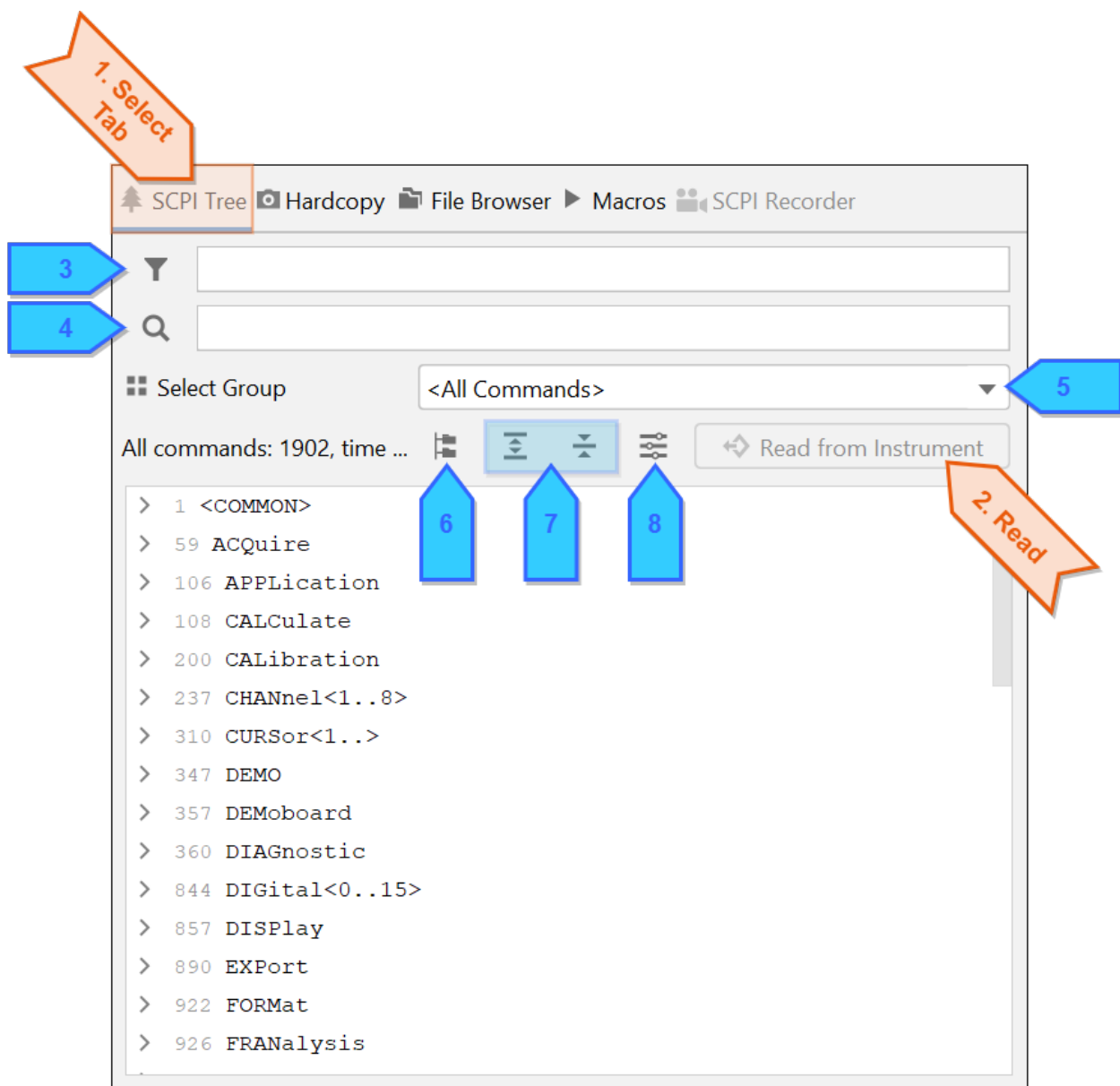
Multi-line SCPI Editor GUI:



The meaning of the **Command Banks** is also described in the chapter *Plain SCPI Editor Flavors*.

## 7. FUNCTION PANEL - SCPI TREE

SCPI Tree is a tab of the Instrument Function Panel allowing you to access all the instrument's supported SCPI commands.



Description of the controls:

1. **Function Panel SCPI Tree Tab** - select this tab to access SCPI Tree features.
2. **Read Tree from Instrument** - button, that requires active connection (see SCPI Communicator). Use it to read the SCPI Tree from the instrument. You only need to do it once. After that, the SCPI Tree is cached in your computer.
3. **Filter Controls** - Filter Toggle button switches the filtering of the tree ON/OFF. Filtering uses a powerful SCPI-syntax tailored engine to filter commands you are searching for. See the control's tooltip for more details on the syntax.
4. **Search Controls** - Search button and the Search Text allow for searching of next element fulfilling the search text. The syntax is the same as for the Filter Control.
5. **Select SCPI Group** - optional list-box, only visible when your instrument supports reading of the SCPI commands per Firmware Application (Group).
6. **Expand/Collapse all Tree Branches** - expands or collapses all the SCPI Tree branches. Use the context-menu to expand/collapse only certain branch, or certain level.
7. **Flat Tree Toggle (F2)** - toggle the display between tree and flat list.
8. **Foldable Settings Panel** - open pupup window with settings related to look-and-feel of the SCPI Tree branches and commands.

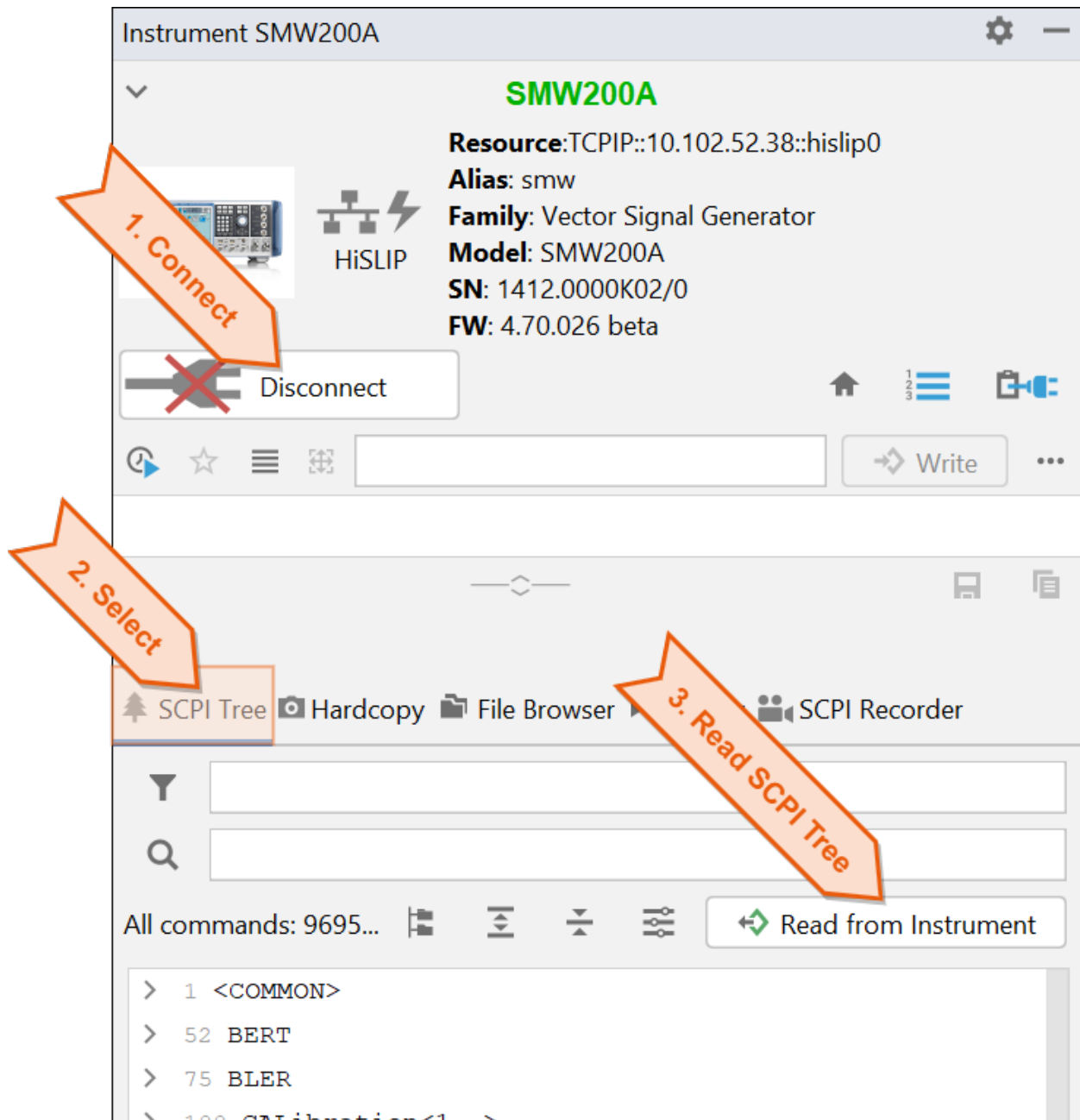
## 8.1 7.1 Reading the SCPI Tree

Once you read the instrument's SCPI Tree, you can use all the features related to SCPI auto-completion and SCPI search operations. You only have to do this once. After that, the SCPI Tree is cached in your computer. Keep in mind, that the cached tree is related to a specific instrument's firmware version. After a firmware update, you need to read it again.

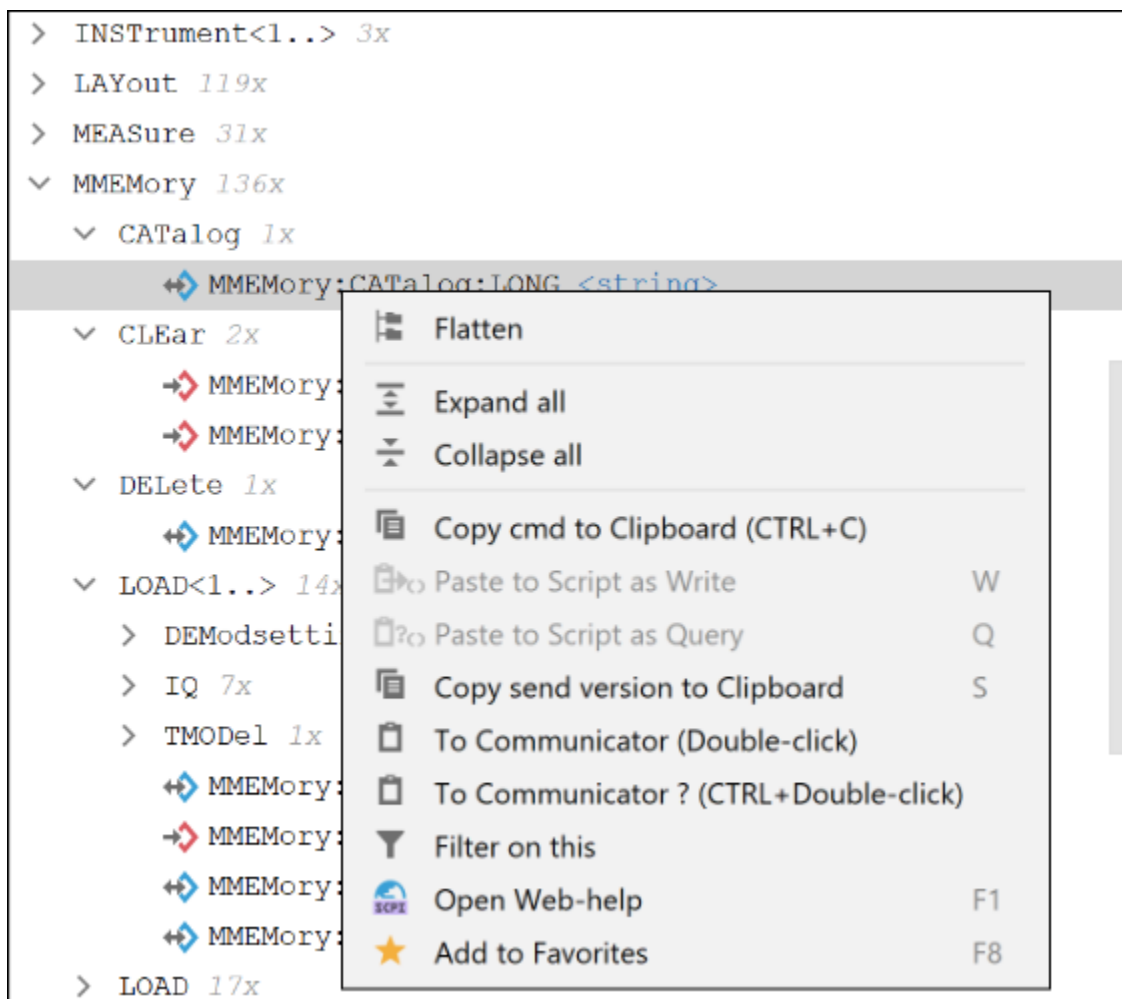
Follow the operations in order shown on the picture below:

1. **Connect** to your instrument
2. **Select** the SCPI Tree Function Panel tab
3. **Read** the SCPI Tree from the instrument





Each SCPI Tree item has a dedicated menu with many useful actions. Example for a single SCPI Command node:



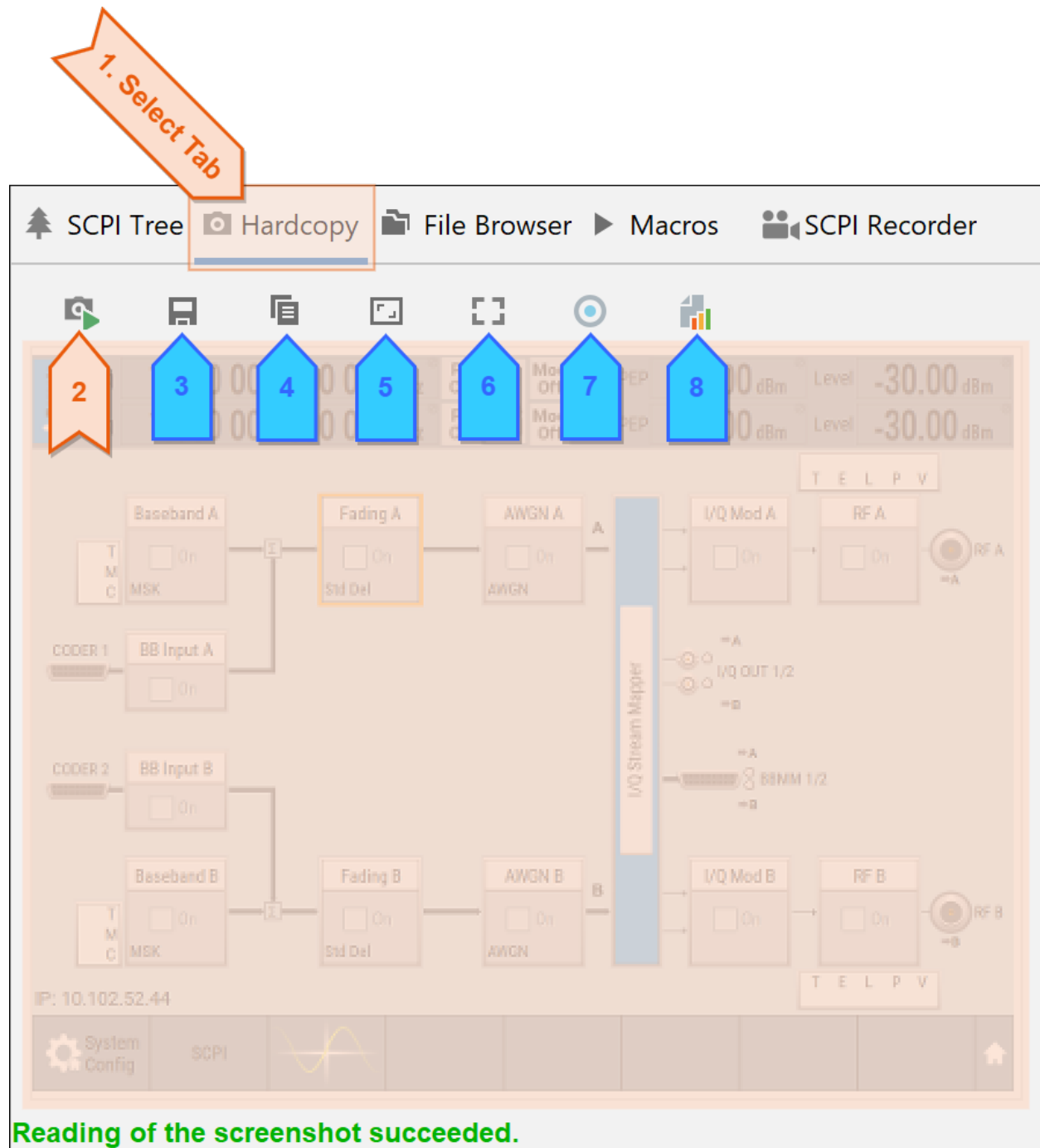
**Tip:** Check the tooltip texts for the Filter and Search text fields, they show syntax and examples for the match expressions: The filter/search text is a combination of space-separated tokens with logical AND between them:

- Search for a SCPI part that must follow in this very order: CONF:FREQ
- Search for SCPI parts regardless of the order: CONF FREQ
- Search for SCPI parts, anchored at the beginning and at the end: :CONF FREQ;
- Search for a parameter: P>EXT
- Search for a SCPI with a parameter: ROSScillator P>EXT
- Search full text: F>quency

**Tip:** Use drag&drop on the SCPI Tree Command nodes to paste write or query snippets to your Python script. The drag&drop also works for SCPI editors.

## **8. FUNCTION PANEL - HARDCOPY**

Hardcopy is a function that captures content of the instrument's screen. Our plugin offers that with one click (OK, it could be up to 3 clicks :-).



Description of the controls:

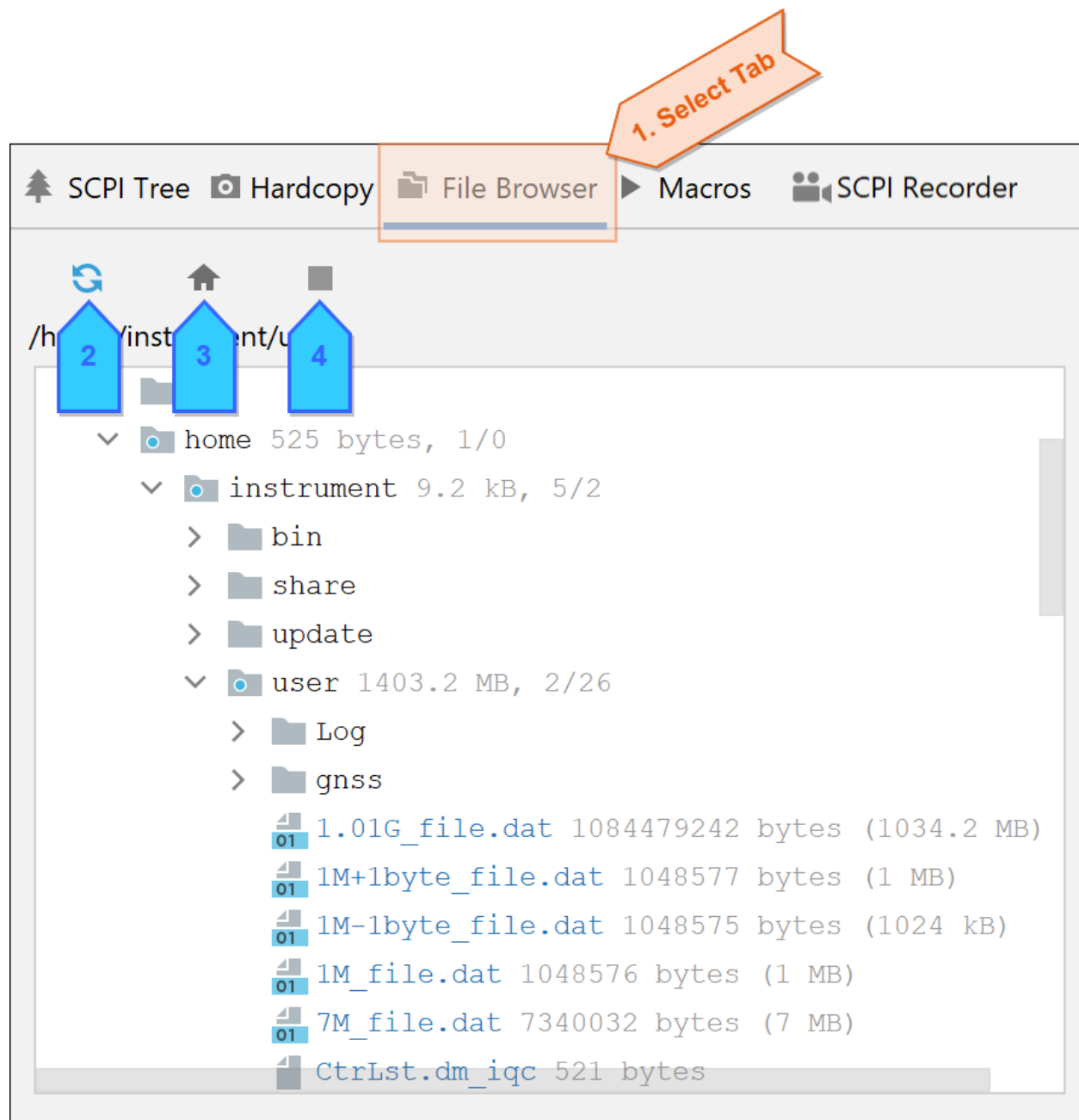
1. **Function Panel Hardcopy Tab** - select this tab to access Hardcopy features.
2. **Make Screenshot Button** - requires active connection (see SCPI Communicator). If connected, this button starts the screenshot acquisition and transfer to your computer.
3. **Save to File Button** - available after the successful screenshot acquisition, allows saving the picture in **jpg** or **png** format and original size.
4. **Copy to Clipboard Button** - copies the file to Clipboard in original size.

5. **Fit to View** - automatically performed after the successful screenshot acquisition. Adjust the picture size to fit the view with limited width or height. Does the zoom > 1:1 too.
6. **Show in Original Size** - changes the picture back to the original size and activates the view pane's scroll controls if necessary.
7. **Go Automatically to Local Afterwards** - after the screenshot is done, the instrument goes to local.
8. **Auto File Naming** - if checked, you do not need to enter new name by each "Save", you only enter it once, and the further ones are then auto-named with name suffix indexes \_001, \_002, \_003 and so on...



## **9. FUNCTION PANEL - FILE BROWSER**

File browser is a handy feature that allows you to seamlessly copy files between your computer and your instrument, or even between two of your instruments. The usage is just like your standard file explorer - copy / paste with CTRL+C / CTRL+V, double-click to open the file in Pycharm. The File Browser can also manage the instrument file system - copying, renaming, deleting, moving of files, creating folders... Check out the items context-menu to see which features are available for which item or group of items.

















Description of the controls:

1. **Function Panel File Browser** - select this tab to access File Browser features.
2. **Complete Refresh of the File Structure** - start from the beginning, read fresh file tree from the instrument.
3. **Go to Home Folder** - navigates through the file structure to your home folder. You can change the home folder by right-clicking on the button.
4. **Stop Running Operations** - sometimes reading of folder consist of many small tasks that can take a long time. You can interrupt them with this button.

Use the context-menu of the items, they offer many features. Below is an example of the context-menu for a single file:



	Set Parent Folder as Home	
	Read and Open the File (Double-click)	
	Copy File to PC Clipboard (CTRL+C)	
	Instrument operations	
	Rename File ...	F2
	Duplicate File ...	
	Delete File	Delete
	Create Folder ...	F7
	Copy File Reference	
	Cut File Reference	
	Copy File Name to Clipboard	F
	Copy File Path to Clipboard	D
	Paste to Script as Read File from Instrument	R
	Paste to Script as Write File to Instrument	W



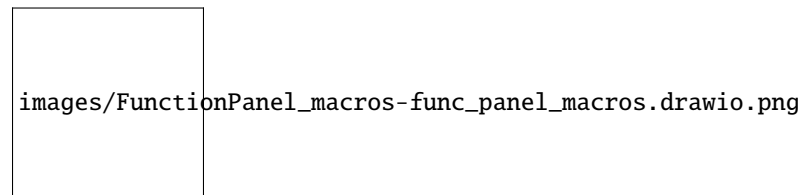
## 10. FUNCTION PANEL - MACROS

Macros Panel provides quick way of performing custom actions with your instruments, for example, reading instrument options, custom screenshots, special preset you need to execute often, or copying certain data from the instrument repeatedly. You can define these actions as standard Python scripts, and they are performed in the Pycharm python console. A Macro in the context of the plugin, is a python script that contains a defined function (by default `execute`) with the following API:

```
execute(rsrc_name: str, opts: str, instr_name: str, instr_alias: str, mods: str, ↵  
↳ **kwargs)
```

Such macro can be reused by multiple instrument panels which then call the `execute` function with their specific instrument information. The script also gets the `mods` variable, that signals which modification keys you used when clicking the macro button - CTRL/ALT/SHIFT. We have some ready-to-use macros (see button **3. Add Default**) to get you started.

Let us look at the Function Panel:



Description of the controls:

1. **Function Panel Macros** - tab selector for the Macros.
2. **Add new Macro Button** - shows a dialog where you specify the macro's name (button's label), script file (if new, it will be created and opened in your Pycharm editor), the function name, and the kwargs (optional).
3. **Add Default Macros** - adds universal macros that you can use with any instrument (for example, read instrument options). Use these as a starting point to define your own macro scripts.
4. **Add from Existing Macros** - copy the existing macros from other instruments.
5. **Macro Buttons Area** - contains all the macros defined for your instrument. The context-menu allows you to open the folder containing the macro scripts. Each button has its own context-menu with related actions.

Use the button **3 - Add Default Macros** and try out the macro **Get All Options**. When you start it, the script `show_instrument_options.py` executes initialization of your instrument, sends `*OPT?` query and splits the response into B- and K- options. Right-click on the **Get All Options** button and select **Open Macro Script** to see how the script looks like:

```
"""Shows the instrument options split to K- and B- types.  
The options do not contain duplicates and are alphabetically sorted."""
```

(continues on next page)

(continued from previous page)

```

from RsInstrument import *

def execute(rsrc_name: str, opts: str, instr_name: str, instr_alias: str, mods: str,
↳ **kwargs) -> None:
    """This function is executed with the parameters coming from the invoking Instrument
↳ Tool Window."""
    io = None
    try:
        io = RsInstrument(resource_name=rsrc_name, id_query=True, reset=False,
↳ options=opts)
        print('Instrument: ' + io.idn_string)
        # -----
        # Insert your custom code here...
        # -----
        k_opts = filter(lambda opt: (opt.upper().startswith('K')), io.instrument_options)
        print("K-Options: " + ', '.join(k_opts))

        b_opts = filter(lambda opt: (opt.upper().startswith('B')), io.instrument_options)
        print("B-Options: " + ', '.join(b_opts))

    finally:
        if io is not None:
            io.close()

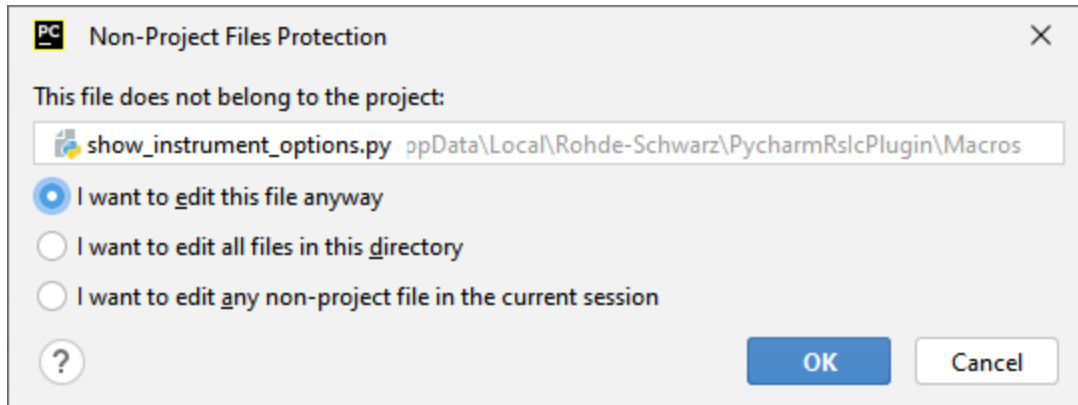
if __name__ == '__main__':
    # This section helps you to debug the macro - you can run it as a script.
    # It will not be executed, when the script is invoked with a macro button.
    # Hash: -1921060725 Do not modify the hash.
    execute(
        rsrc_name='TCPIP::10.102.52.42::hislip0',
        opts='VisaTimeout = 3000',
        instr_name='SMW200A',
        instr_alias='smw',
        mods='')

```

The function `execute` receives the data from the instrument panel that invoked it. After that, you can do anything you like in the function body. In our case, we open the `io` session to our instrument, read the options and print them to the console.

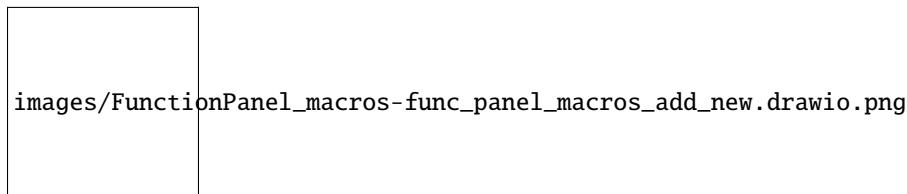
Notice the `__main__` guard at the bottom: it helps you to debug the script - you can run it as is. The parameters supplied are of the function panel from which you opened it, in our example an SMW200A instrument. The cool feature is, if you open the same macro script from another panel, for example an FSW, this part is updated to correspond the FSW settings. This way, if you debug the macro, you always have the correct instrument parameters.

**Hint:** The macros are stored to a folder which you can change in the *Settings -> Instrument Tool Windows*. If you want to edit a macro, you get a warning window from Pycharm, because the folder is not a part of your project. You can safely select the first or the second option:



Remember, to manually save the modified script, because Pycharm does not do it automatically for files outside your project.

We are going to create a new macro. Use the button **2 - Add New Macro**, fill out the Button Display Name, the Script Name and hit **OK**:



The RsIC creates a macro file for you with the entered name `my_first_macro.py`. Modify the section with the comment *Insert your custom code here...*:

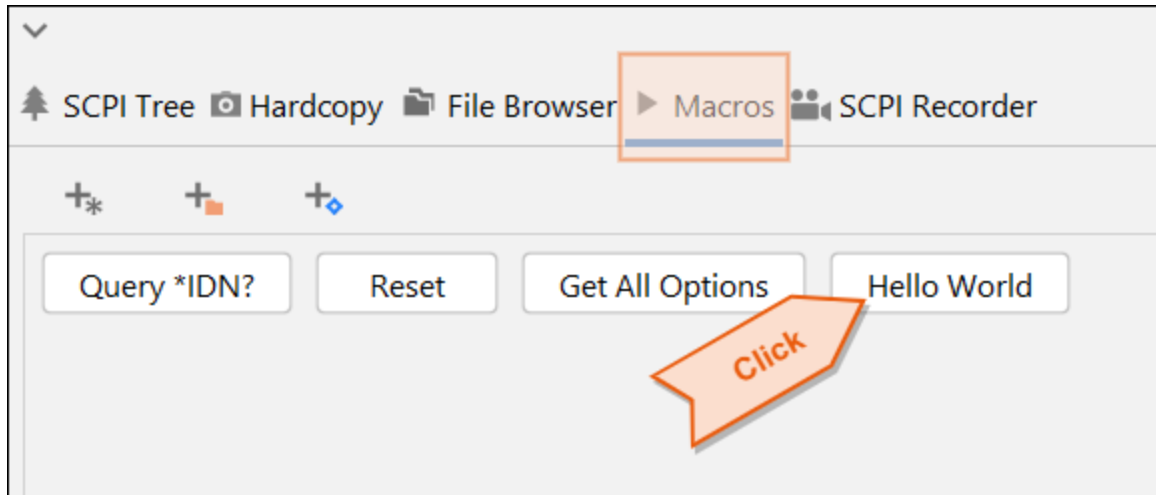
```
# -----
# Insert your custom code here...
# -----
```

to (remember to keep the same indent...):

```
print('Hello, I have entered my custom code here...')
# -----
# Insert your custom code here...
# -----
```

Uncomment the other lines, if you want to enter custom keyword arguments, or recognize CTRL / ALT / SHIFT modification when pressing the macro button. You can also use the prepared function `_get_popup_parameter()` to show a popup to the user to enter a string parameter.

After you have modified the macro, it is time to start it. Press the newly added macro button. By the first start, you might need to start a new Python Console:

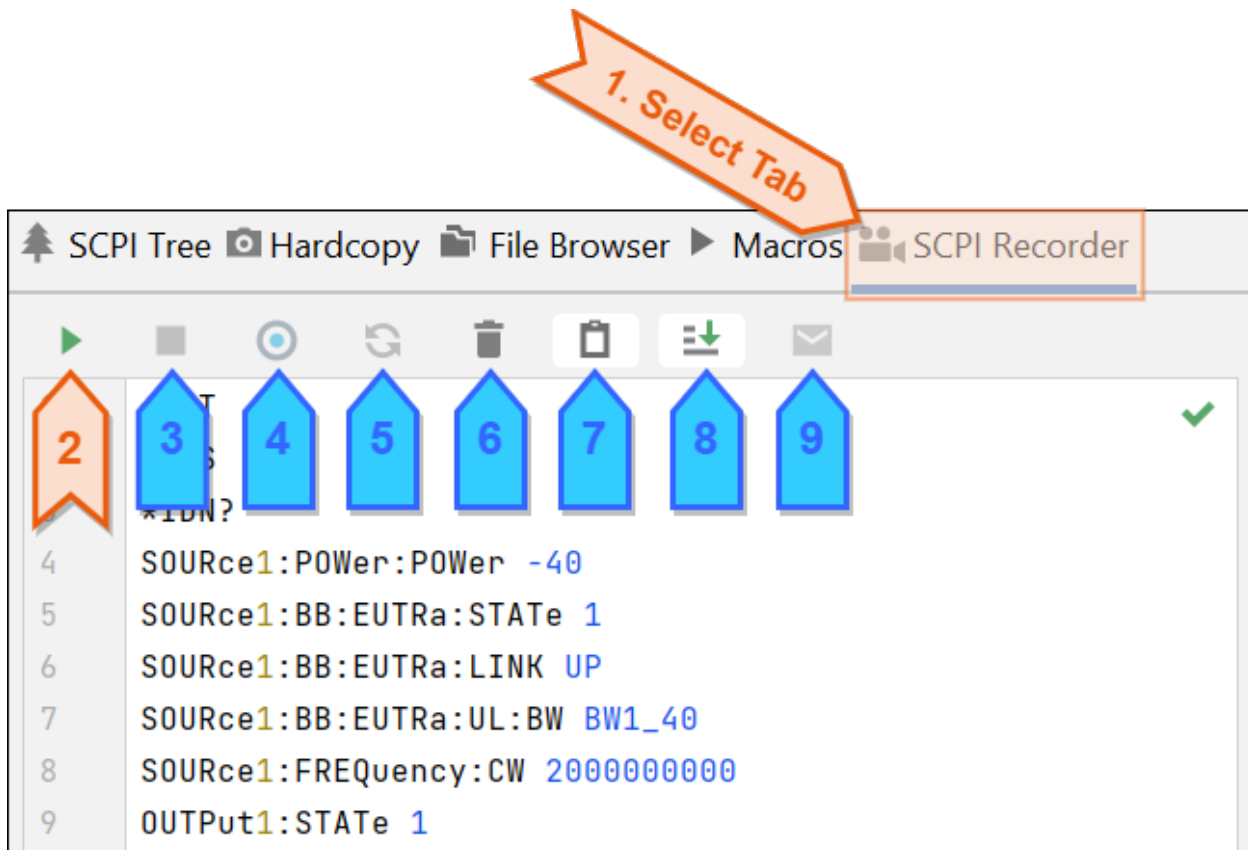


Your script is executed in the python console tool window:

```
...# ----- Starting new macro - instrument SMW200A
...if 'my_first_macro' not in dir(): import my_first_macro
...importlib.reload(my_first_macro2)
...my_first_macro.execute(rsrc_name='TCPIP::10.99.2.10::hislip0', opts='', instr_name=
'SMW200A', instr_alias='smw', mods='')
...:
Instrument identification: Rohde&Schwarz,SMW200A,1412.0000K02/0,5.10.035.29
Hello, I have entered my custom code here...
```

## 11. FUNCTION PANEL - SCPI RECORDER

SCPI Recorder offers the easiest way to convert manual operation of your instrument into remote-control SCPI calls. Simply put: you operate your instrument manually, and the corresponding SCPI commands appear in your script. Currently only our Hi-end Signal Analyzers and Generators offer this feature, but if you are a lucky user of such instrument(s), you are going to enjoy the heck out of it.



Description of the controls:

1. **Function Panel SCPI Recorder** - select this tab to access SCPI Recorder features.
2. **Start the SCPI Recorder** - starts recording of user interactions on the instrument. If the SCPI Recorder already contains some commands, you get a choice to keep or clear them.
3. **Stop the SCPI Recorder** - stop the SCPI recording.
4. **Go To Local** - orders your instrument to switch to local mode so you can operate it. This is already done after you start the SCPI Recorder, but if for any reason you need to do it again, use this button.

5. **Poll periodically** - if ON, the SCPI Recorder Panel continuously polls your instrument for new SCPI commands. You can stop it with this button and just perform single poll by right-clicking it or hitting **F5**.
6. **Delete commands** - deletes all the SCPI commands in your list and in the instrument's SCPI recorder list.
7. **Paste to your script as snippets** - if ON, recorded commands not only appear in your SCPI Recorder list, but also in your script as SCPI call snippets. You can always drag them there afterwards by grabbing the gutter.
8. **Autoscroll** - if ON, the window scrolls automatically when new commands are added.
9. **Poll signal** - lights up every time the SCPI Recorder Panel queries the instrument for new commands. Notice, that when there are no new commands coming, the polling becomes less frequent, and it is reset back to 1 second after new commands are added.

---

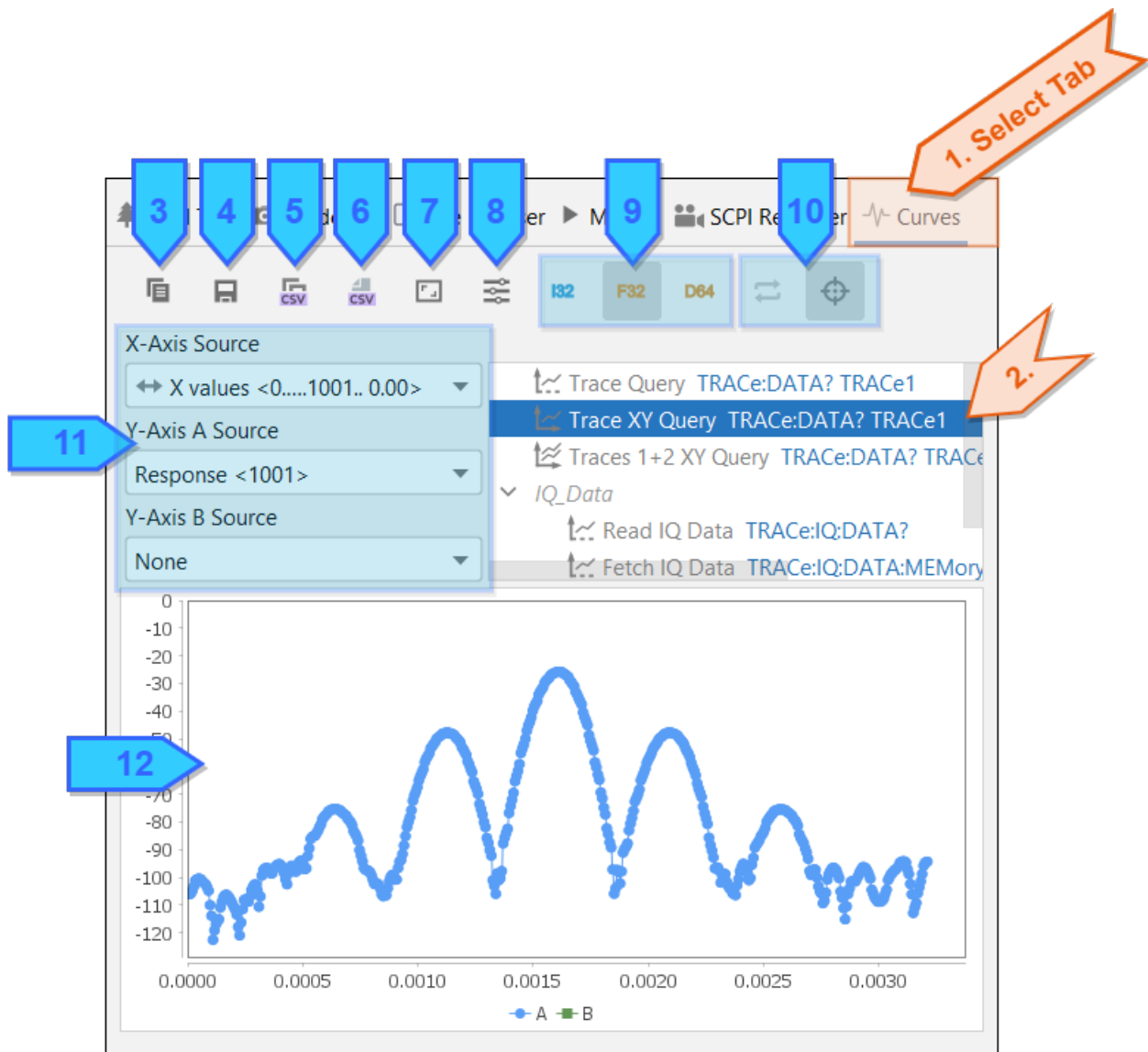
**Hint:** The text editor here is a standard *SCPI Editor* without the controls strip. It also does not offer executing or parsing the commands. For that, you have to drag them to another SCPI Editor.

---



## 12. FUNCTION PANEL - CURVES

Curves provide quick way of querying traces and waveforms from your instrument. You can use the pre-defined queries, adjust them, or define your own. Any response from the instrument, that can be interpreted as an array of numbers, can be displayed in the 'Curves' XY Chart. You can save/copy them as pictures, or csv-text data. Curves are also a great way for checking the proper format of the values received from binary data blocks.




Description of the controls:

1. **Function Panel Curves** - select this tab to access Curves Features. When selected, each instrument response is checked for array content and potentially displayed in the chart.
2. **Pre-defined queries** - connect to the instrument, and click on one of the lines to query the trace / waveform.
3. **Copy to clipboard as picture** - copy the picture to clipboard.
4. **Save to file as picture** - save the current chart content to a PNG file.
5. **Copy to clipboard as csv-text** - copy the data to clipboard as csv-text.
6. **Save to file as csv-text** - save the current chart content to a csv-text file.
7. **Keep the chart size constant** - if checked, the chart size does not change if you resize the Tool Window. This is helpful, if you are making set of screenshots where you want them to all have the same size. You can define this size in the **Curves Settings** dialog (see control 8).
8. **Open the curves settings** - settings for the chart and data. For more details, see [Curves Settings](#).
9. **Binary Data Numbers Type** - select the type of the received data. This only affects the binary responses, for ASCII responses this setting is ignored.
10. **Binary Data Endianness** - in binary data, endianness define which byte of the number comes as first. MSB (Most Significant byte first) or LSB (Least significant byte first). Here, you can define whether to swap the endianness, or detect it automatically. For ASCII data this setting is ignored.
11. **Chart Data Source Selector** - define, which data to display on which axis. X axis is shared by both Y-A-Data, and Y-B-Data.
12. **Displayed Chart Data** - Area for displaying the curve. You can zoom on the points, or scroll left/right. The chart also has right-click menu with different features.

## 13.1 12.1 Pre-defined Queries

The table with pre-defined queries allows you to script several operations, meaning several SCPI command / queries to retrieve the data you want. You can query only Y-data, and the X-data is simple 0-based or 1-based integer index, or you can query XY data in two or more queries. You can also query two Y-traces and feed them into Y-A-Data, Y-B-Data, for example two traces of the spectrum analyzer. To see what is possible, right-click on one of the pre-defined queries and choose **Edit**. We will do this for the **Trace XY Query** item:

 Edit the Curve Query for the FSW-26 ✕

---

Name / Path

---

Query X-Data

Query X-Range Start

Query X-Range End

Query X-Range Start .. Stop

X-Data Source

---

Query Y-Data

Query Y-Data 2

Y-Data A Source

Y-Data B Source

---

Command at the Start

Command at the End

---

Format and Endianness ☒ ☐ I32 ☒ F32 ☐ D64 ☒ ☐ ↔ ☐ ☐

The command fields all have the SCPI code-completion option. Use the CTRL+Space to invoke the helps. Check out

the tooltips to see what are the function of the different fields.

---

**Note:**

- ;\*OPC suffix helps the query to finish immediately, if there was an error executing it. That speeds up the whole process in case of an error.
  - When you execute commands from the SCPI Communicator, you can only affect the response equivalent to the **Query Y-Data**. In order to be able to do more complex queries in a sequence, you have to define a Curve Query.
-

## 13. PLAIN SCPI SCRIPTS EDITOR

Before you continue with this chapter, *read out the SCPI Tree*. That allows you to use SCPI auto-completion and SCPI search in your SCPI script.

Since version 2.0.0, the RsIC has the option to create SCPI script files. A SCPI script file is a file with the extension `.scpi`. The content of the file are lines with plain SCPI commands, plus some extra features:

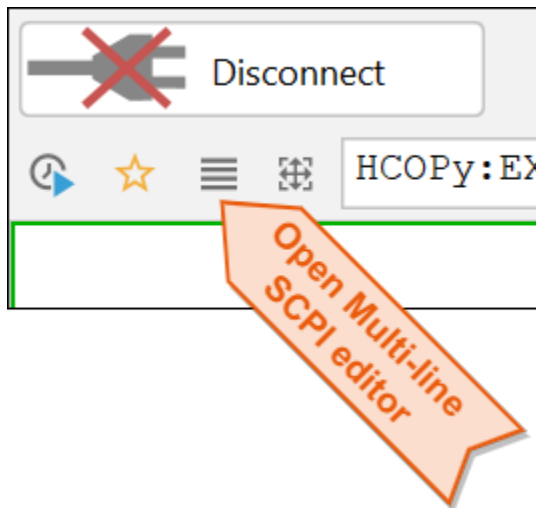
- full-line comments - lines starting with `//'` or `#` or `!`.
- comments after SCPI commands with the same syntax as the full-line comments.
- definition of a default target instrument - the instrument that all the commands are sent to.
- explicit definition of another target instrument for a command. This way, the SCPI script can send commands to **multiple instruments**.

---

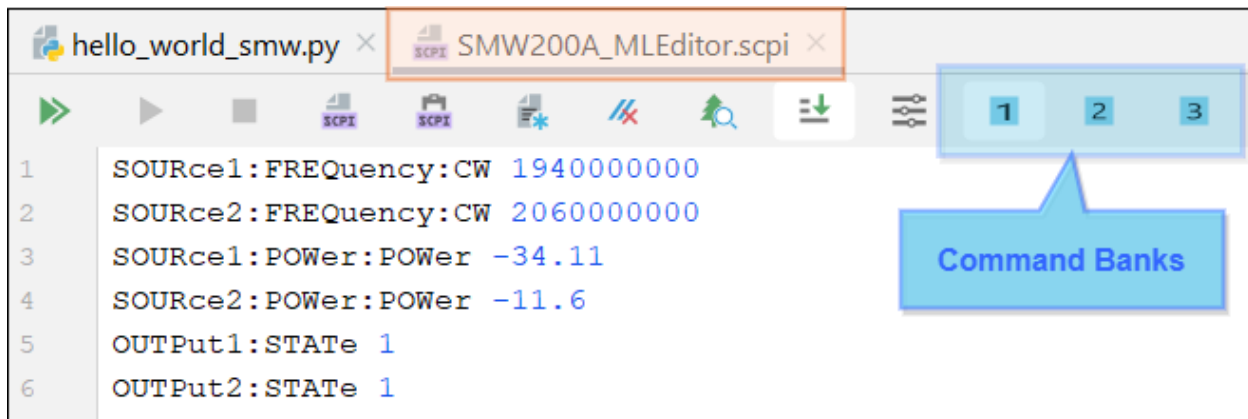
**Hint:** SCPI scripts are **not transferrable** - they can be executed only in Pycharm with the RsIC installed. But they serve as a good debugging tool before you create a code in your python scripts. Even if you do not plan to write any python scripts, the SCPI editor can help you debug you SCPI sequence. At the end, with the correct *snippet templates*, you can per drag&drop export this sequence as any programming language code. See *Writing Python Scripts per Drag & Drop*

---

The easiest way to start with the multiline \*.scpi files is to open them from the ITW (Instrument Tool Window):



Multi-line SCPI Editor GUI:

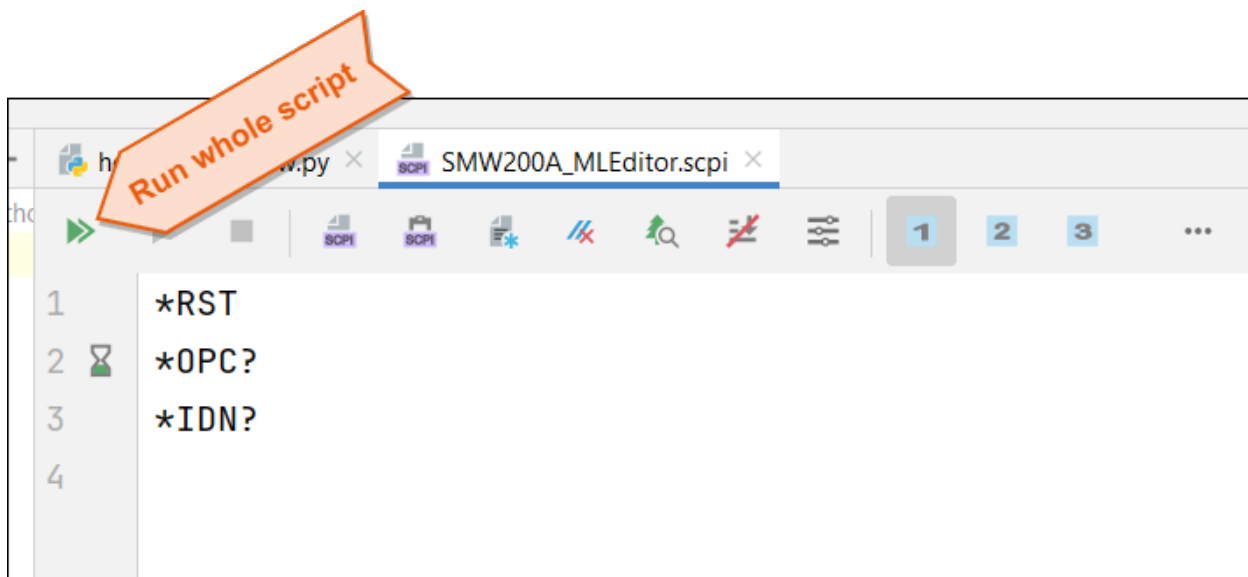


This SCPI editor is bound to the instrument's ITW. This means, the default instrument (see further for how you can control multiple instruments from one scpi script) is always the ITW's instrument, in our case the SMW200A. The file `SMW200A_MLEditor.scpi` is not saved separately in your local file system, but directly in the plugin data for the instrument. If you export/import plugin settings, the content of this file persists. You can have 3 different SCPI sequences (banks) under one instrument. Let us create a small 3-lines script and execute it:

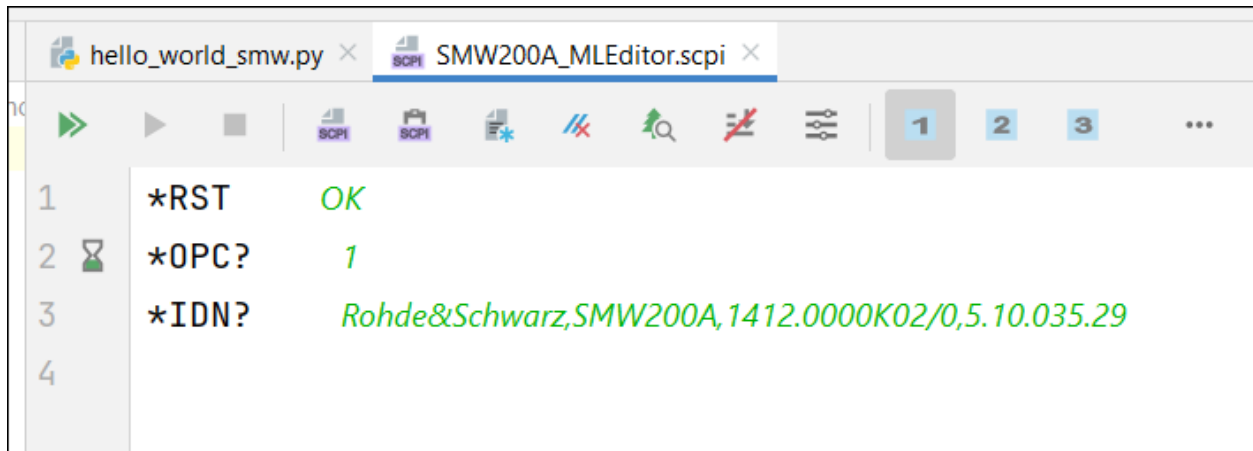
```

*RST
*OPC?
*IDN?

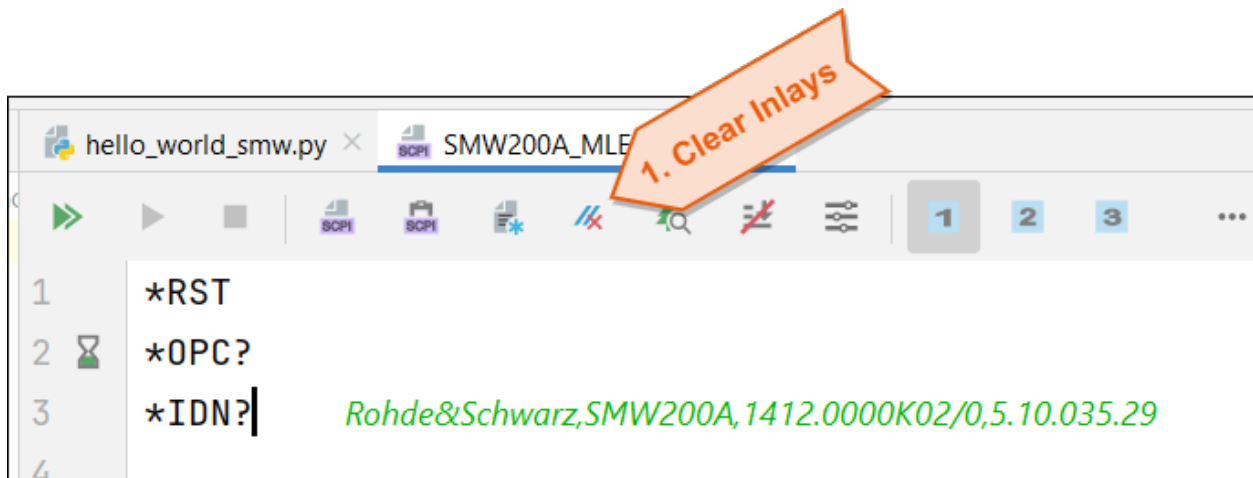
```



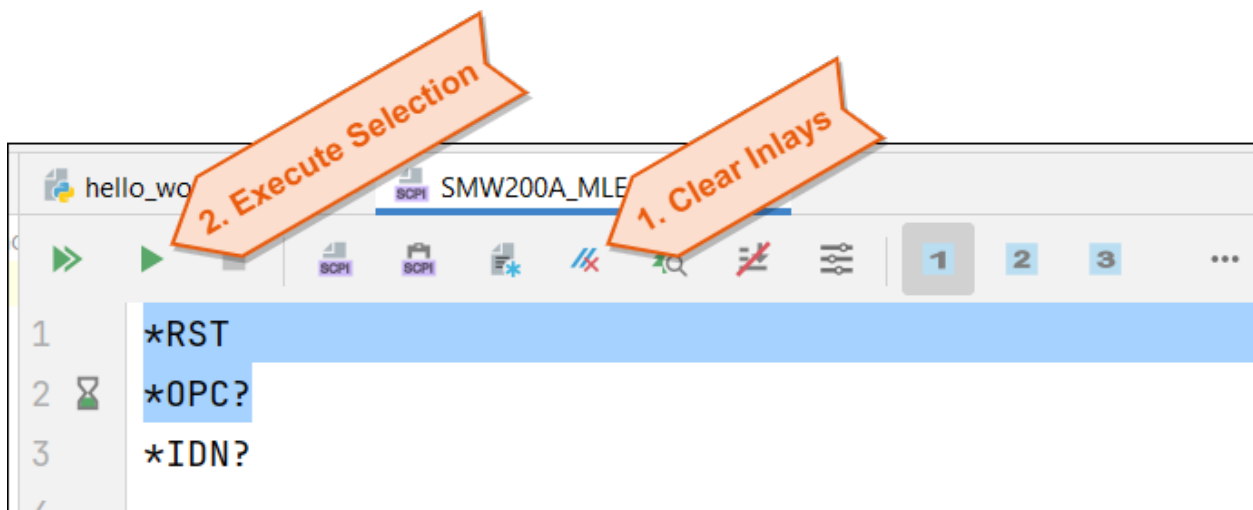
Execute the script. Remember, you need the instrument to be present and connected. You can see the execution results and responses for queries in a form of inlay text:



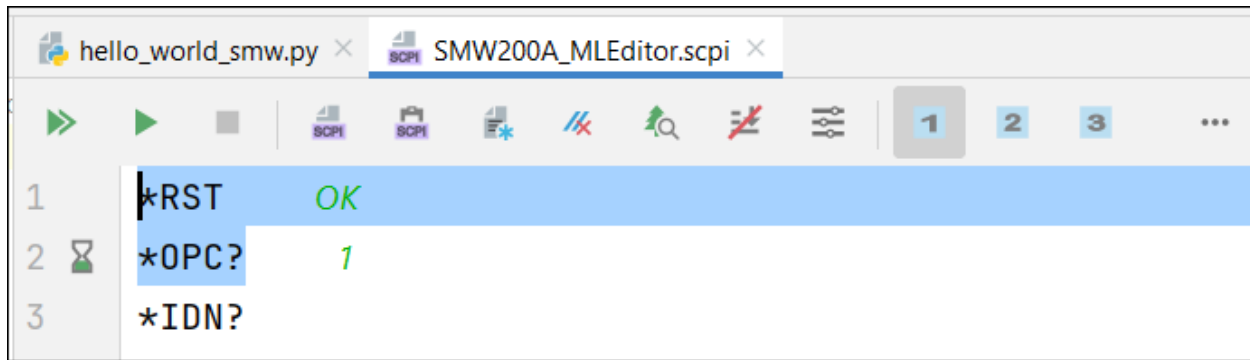
The big advantage compared to the single line communicator in the ITW is, you can execute either the entire script, or just a part of it, or just one line. Clear the inlays, go with the caret to the line 3, and press F4 (Execute line):



Only the **\*IDN?** was executed. Now select the first two lines and press the single green arrow:

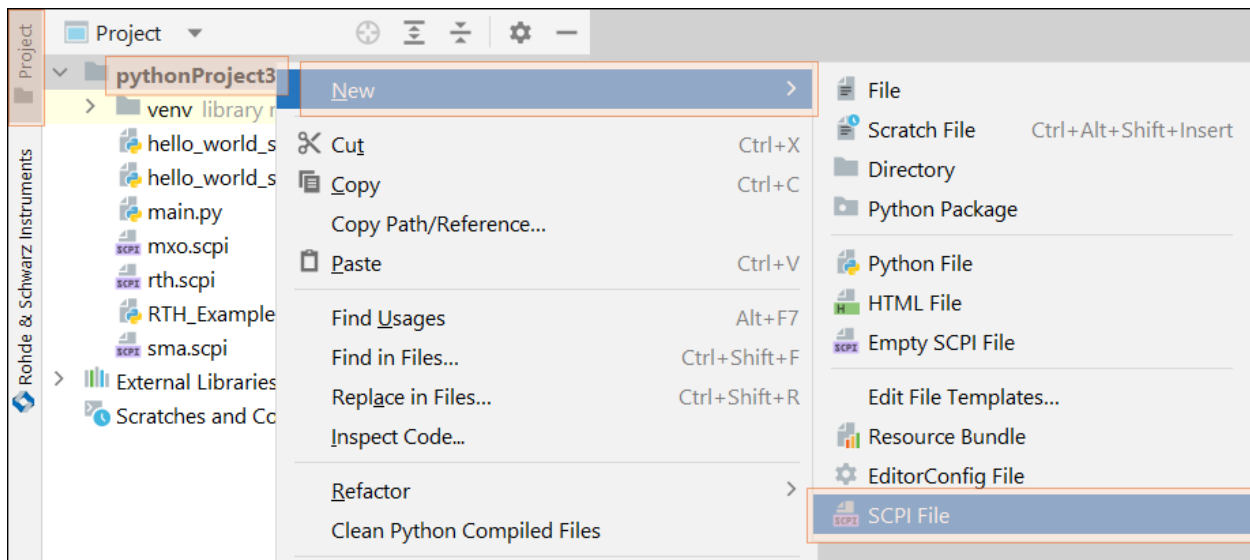


You see only the first two lines were executed:



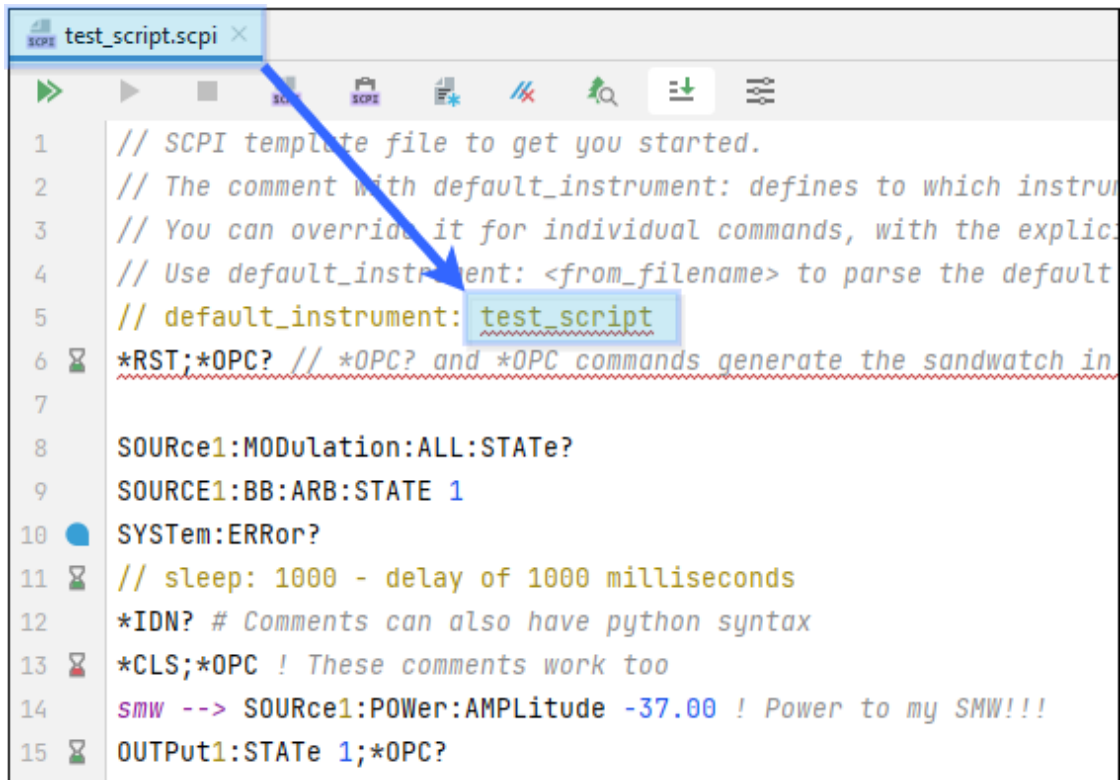
Right-click gives you even more options, for example Run-to, Run-from... See the descriptions for all the editor controls in the chapters below.

Multiline editor from the ITW is not the only type of the SCPI file. You can create your own generic SCPI file not bound to any specific instrument: Go to your Project tab, right-click on the project name, and select **New -> SCPI File**:



Type the name of your script without an extension and press ENTER. Pycharm creates a SCPI script with basic structure. Notice, that the comment with `default_instrument` has a special meaning: it defines the default target instrument (instrument, to which all the SCPI commands in the script are sent). The template sets this value to the name of your file:





The valid default\_instrument value is either instrument name, or its alias. Hover over the text and choose the quick action **Select a valid instrument**.

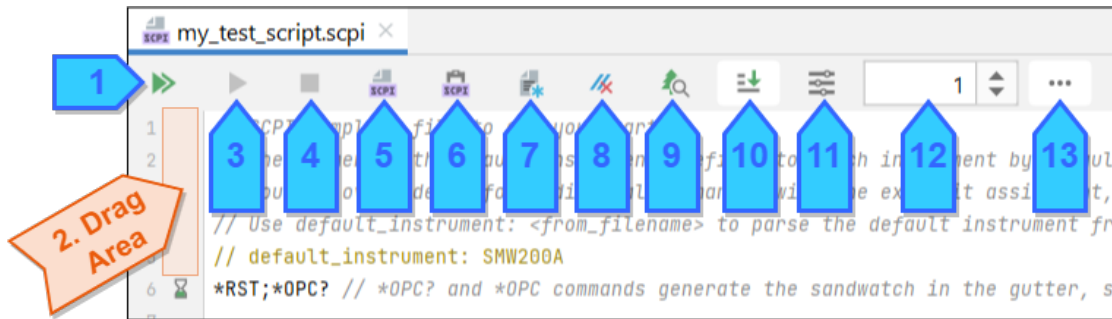
**Note:** If the comment with default instrument is missing, the RsIC tries to get the default instrument from the name of the file. The file name has to start either with the name or alias of the target instrument, all case-insensitive. Examples:

- File name smw\_test\_script.scpi will recognise smw if you have it defined as alias or name (case-insensitive).
- File name smw200a\_test\_script.scpi will recognise smw if you have it defined as alias or name (case-insensitive).
- File name blablabla.scpi will report an error when you try to execute the script. Unless, of course you have the instrument with the name blablabla :-)

Another special comment is on the **Line 11**: `// sleep: 1000`. This causes the execution engine to pause for 1000 milliseconds. Use it when you need to enter a pause into your execution chain.

Notice the syntax of the **Line 14**: `smw --> SOURCE1:POWer:AMPLitude -37.00`. It explicitly says that this command should be sent to the instrument with the name or alias smw. If the default instrument is, for example defined as fsw, this allows the script to work with two different instruments. If the explicit target instrument is not recognised, you will get a quick action to correct it.

Let us have a look at the controls:



Description of the controls:

1. **Execute all valid commands (F5)** - executes all the commands in the script, that are valid. Blank lines, comments and invalid commands are skipped. To execute a single line, use the shortcut **F4**
2. **Drag Area** - drag one or more lines (selection) to another SCPI or Python script or your SCPI communicator. Based on the drop target context, the RsIC choses or offers different paste formats.
3. **Execute selected commands** - executes all the commands in the selection.
4. **Stop running actions** - stops all the actions.
5. **Parse SCPI from file** - select a file to parse SCPI commands from. This feature uses advanced procedures to recognize the file format and parse the relevant SCPI commands out of it. Try it on IO Traces or different scripts, like for example Python, MATLAB, or IECWin.
6. **Parse SCPI from Clipboard** - same as the parsing from file, but the source is a text content of the Clipboard.
7. **Advanced Parser Window** - opens the *Advanced SCPI Parser*.
8. **Clear all inlays** - clears all the visible inlays that are displayed when executing script or checking for SCPI commands.
9. **Check in SCPI Tree** - checks, whether the commands are present in the instrument's SCPI Tree.
10. **Autoscroll** - if ON, the window automatically scrolls by execution or parsing.
11. **SCPI Editor Settings** - opens the *Settings for the SCPI Editors*.
12. **Run Cycles** - here you can define how many cycles should be executed. Set it to higher number then 1, if you want to repeat the selected sequence, one line, or the entire script.
13. **Show more / less** - shows/hides additional controls.

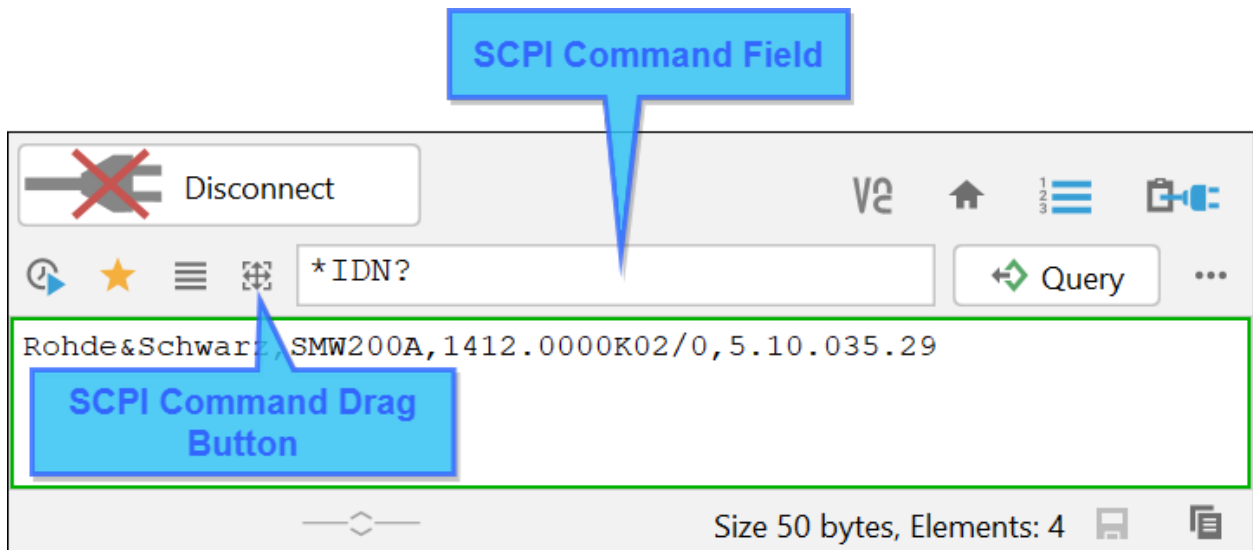
---

**Note:** By splitting/moving editors, you can have a situation where one SCPI file is opened in multiple editors. In such case, the executing controls are only available in one of them.

---

## 14.1 13.1 Writing scripts - SCPI Communicator drag function

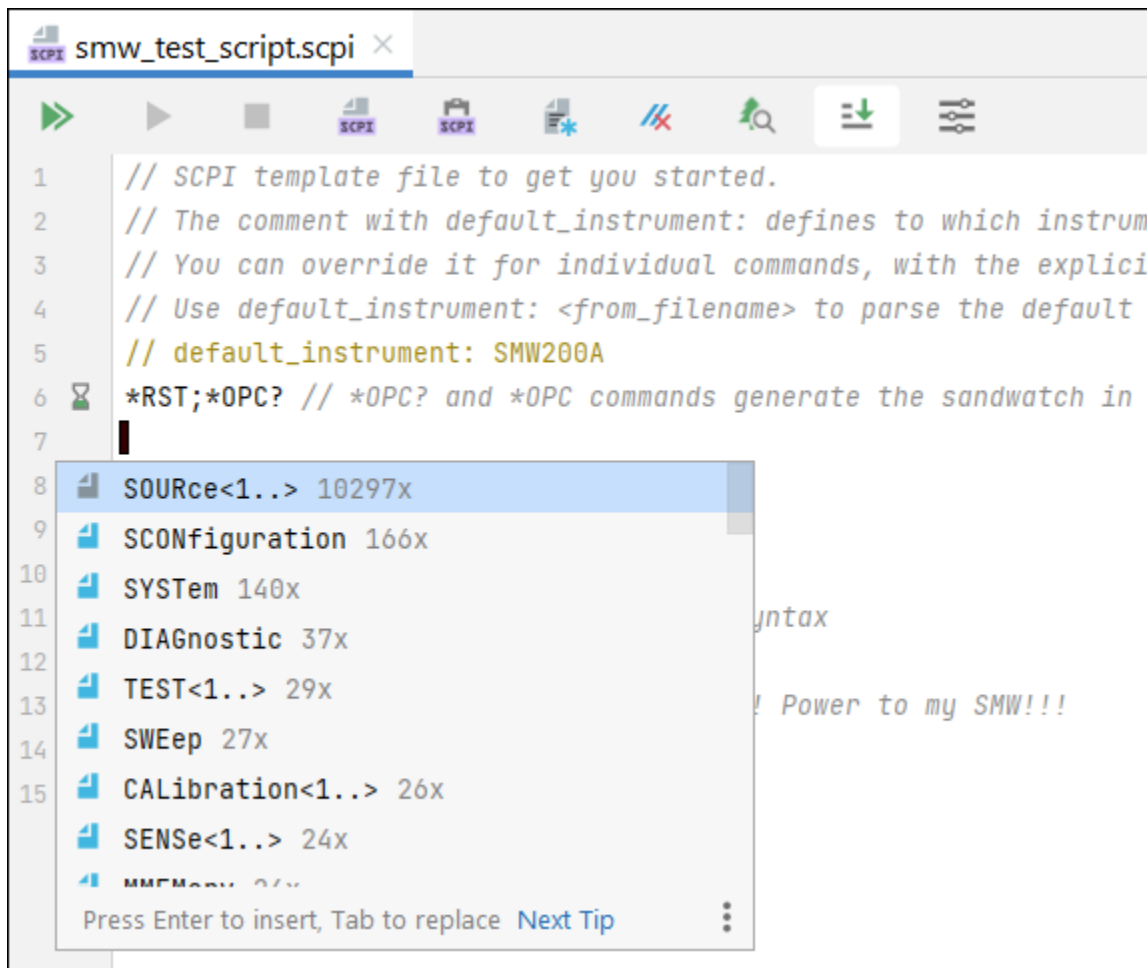
Once again, the important controls of the SCPI Communicator for the purpose of this chapter:



Enter the \*IDN? to the SCPI Command Field, and use the **SCPI Command Drag Button** to drag the command to the desired place in your script.

## 14.2 13.2 Writing scripts - SCPI Auto-completion

In the SCPI Editor, Press **CTRL+Space** to invoke the auto-complete suggestions:



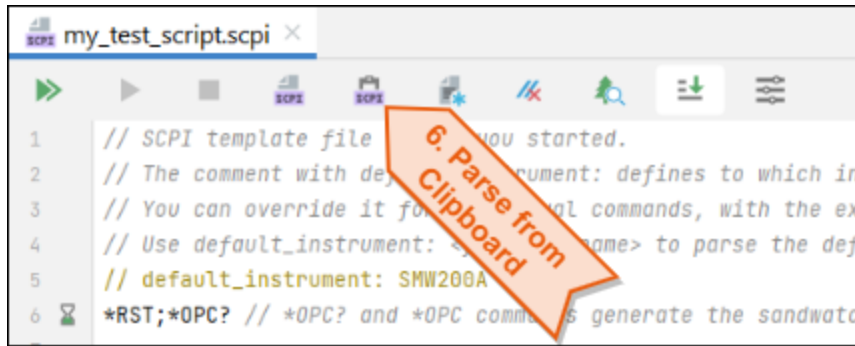
Select your desired command header and use **TAB** to insert it. Pycharm shows you another header(s) of the command to insert.

## 14.3 13.3 Writing scripts - Parsing from other formats

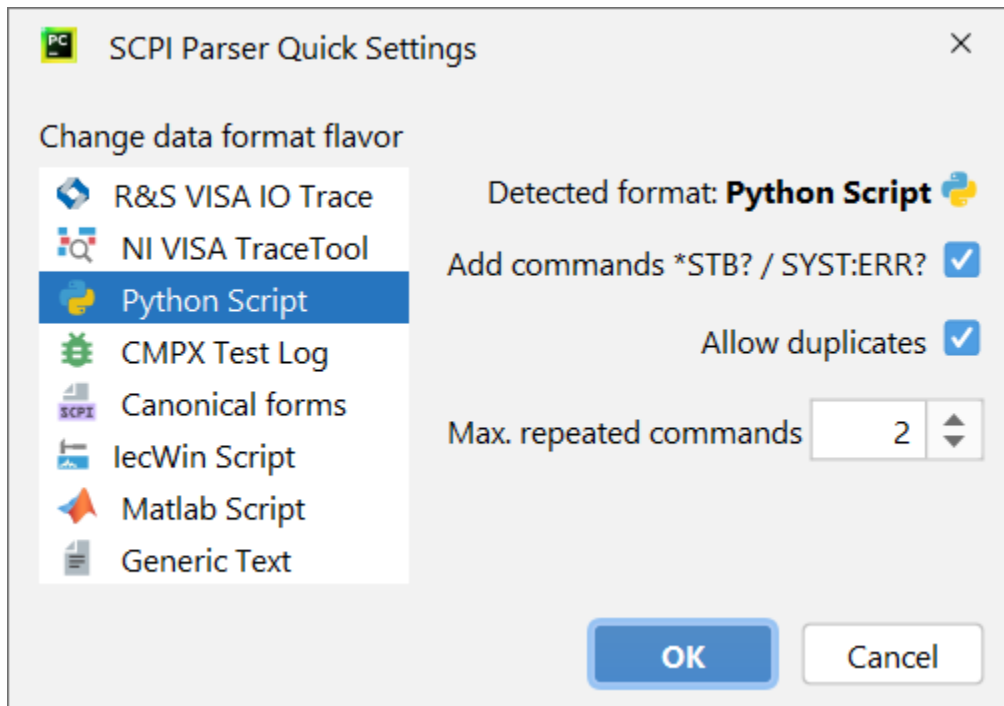
RsIC has a powerful parser that can take almost any text format and extract plain SCPI commands out of it. You can use two sources for that - clipboard or file. Let us take an existing python code for our FSW. Copy the code below into your Clipboard:

```
fsw.write("*RST")
fsw.query("*IDN?")
fsw.write("INSTRUMENT:CREate:NEW IQ,'IQ 1'")
fsw.write("SENS:SWEeP:COUNT 6")
fsw.write("DISP:TRAC1:MODE BLANK")
fsw.write("INIT:CONT OFF")
fsw.write("INST:SEL 'Spectrum';*WAI")
fsw.write("SENSe:SWEeP:MODE ESpectrum")
fsw.write("SENSe:ESpectrum:PRESet:STANdard 'WCDMA/3GPP/DL/3GPP_DL.xml'")
fsw.write("SENS:SWEeP:COUNT 5")
```

Click the button **Parse SCPI from Clipboard**:

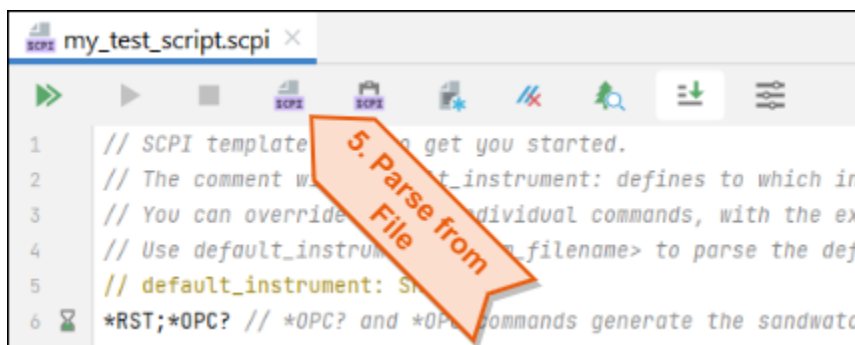


The following window pops up:



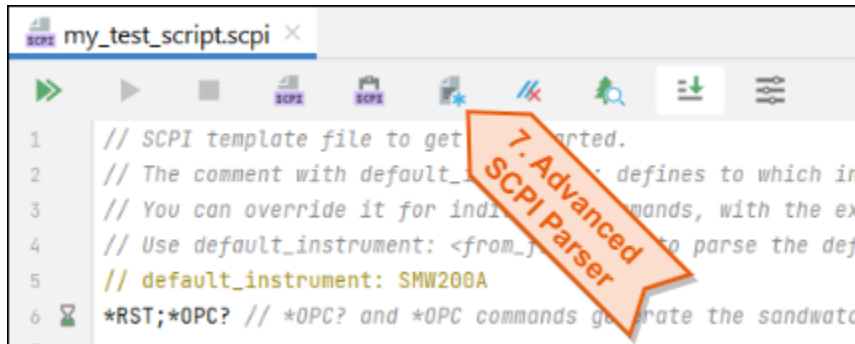
You can see the format was properly detected as a python script. In case the format auto-detection did not work, you can always change it. Press OK, and the parsed plain SCPI commands are added to your script.

Often you would like to reproduce the SCPI sequence that you have in a form of an **IO Trace** file. In this case, click the button **Parse from File**:



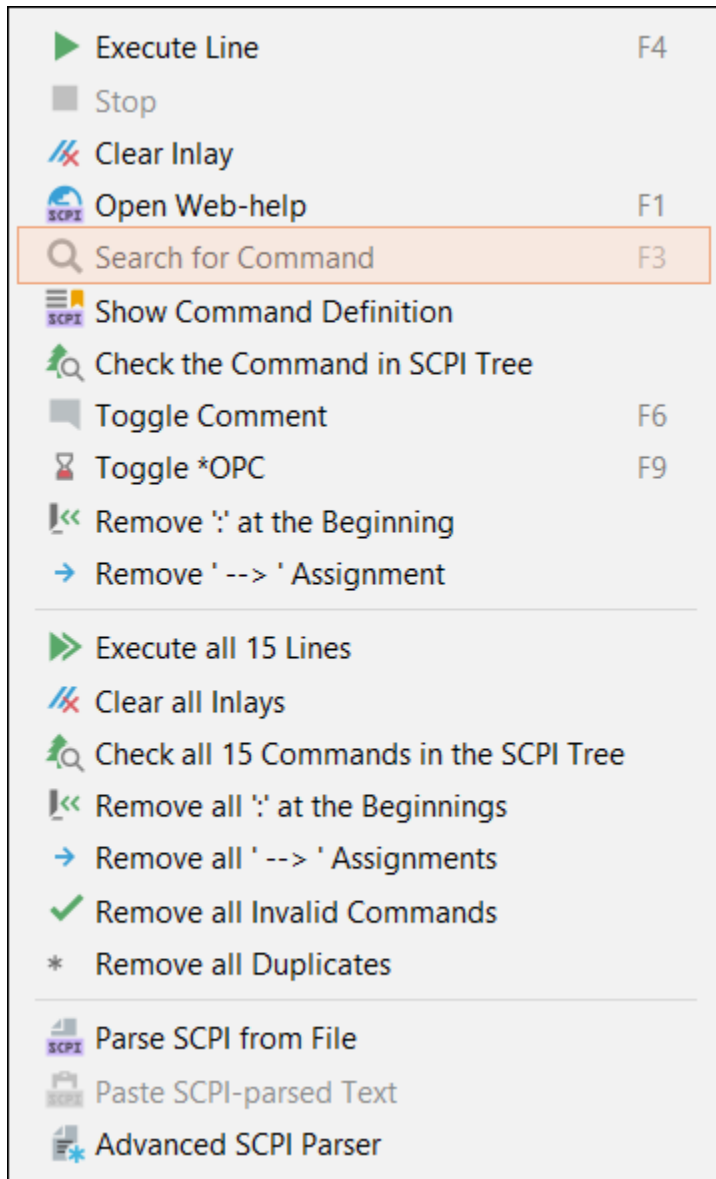
Select the file (try for example this one: `rsvisa_log_smw.log`), and the parser does the rest. Supported IO Traces are Rohde&Schwarz IO Trace and NI Trace Log exported as \*.txt file.

For IO Traces, it is useful to have an option to filter out only commands for one or more specific instruments. The *Advanced SCPI Parser* does exactly that:

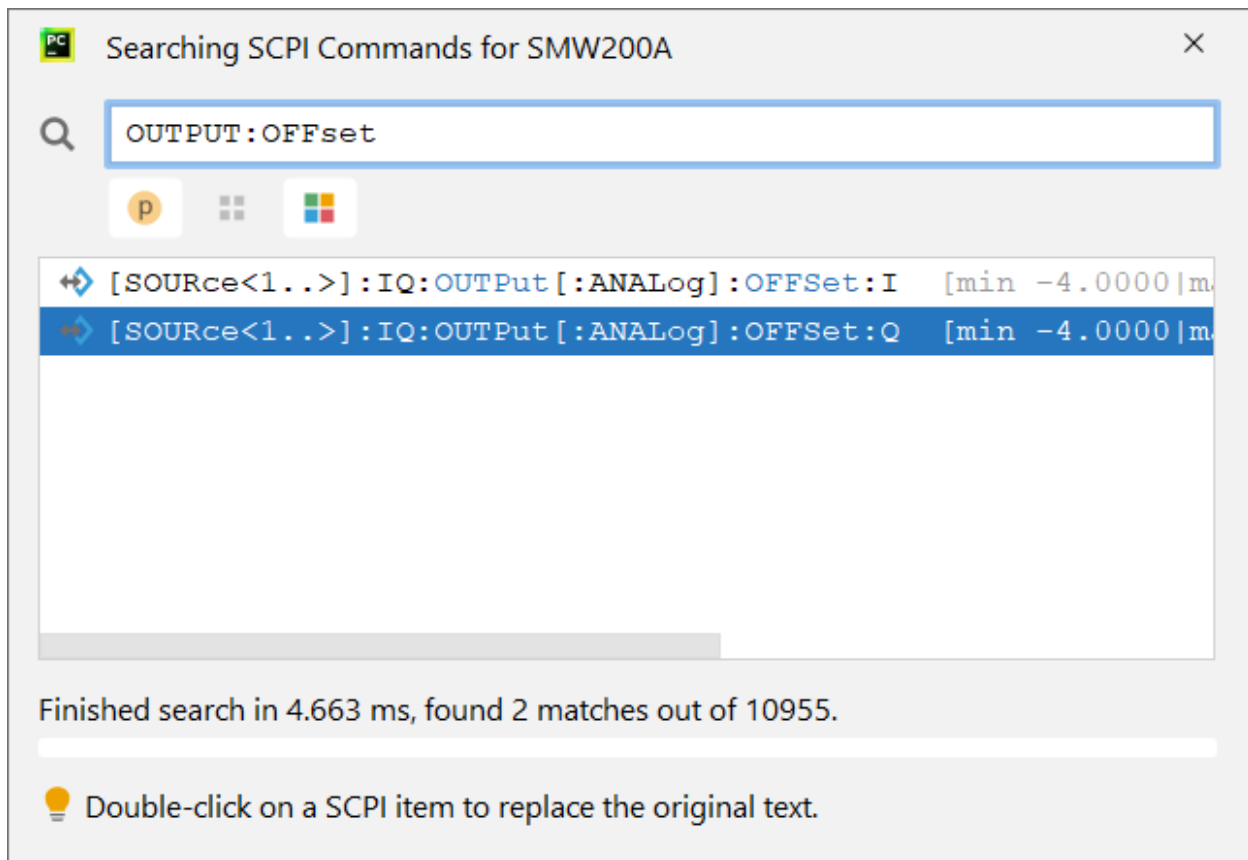


## 14.4 13.4 Writing scripts - Searching for commands

Another option to add commands, is to search for them in the SCPI Tree. You can use the *SCPI Tree Function Panel*, or search directly in the script. Right-click (or press F3) on a command or an empty line, and select **Search for Command**:



The already written command is pre-filled to the search box, and you can adjust it. **Double-click** on the desired command line in the table to insert it into your script:



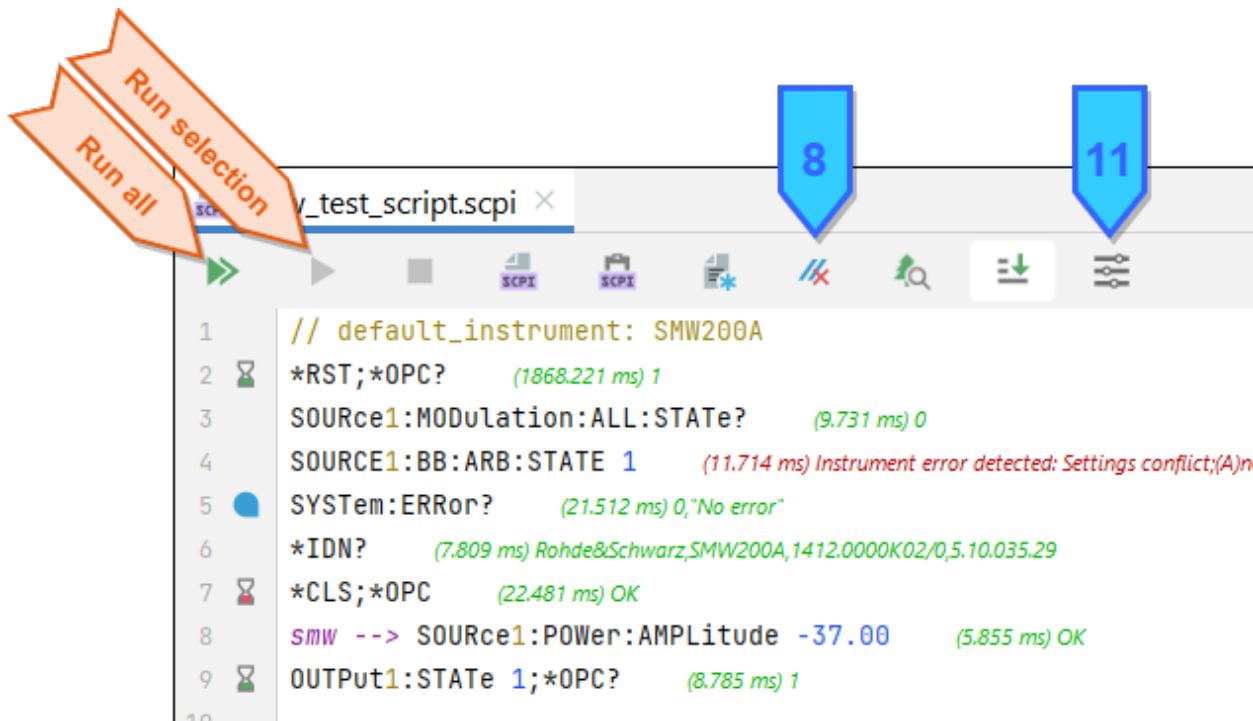
**Tip:** The Search Window is persistent, and is by default updated everytime you change line in your SCPI Editor, or you type text. Because of the fact that the space character is used as a separator between the commands and parameters, the search text ignores the characters after the space. To overcome this, use double-spaces. For example, `freq center cw` is then converted to the search string `freq center  cw`.

## 14.5 13.5 Executing scripts

To execute the script, all the involved instruments must have the Instrument Tool Windows active, and their connections established.

When you execute a script, selection, or a single line, the results of the execution are presented as inlays directly in the editor:



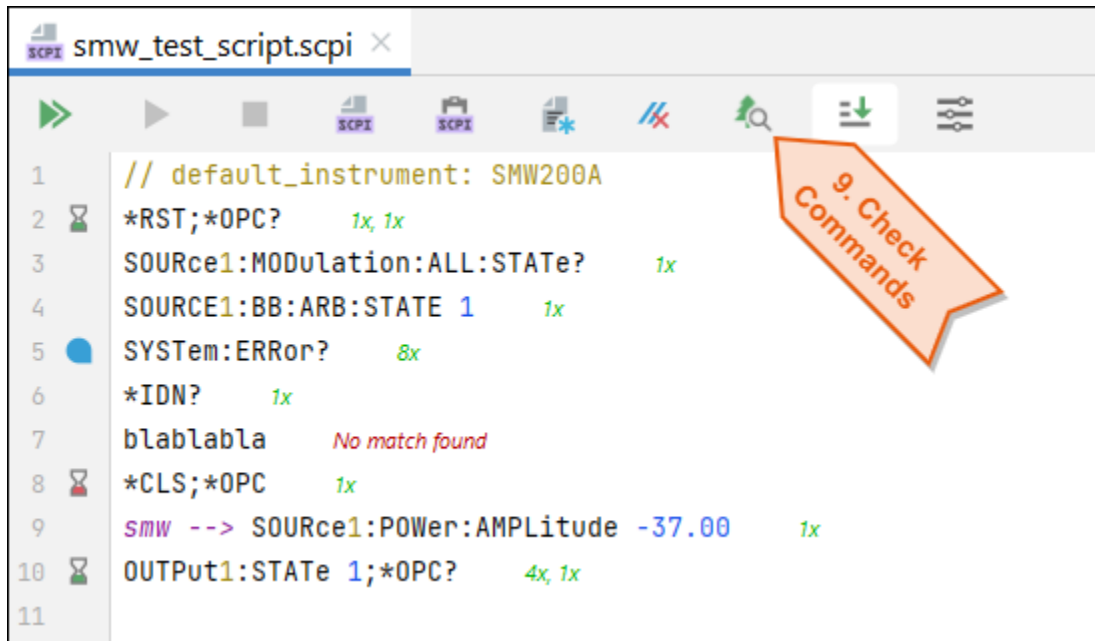


You can adjust the inlay properties in the *Settings for the SCPI Editors* (button 11).

**Hint:** Clear the inlays with Button 8 - **Clear all inlays** or just a portion of them by selecting the desired lines and right-click context menu **Clear Inlays**.

## 14.6 13.6 Checking the commands

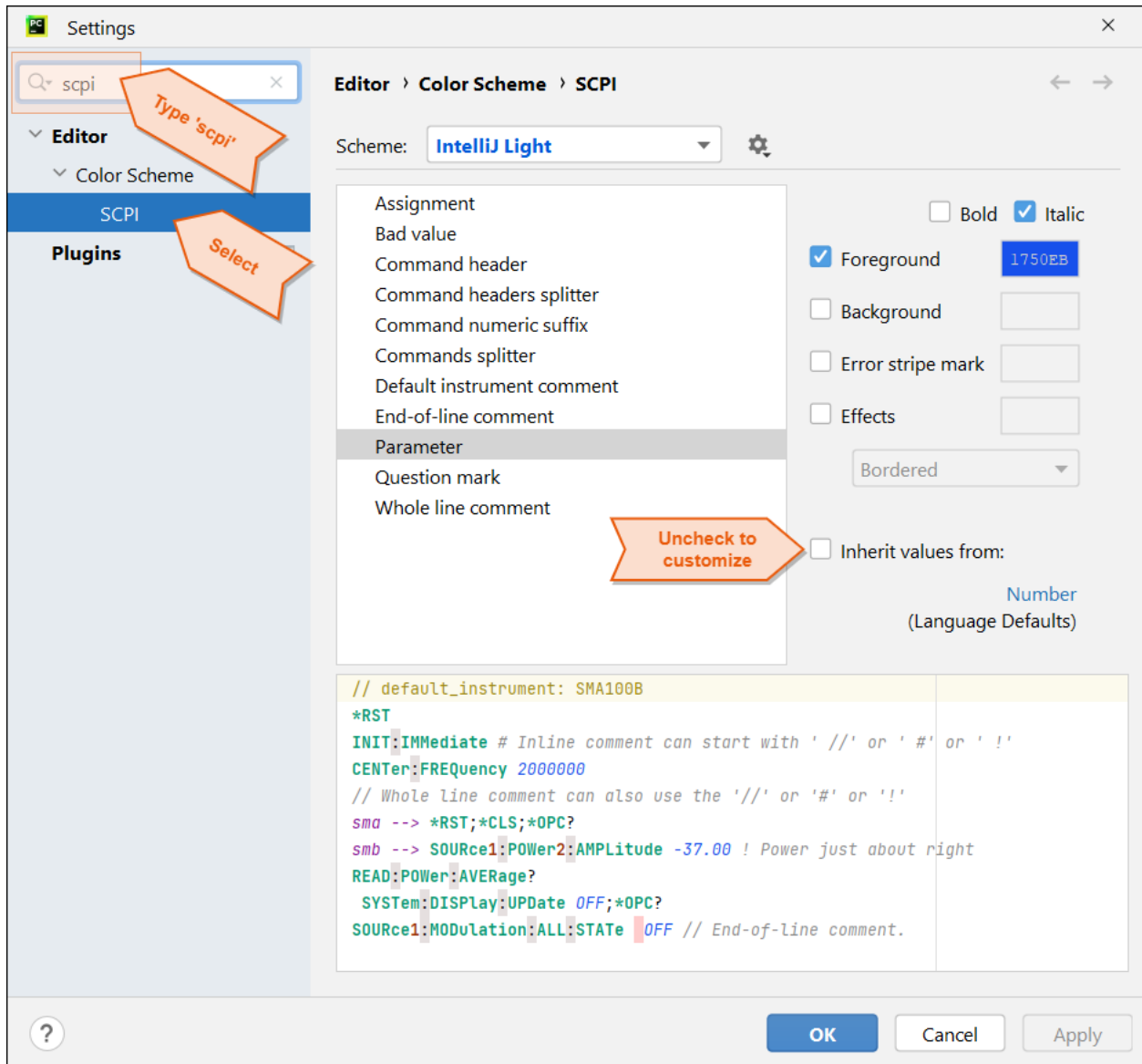
A very useful feature is to check the commands in the script, whether they are defined in the instrument's SCPI Tree. This provides a quick plausibility check. Press the **Check in SCPI Tree** button:



Green inlays signal how many matches in the SCPI were found. Red inlays mark the commands not present. Do this quick check to prevent unnecessary VISA timeout errors when executing the script with unknown commands.

## 14.7 13.7 SCPI Editor color scheme

You might have noticed that the SCPI Editor text has several different colors. The cool thing is, you can fully customize it. Go to **Pycharm settings -> Editor -> Color Scheme -> SCPI**, and let's adjust the command headers to be bold green, the header suffixes bold violet, parameters italic, and the header separators to have gray background - just for fun :-)

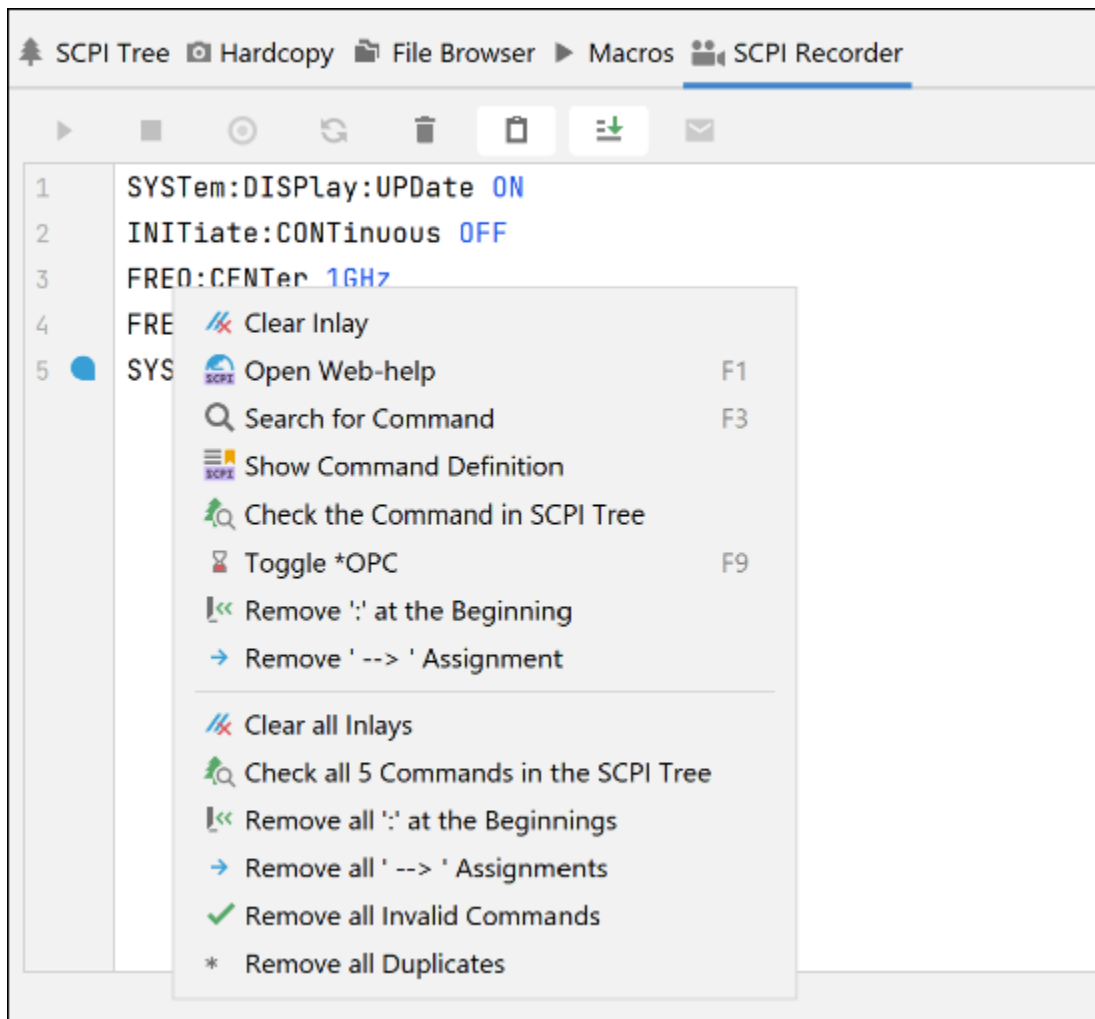


## 14.8 13.8 SCPI Editor Flavors

The SCPI Editor component is used in multiple different flavors. They all have the auto-completion, but differ in execution and parsing features.

## 14.8.1 13.8.1 SCPI Recorder Function Panel

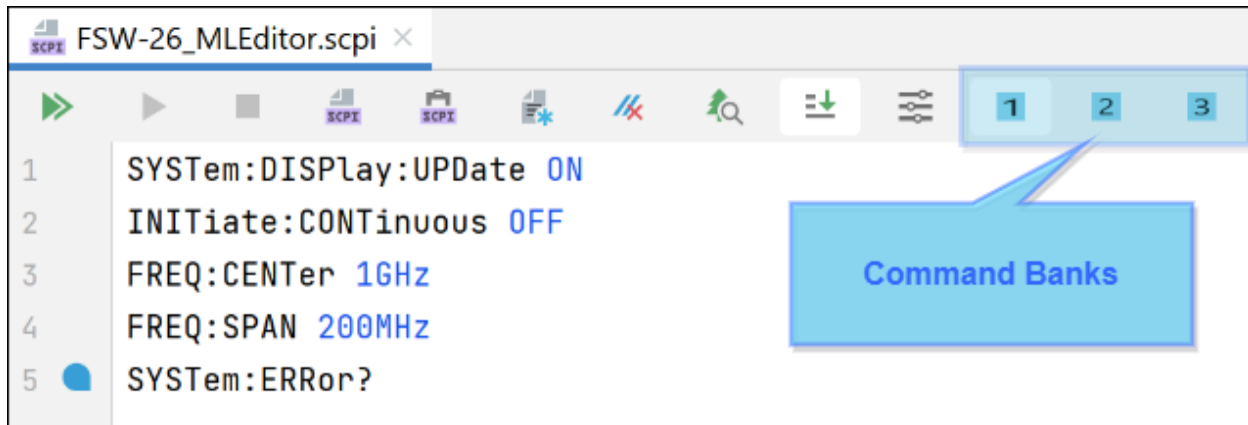
*SCPI Recorder Function Panel* does not have the control panel, but most of the functionalities are available per right-click context menu:



However, the execution and parsing are not available. For that, you have to drag or copy the content to a SCPI file.

## 14.8.2 13.8.2 Multiline SCPI Editor

We already mentioned *Multiline SCPI Editor* at the beginning: The content is always related to the instrument's ITW. Its content is not saved in a \*.scpi file. Instead, you have three **Command banks** that you can switch between and use them practically as three different files. The entire content is persistent and is stored in the plugin XML file:

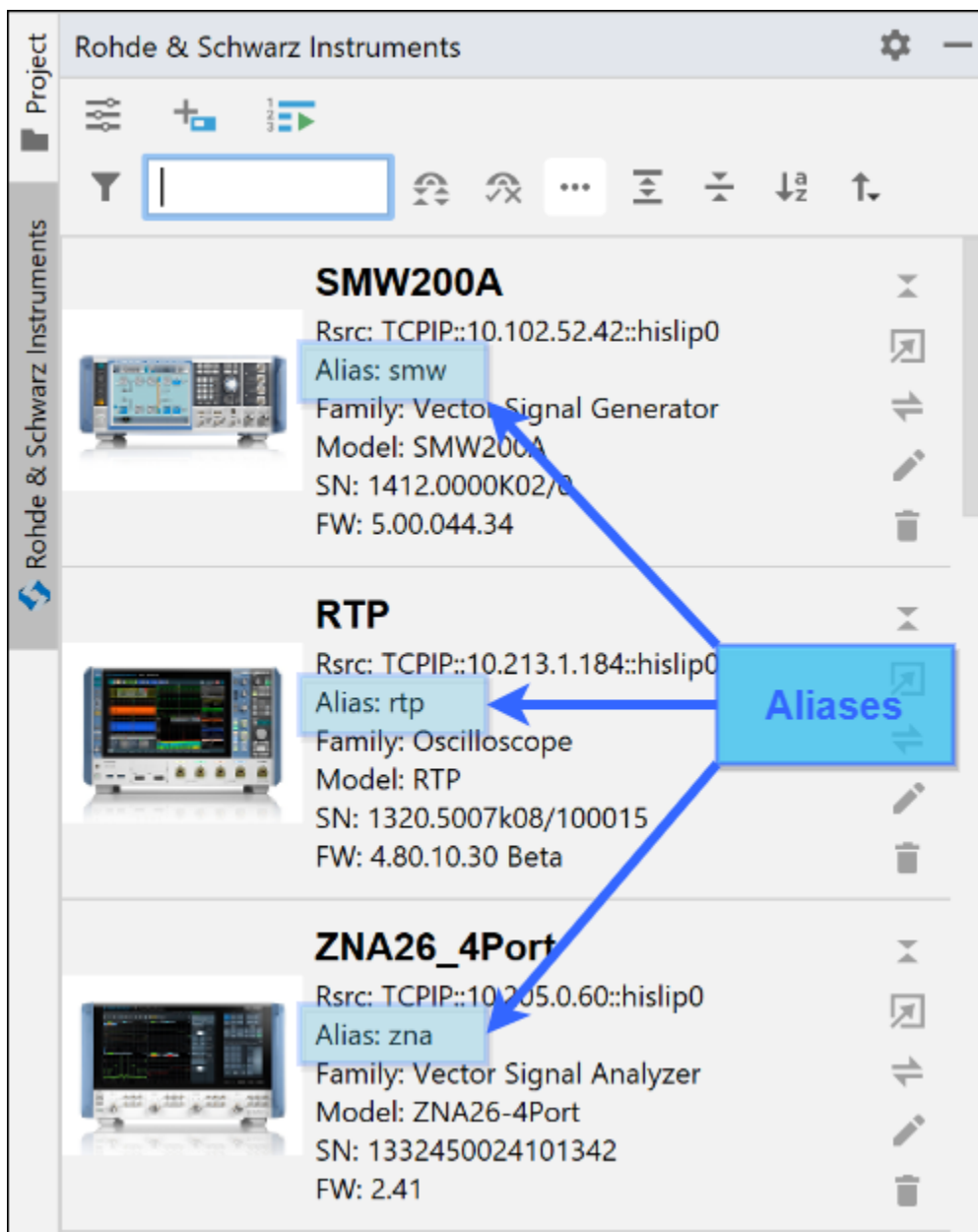




## 14. WRITING PYTHON SCRIPTS

Before you continue with this chapter, *read out the SCPI Tree*. That allows you to use SCPI auto-completion and SCPI search in your SCPI script.

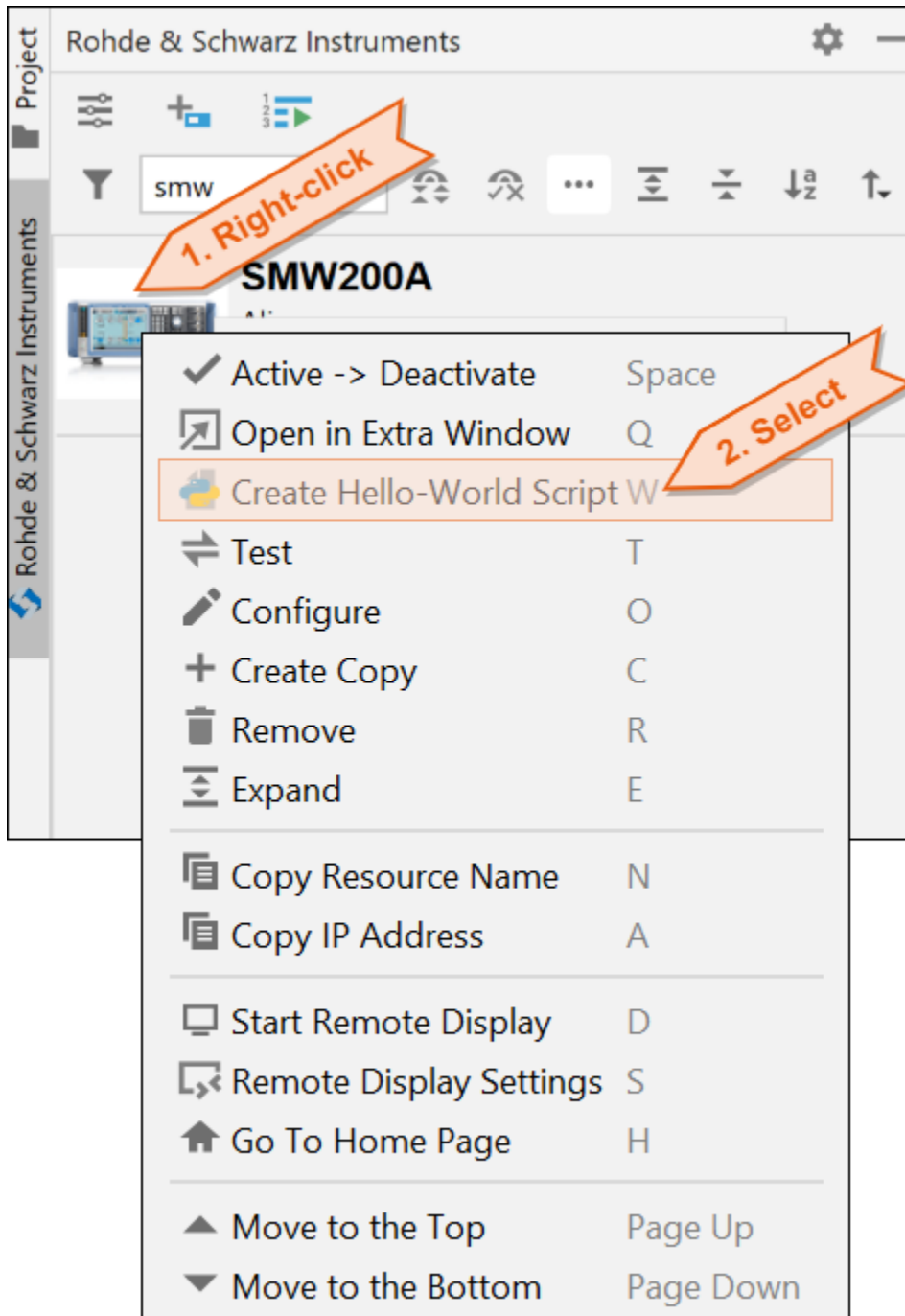
Let us write some python remote-control script for your SMW200A. The most important value that binds your instrument from the list to your python script is the **Alias**, in our case, `smw`. This is going to be the script variable name for our object:



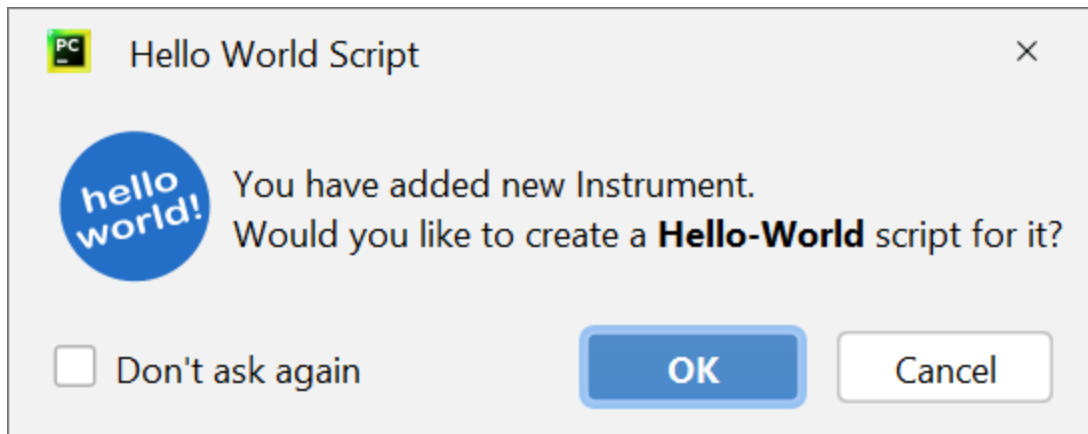
**Tip:** In some cases, where you cannot have the alias equal to the script variable name, you can set a **Default Instrument for SCPI Tree**. This is then used in cases where your script variable name does not fit any instrument alias. See the [16.4 SCPI Code Completion](#).

Let us start with creating the script file. The easiest way is to use the feature **Create Hello-World Script**:





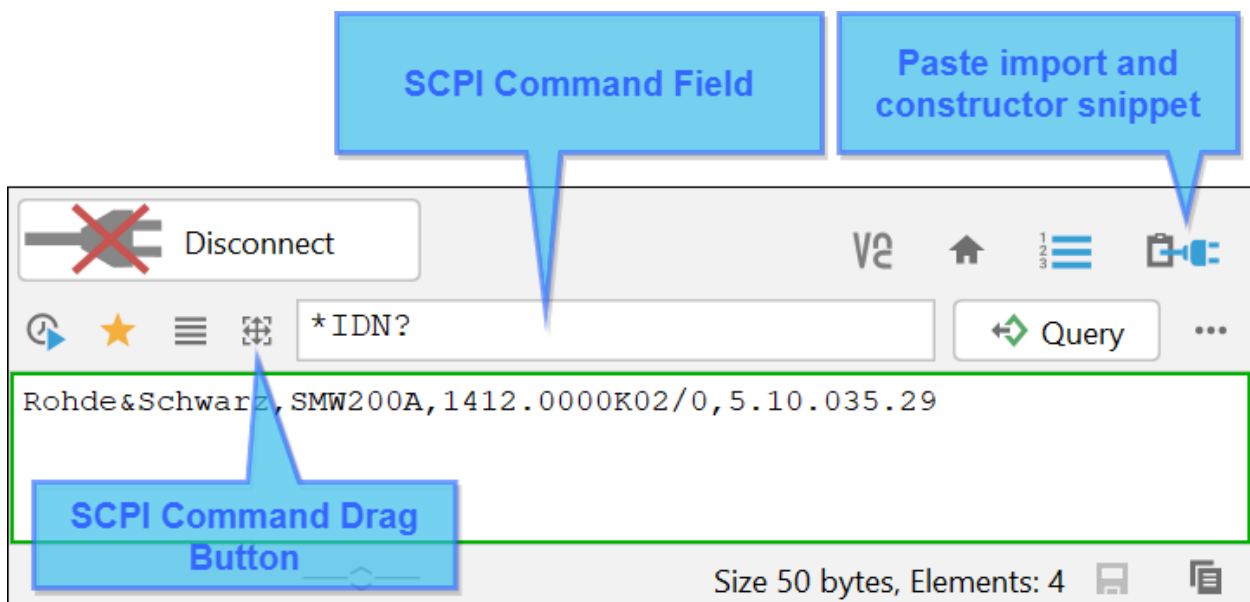
You also get this option offered when you add a new instrument:



You can choose the path where to store the script. By default, it is stored in your project's root folder under the name `hello_world_<alias>.py`, in our case `hello_world_smw.py`. The created script is already runnable, and the cursor is on the correct position to add SCPI calls. We now go through ways how to do that:

## 15.1 14.1 Writing scripts - SCPI Communicator drag function

Once again, the important controls of the SCPI Communicator for the purpose of this chapter:

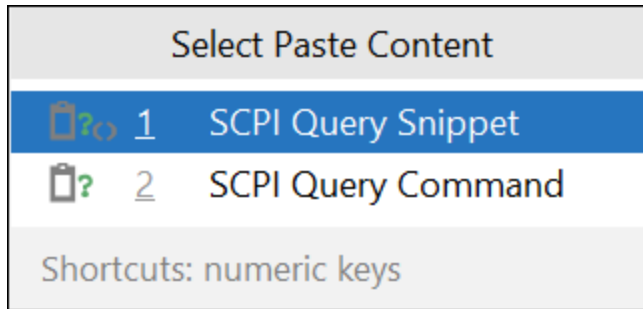


If you start with a python script from scratch, you need to initialize the connection to your instrument. Use the button **Paste Import and Constructor Snippet** to insert for example this code:

```
from RsInstrument import *

smw = RsInstrument('TCPIP::10.102.52.47::hislip0', reset=False)
```

Enter the `*IDN?` to the **SCPI Command Field**, use the **SCPI Command Drag** button, and drag the command to an appropriate place in your script. On dropping, select **SCPI Query Snippet**:



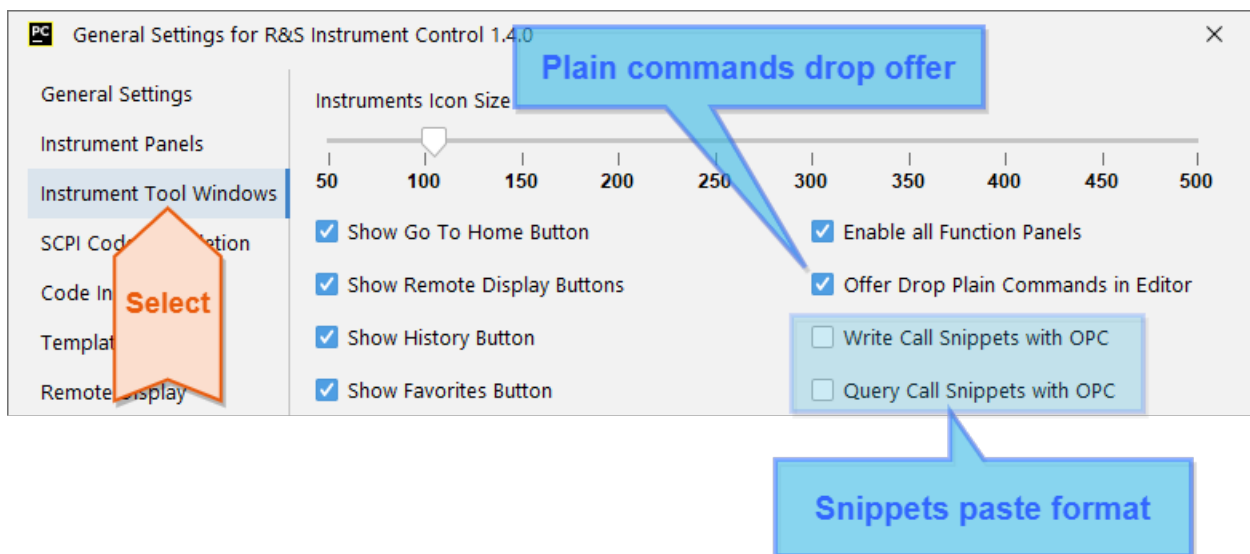
You script will look similar to this:

```
from RsInstrument import *

smw = RsInstrument('TCPIP::10.102.52.47::hislip0', reset=False)
result = smw.query('*IDN?');
```

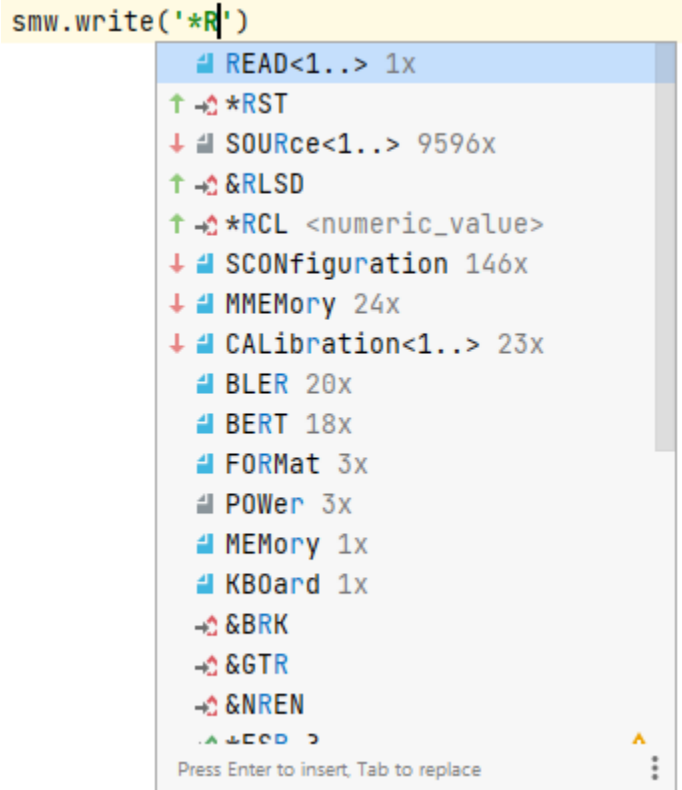
**Tip:** You can change the format of the pasted code with templates in [16.6 Templates](#).

To change the pasted snippets between `write` / `write_with_opc` and `query` / `query_with_opc`, use the [16.3 Instrument Tool Windows](#). You can also disable the option of dropping the plain commands. In such case, the dropping happens immediately:



## 15.2 14.2 Writing scripts - Auto-completion

SCPI auto-completion works with python call expressions, where the origin object name is your instrument alias, and the method contains an argument of string-type. Type (do not copy/paste) for example this:



Select your desired command and use **TAB** to insert it. Pycharm immediately shows you another part of the command to auto-complete.

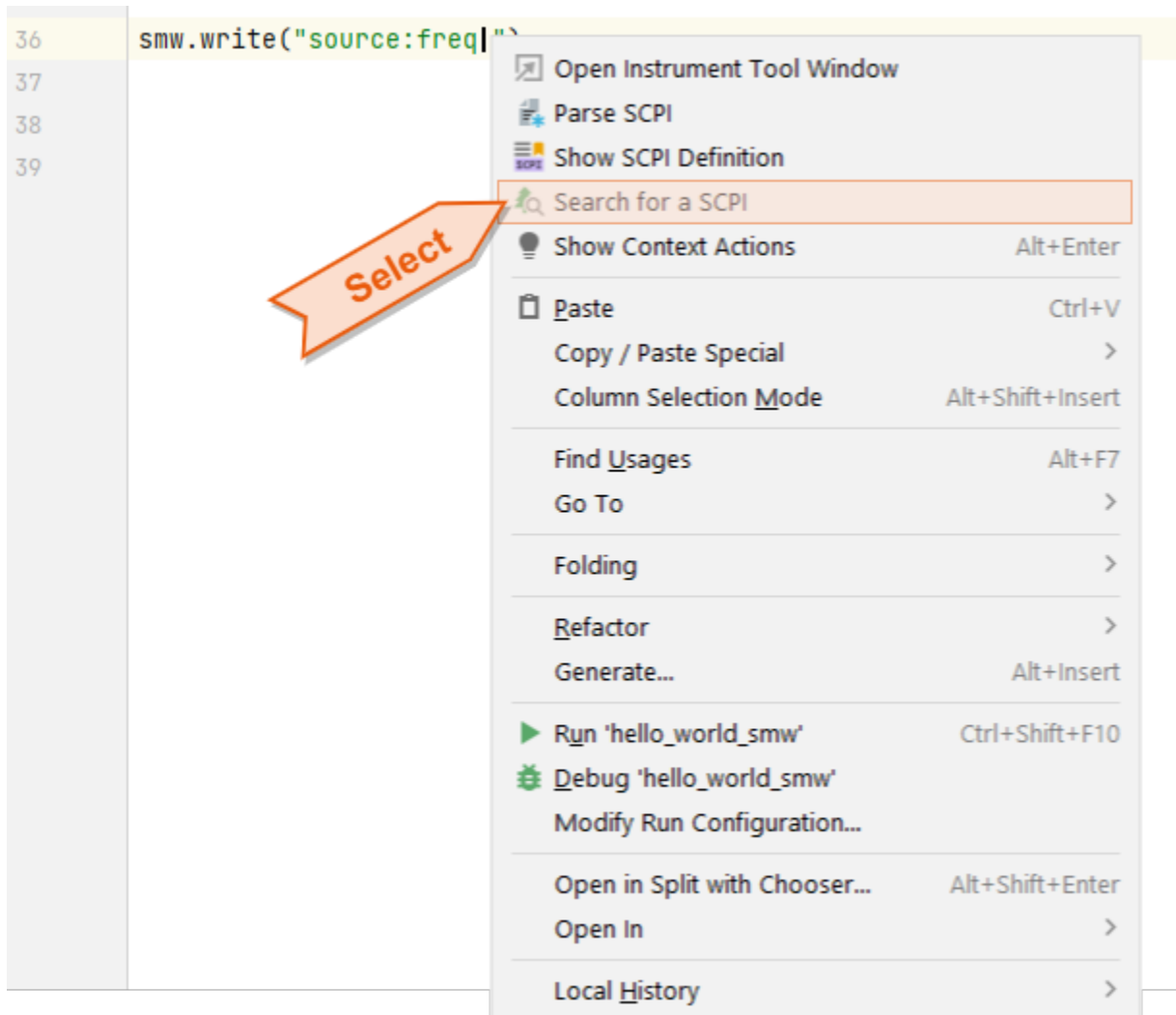
---

**Tip:** You can force the auto-completion window to pop up with the keyboard combo **CTRL+SPACE**

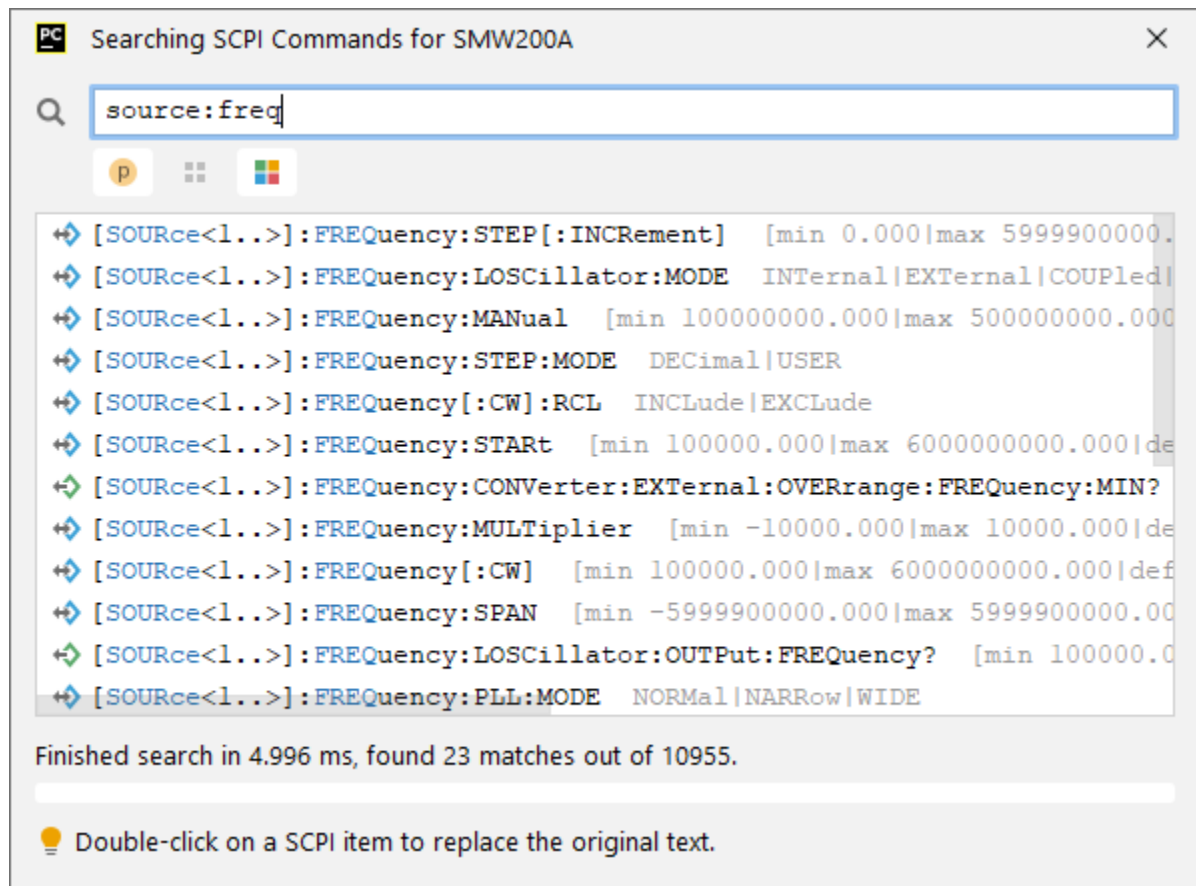
---

## 15.3 14.3 Writing scripts - Searching for commands

Similar to \*.scpi files, you can add commands by searching for them. The right-click context menu is caret-position-sensitive, it shows the option **Search for command** only if you place the caret within a SCPI string:



The already written command is pre-filled to the search box, and you can change or amend it. **Double-click** on the desired command line in the table to insert it into your script:



## 15.4 14.4 Writing scripts - Parsing from other formats

The *SCPI parser* is available in the right-click context menu. Let us take the following R&S SMW example written as MATLAB script. Copy it to your Clipboard:

```
smw = visa('ni', ['TCPIP::192.168.221.3::INSTR']);
try
    fopen(msInstr);
    msgbox('Connecting to SMW successful');
catch
    errordlg('Cannot connect to the SMW');
    return;
end

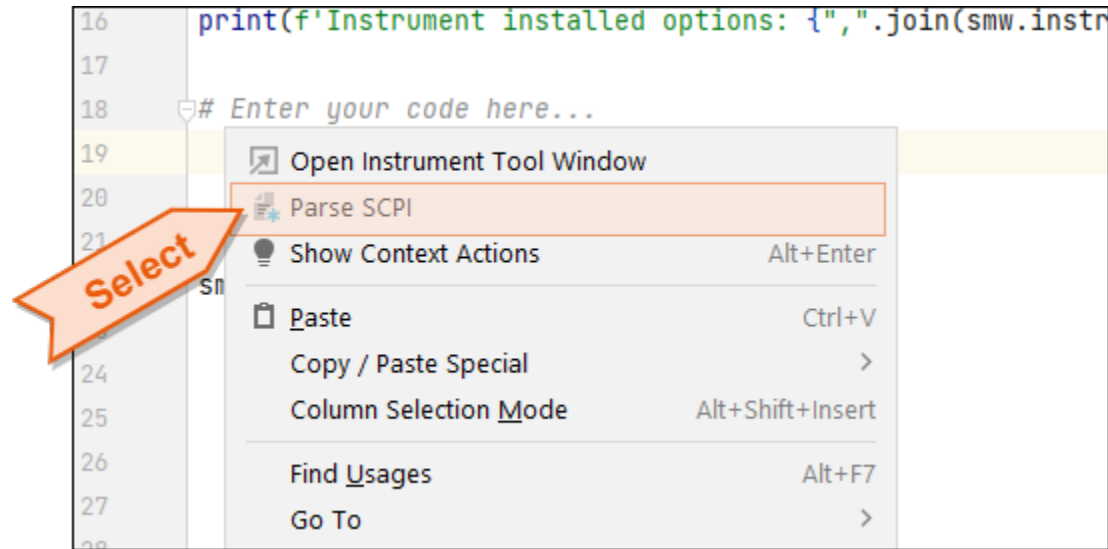
fprintf(smw, '*RST');
fprintf(smw, '*CLS');
fprintf(smw, '*IDN?');
idn = fread(smw);
fprintf(smw, 'SOURce1:POWer:POWer -40');
fprintf(smw, 'SOURce1:BB:EUTRa:STaTe 1');
fprintf(smw, 'SOURce1:BB:EUTRa:LiNK UP');
fprintf(smw, 'SOURce1:BB:EUTRa:UL:BW BW1_40');
fprintf(smw, 'SOURce1:FREQuency:CW 2000000000');
```

(continues on next page)

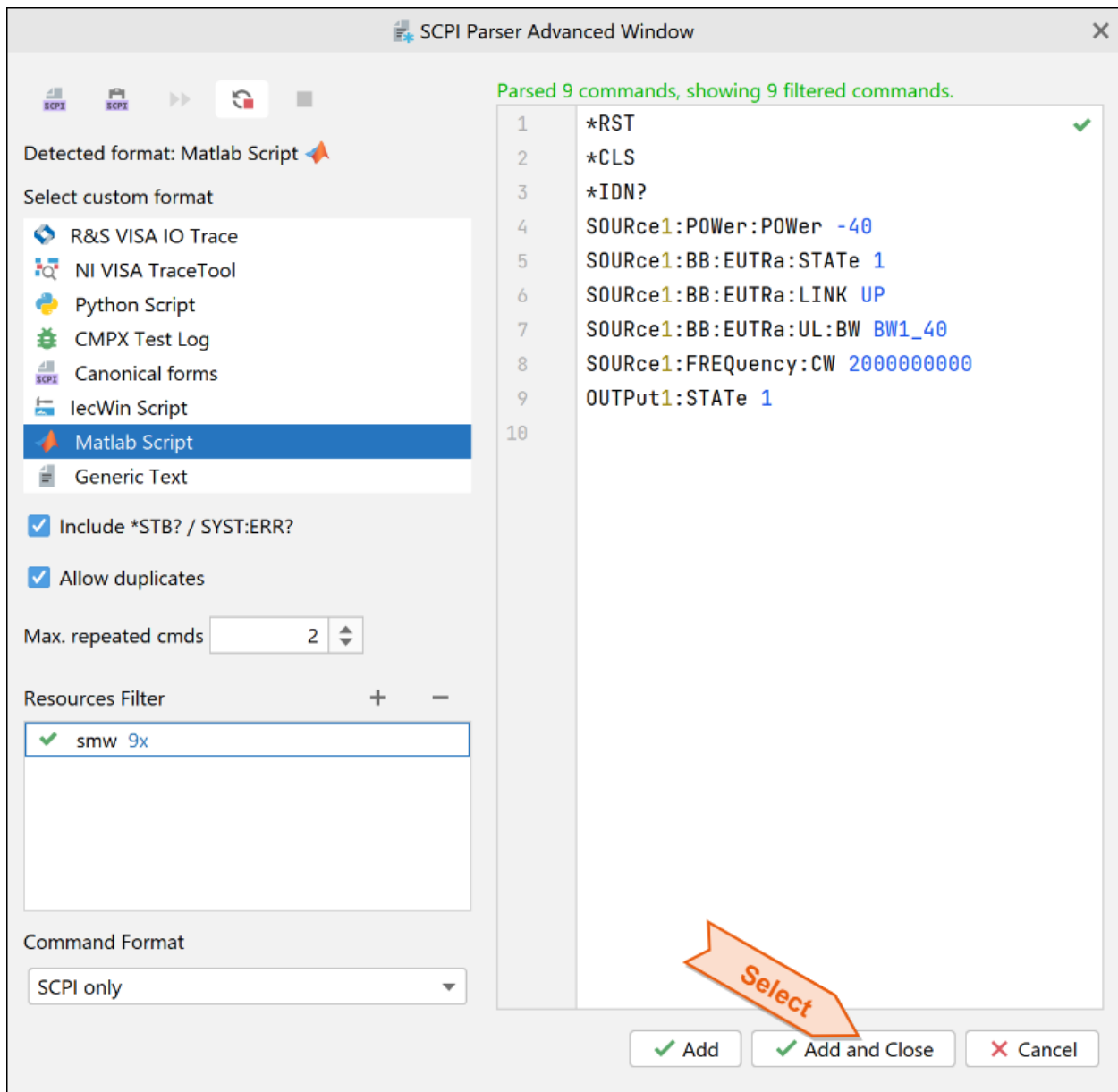
(continued from previous page)

```
fprintf(smw, 'OUTPut1:STATe 1');
```

Go to your python script, right-click on the place you want to add the code and select **Parse SCPI**:



The parser recognises the format as MATLAB and shows you only the parsed SCPI commands:



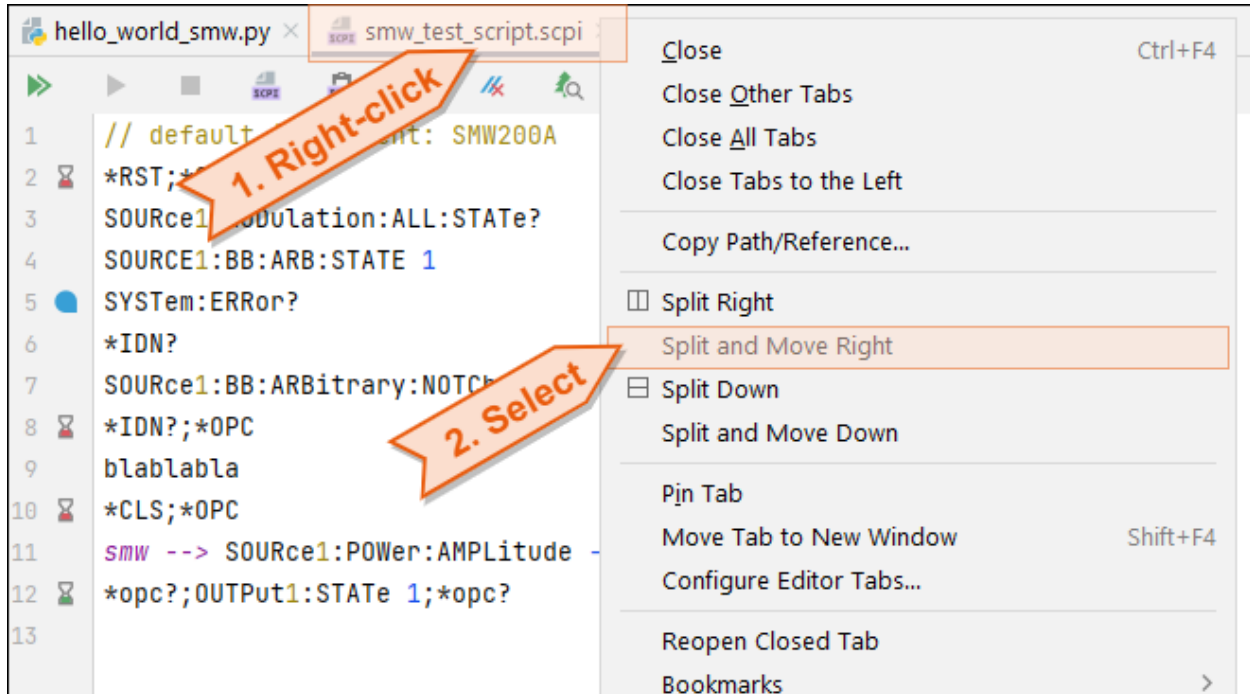
Now click **Add and Close**, select the default target instrument and your commands are added as python snippets. The result looks something like this:

```
smw.write("*RST")
smw.write("*CLS")
resp = smw.query("*IDN?")
smw.write("SOURce1:POWer:POWer -40")
smw.write("SOURce1:BB:EUTRa:STATE 1")
smw.write("SOURce1:BB:EUTRa:LINK UP")
smw.write("SOURce1:BB:EUTRa:UL:BW BW1_40")
smw.write("SOURce1:FREQuency:CW 2000000000")
smw.write("OUTPut1:STATE 1")
```

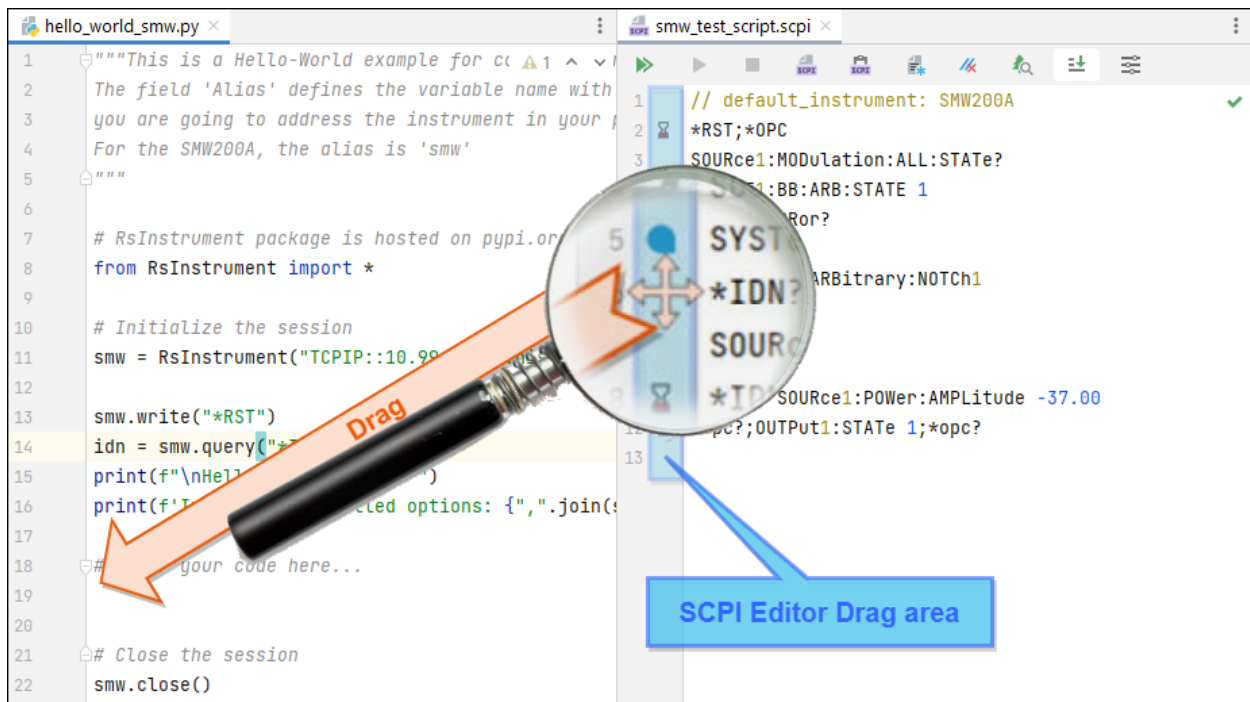


## 15.5 14.5 Writing scripts - Drag & Drop from SCPI files

Here we combine two great features - one of Pycharm and one of RsIC plugin: We arrange our SCPI Editor and our Python Editor side-by-side. Open the scripts `smw_script.scpi` and `hello_world_smw.py` in your editors. Right-click on the `hello_world_smw.py` tab, and select **Split and Move Right**:



Now drag for example the `*IDN?` command using the SCPI Editor Drag area to your python script:



Of course, we support dragging of multiple lines: make a selection in the SCPI editor and use the drag area to drag

them to your python script.

---

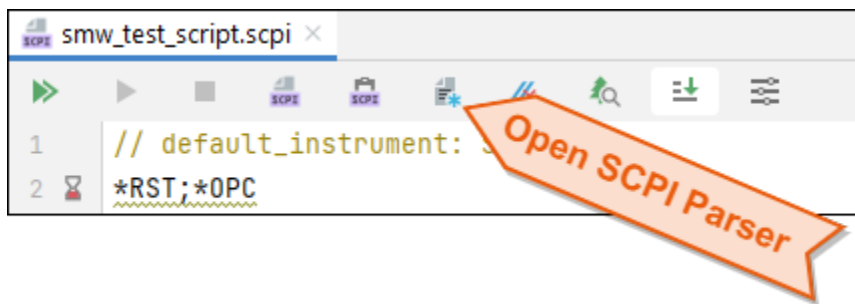
**Note:** By splitting/moving editors, you can have a situation where one SCPI file is opened in multiple SCPI Editors. In such case, the executing controls are only available in one of them.

---

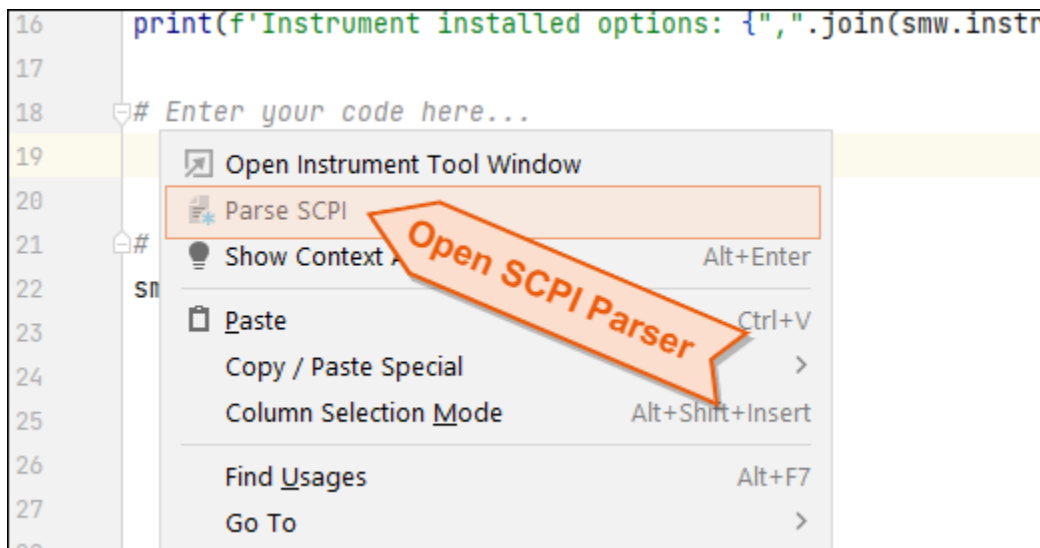
## 15. ADVANCED SCPI PARSER

This parser is accessible from any SCPI Editor or a context-menu of a python script. It gives you the possibility to parse a Clipboard text or a file content and extract SCPI commands out of it. We support typical formats, like for example, R&S IO Trace, NI IO Trace, MATLAB, IECWin, R&S Forum, another Python script...

Accessing the parser from a **SCPI Editor**:



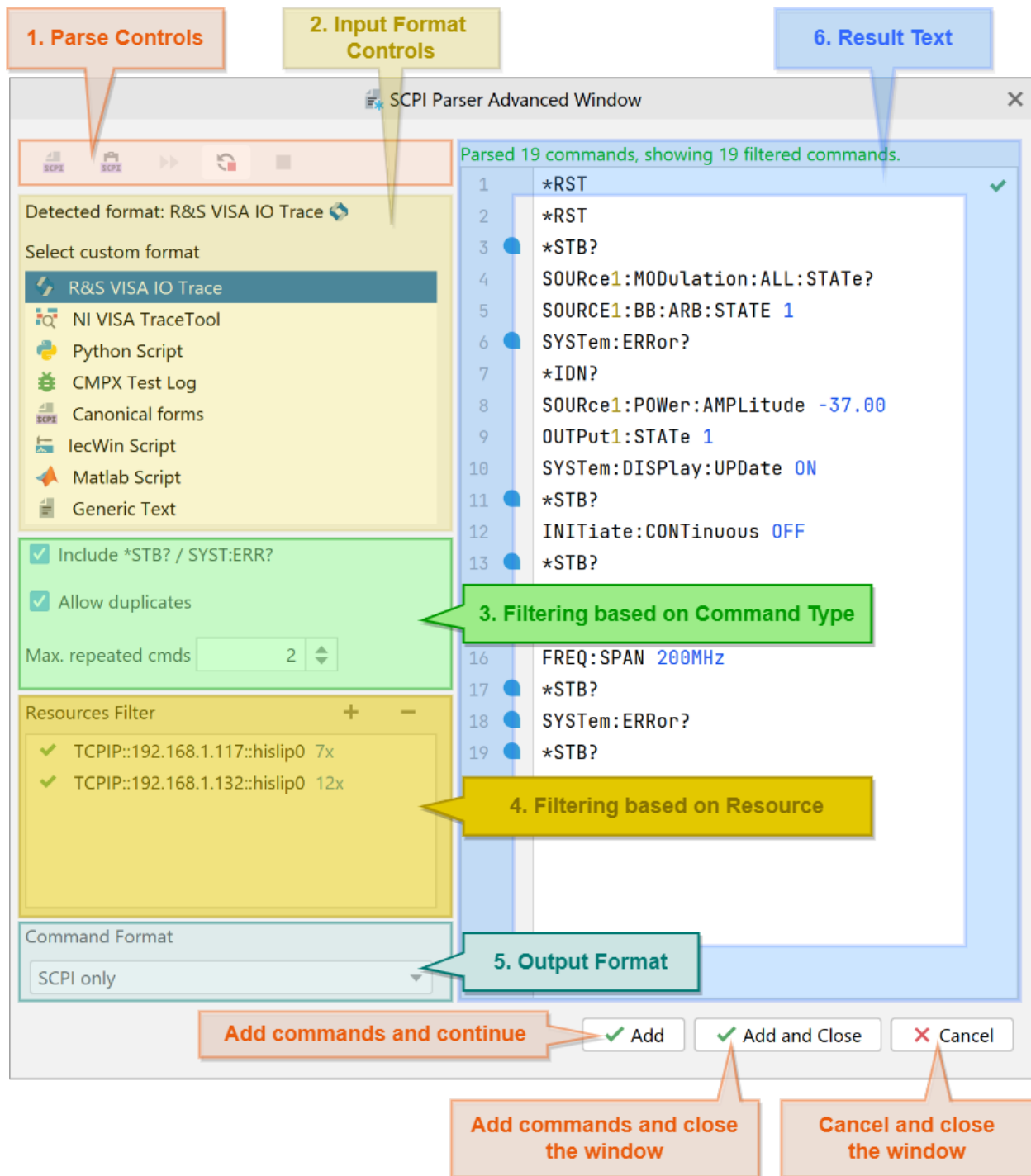
Accessing the parser from a **python script** through right-click context menu:



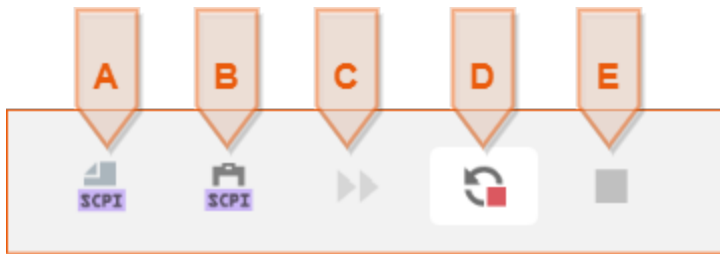
Let us parse a **R&S VISA Trace Log** as an example:

- Copy the content of the following log file to the Clipboard: `rsvisa_log_smw.log`
- Open your python script `hello_world_smw.py` and place the caret to the place where you want to add the code.
- Right-click and select **Parse SCPI** as described above.

The Advanced Parser window opens:



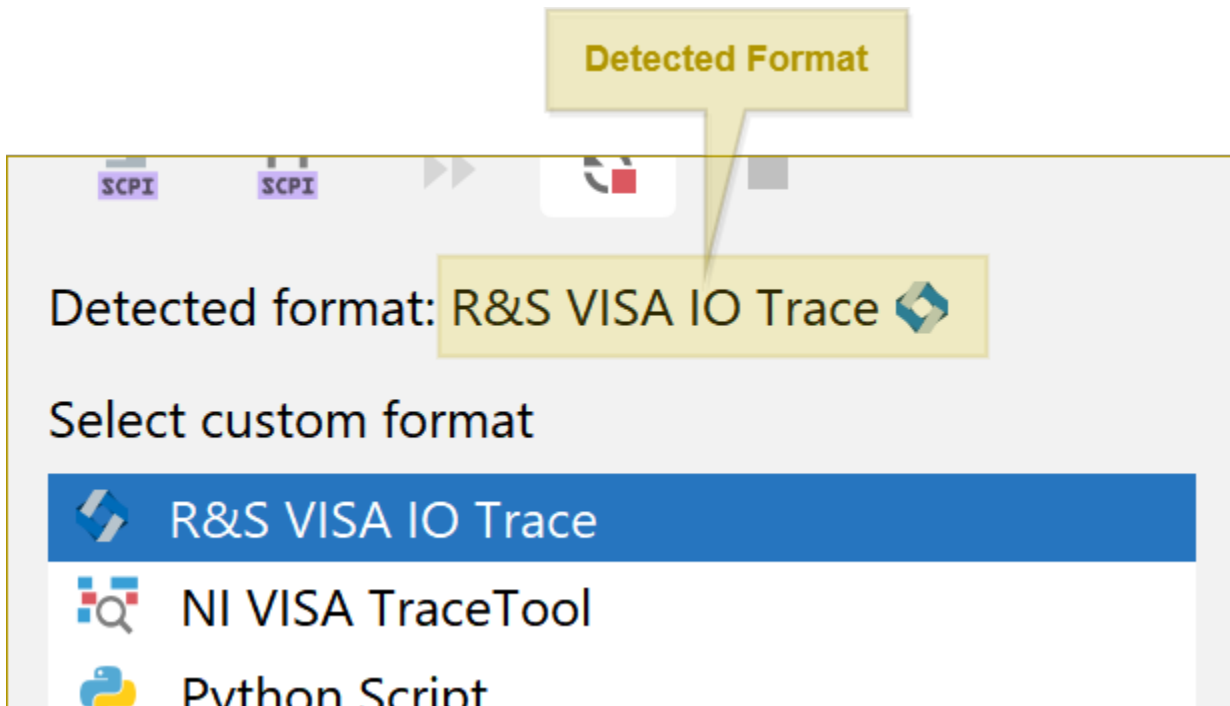
As the window opened, it detected text content in your Clipboard and immediately parsed it for SCPI commands. You can always repeat the parsing with the **1. Parse Controls**:



Description of the **Parse Controls**:

- A. **Parse from File** - select a file to parse its content - use this to parse for example, a VISA IO Trace Log.
- B. **Parse from Clipboard** - parses current text content of the Clipboard.
- C. **Start the parsing** - by default, smaller content is parsed automatically after any change of parse/format settings. For big files, you might want to initiate the parsing manually. This button is enabled only if the **Autoparsing Toggle** is OFF.
- D. **Autoparsing Toggle** - starts the parsing after any change of the controls / filters. If you disable this, you can manually start the parsing.
- E. **Stop** - stops the current parsing job. The already parsed commands are retained.

You can see that the **2. Input Format Controls** show the proper detected format: R&S VISA IO Trace:



If for any reason the auto-detection fails, you can always change the format manually.

**Hint:** The result you see when you open the Advanced Parser window, is just the initial parsing of the Clipboard text. You can copy new text to the Clipboard or open a file and perform the parsing with different content.

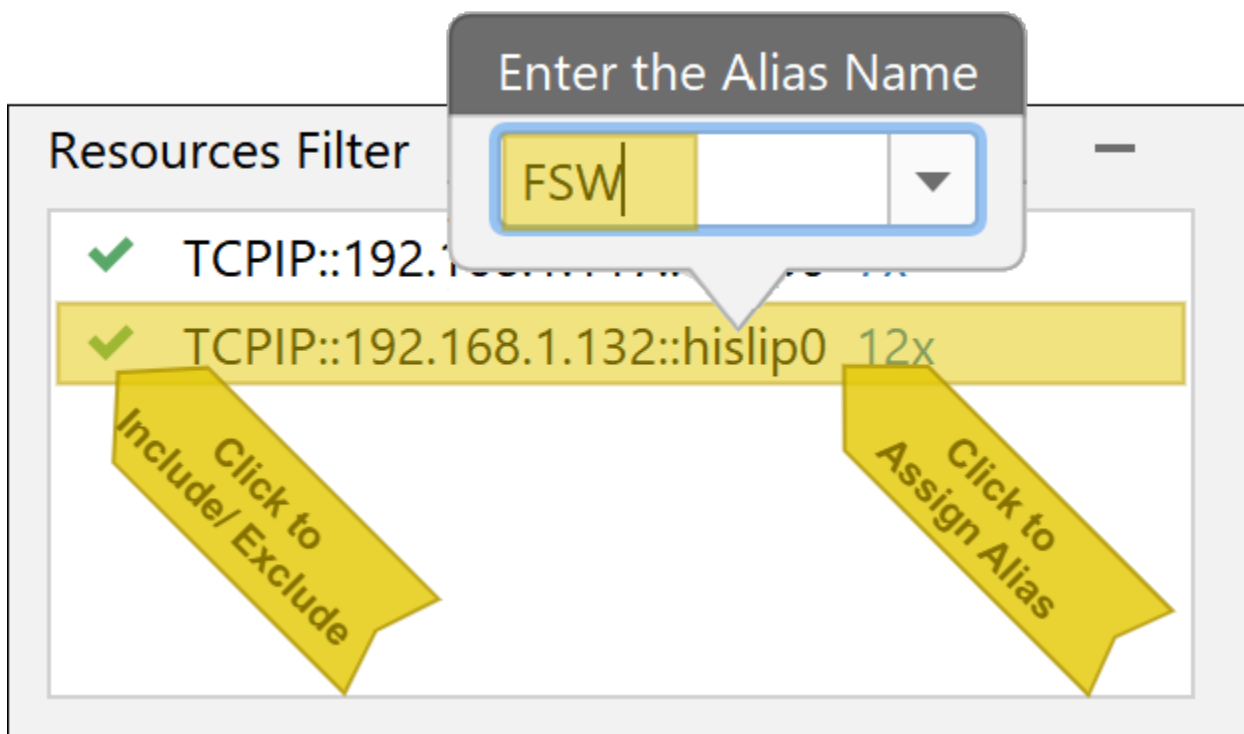
Sometimes you want to exclude error-checking commands or commands that are repeated many times (in a row or globally). For that, you can use the **3. Filtering based on Command Type**. By default, all the commands are shown, but only two same commands in a row:

☒ Include \*STB? / SYST:ERR?

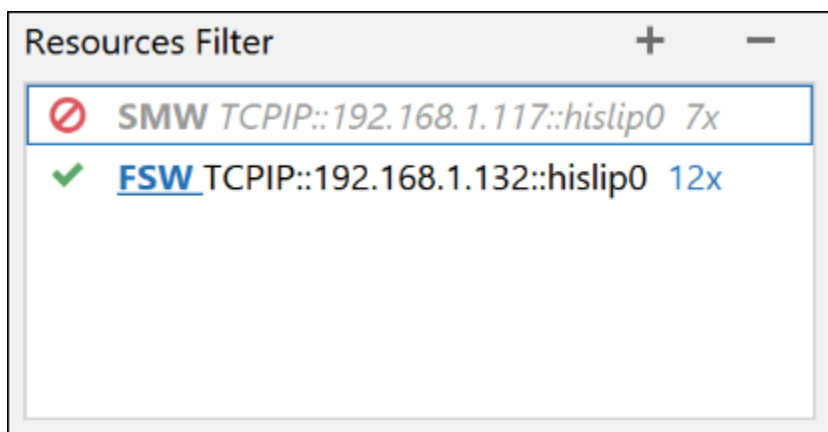
☒ Allow duplicates

Max. repeated cmds

The last input control is **4. Filtering based on Resource**. Using this, you can extract only commands belonging to specific resources. For example, select only the commands sent to the spectrum analyzer. That is the resource `TCPIP::192.168.1.132::hislip0`. Let us give it a better alias name. Click on it and name it `FSW`:



Now rename the first resource `TCPIP::192.168.1.117::hislip0` to `SMW` and disable it by clicking on its check symbol. The result resource filter will look similar to this:



Notice the **6. Result Text** now only show the commands for the FSW.

You might ask - what for did we enter the Alias? That value has to do with the last control **5. Output Format**:

Command Format

SCPI only

By default, this value is set to **SCPI only**. Switch it to **SCPI with resource prefix**. You see the result text have the commands with the resource prefix. For example: FSW --> \*RST. This has consequences on how your commands are imported into your python or SCPI scripts:

- **SCPI scripts** import the text verbatim. That means, what you see in the **6. Result Text** is what you get in the SCPI Editor.
- **Python scripts** import commands as snippets. For commands, that also have the resource assignment, the resource is used as an object name for the call. Example: FSW --> \*RST is imported as:

```
FSW.write("*RST")
```

For commands without resource association, the Pycharm asks you to select a default instrument:

DEFAULT Target Instrument

<CUSTOM>

FSW-26: fsw

MXO44: mxo

SMA100B: sma

SMW200A: smw

ZNL20-2Port: znl

Type to filter the elements

**Tip:** For Python scripts, if you want to import the commands but use different call object name, import them as SCPI only and select your own call object.

#### List of all output formats:

- **SCPI only:** only show SCPI command. Examples:

```
*RST
*CLS
*IDN?
```

imported into python script after defining default instrument **smw**:

```
smw.write("*RST")
smw.write("*CLS")
resp = smw.query("*IDN?")
```

- **SCPI with comment:** SCPI command and an end-line comment (if present). Example:

```
*RST # Reset the instrument
*CLS
*IDN? # Query the identification string
```

imported into python script after defining default instrument smw:

```
smw.write("*RST") # Reset the instrument
smw.write("*CLS")
resp = smw.query("*IDN?") # Query the identification string
```

- **SCPI with comment or resource comment:** SCPI command and a resource as a comment, or an end-line comment. Resource comment has priority. Example:

```
*RST # TCPIP::192.168.1.132::hislip0
*CLS # FSW
```

imported into python script after defining default instrument smw:

```
smw.write("*RST") # TCPIP::192.168.1.132::hislip0
smw.write("*CLS") # FSW
```

- **SCPI with resource prefix:** this case we discussed above. Example:

```
FSW --> *RST
FSW --> *CLS
SMW --> *RST
SMW --> *IDN?
```

imported into python script (no default instrument needed):

```
FSW.write("*RST")
FSW.write("*CLS")
SMW.write("*RST")
resp = SMW.query("*IDN?")
```

- **SCPI with resource prefix and comment:** combination of two above. Example:

```
FSW --> *RST # Reset the instrument
FSW --> *CLS
SMW --> *RST # Reset the instrument
SMW --> *IDN? # Query the identification string
```

imported into python script (no default instrument needed):

```
FSW.write("*RST") # Reset the instrument
FSW.write("*CLS")
SMW.write("*RST") # Reset the instrument
resp = SMW.query("*IDN?") # Query the identification string
```



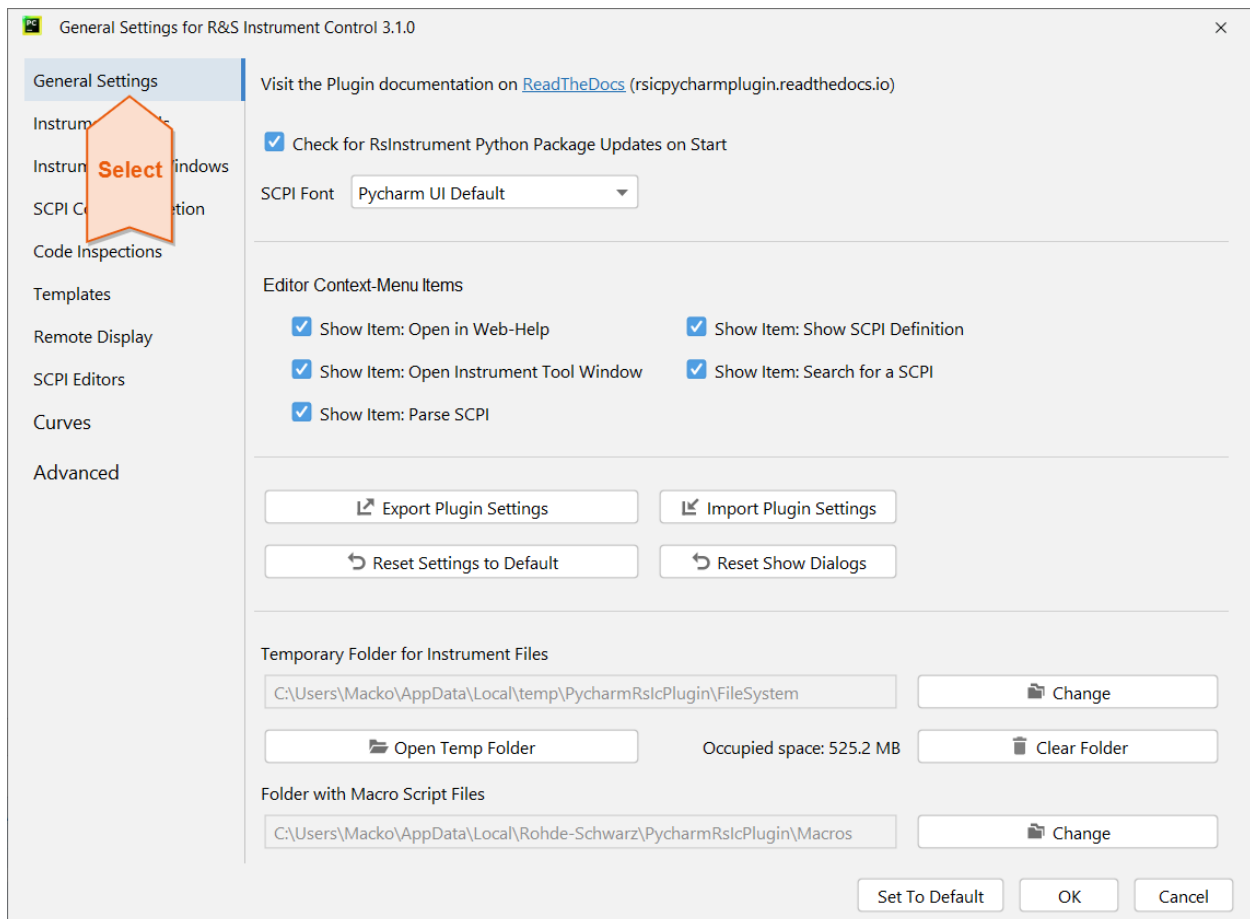
## 16. PLUGIN SETTINGS

Here, you can find all the general settings (non-instrument specific) for the plugin. You can invoke it from:

- the *Main Instrument Panel*
- the Pycharm menu *File -> Rohde & Schwarz IC Settings*
- any *SCPI Editor* button 11

### 17.1 16.1 General settings

General settings for the plugin, plus import/export features for the stored data.

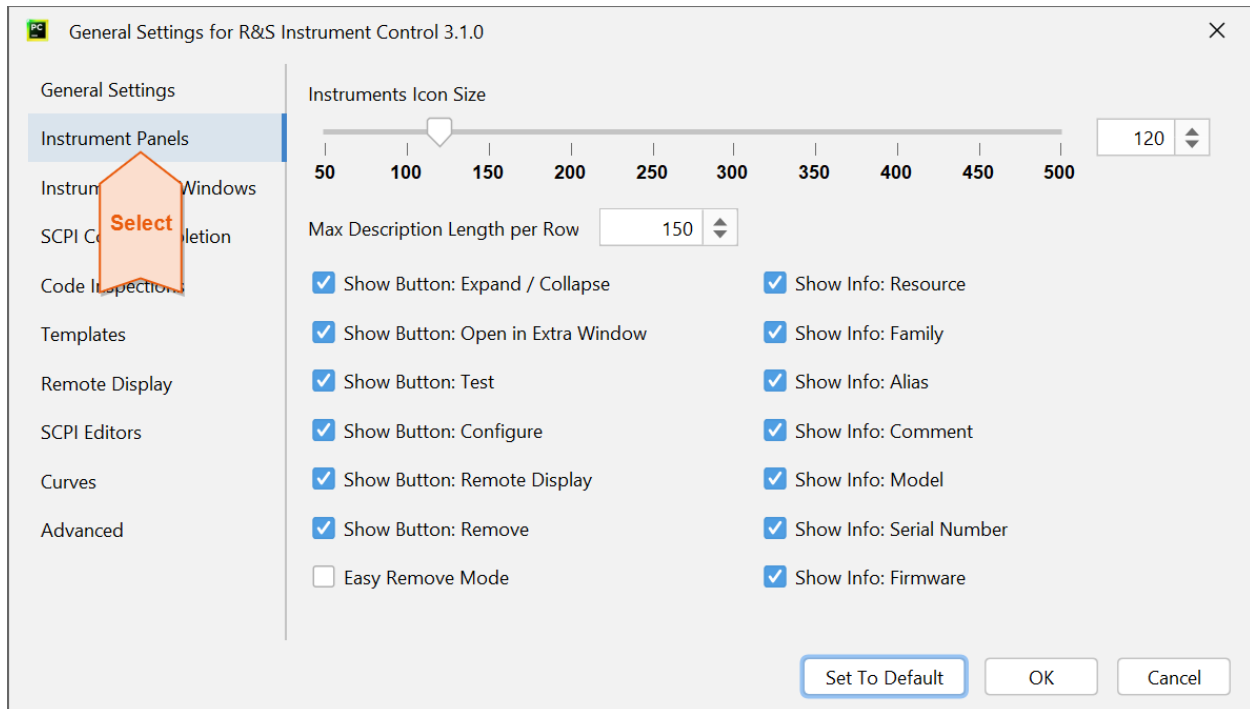


Description of the controls:

- **Check for RsInstrument Python package** - when checked, the plugin checks for the updates of the RsInstrument package. This only happens, during the IDE start. Disable it if it takes too long.
- **SCPI Font** - change the font for SCPI components - SCPI Communicator, SCPI Tree, File Browser, Favorites, History and history windows, SCPI Logger (new entries).
- **Editor Context Menu Items** - items displayed in your Python script context-menu, when your caret is on a SCPI-command call:
  - **Show Item: Open in Web-Help** - show entry to open the SCPI command Web-Help.
  - **Show Item: Open Instrument Tool Window** - show entry to open the Instrument Tool Window.
  - **Show Item: Parse SCPI** - show entry to open the *Advanced SCPI Parser*.
  - **Show Item: Show SCPI Definition** - show the entry that displays the SCPI command definition.
  - **Show Item: Search for a SCPI** - show the entry to search for SCPI commands.
- **Export Plugin Settings** - exports selected plugin settings, which you can transfer to another computer, or make a backup.
- **Import Plugin Settings** - imports the selected xml file as a plugin setting. You need to restart the Pycharm afterwards.
- **Reset Settings to Default** - resets all the plugin's settings to default values. The Instrument List is preserved.
- **Reset Show Dialogs** - resets all the 'Do not show anymore' checkboxes on popup messages back to false.
- **Temporary Folder for Instrument Files** - when you operate the File Browser, all the files copied from the instrument are stored in this path. We recommend to check the occupied size and clear this folder from time to time.
- **Folder with Macro Script Files** - in this folder, all the macro scripts are stored. Change it to your liking.

## 17.2 16.2 Instrument Panels

Settings related to the *look-and-feel* of the Instrument Panel List (IPL). The chapter *Instrument Panel List* describes the IPL in more detail.



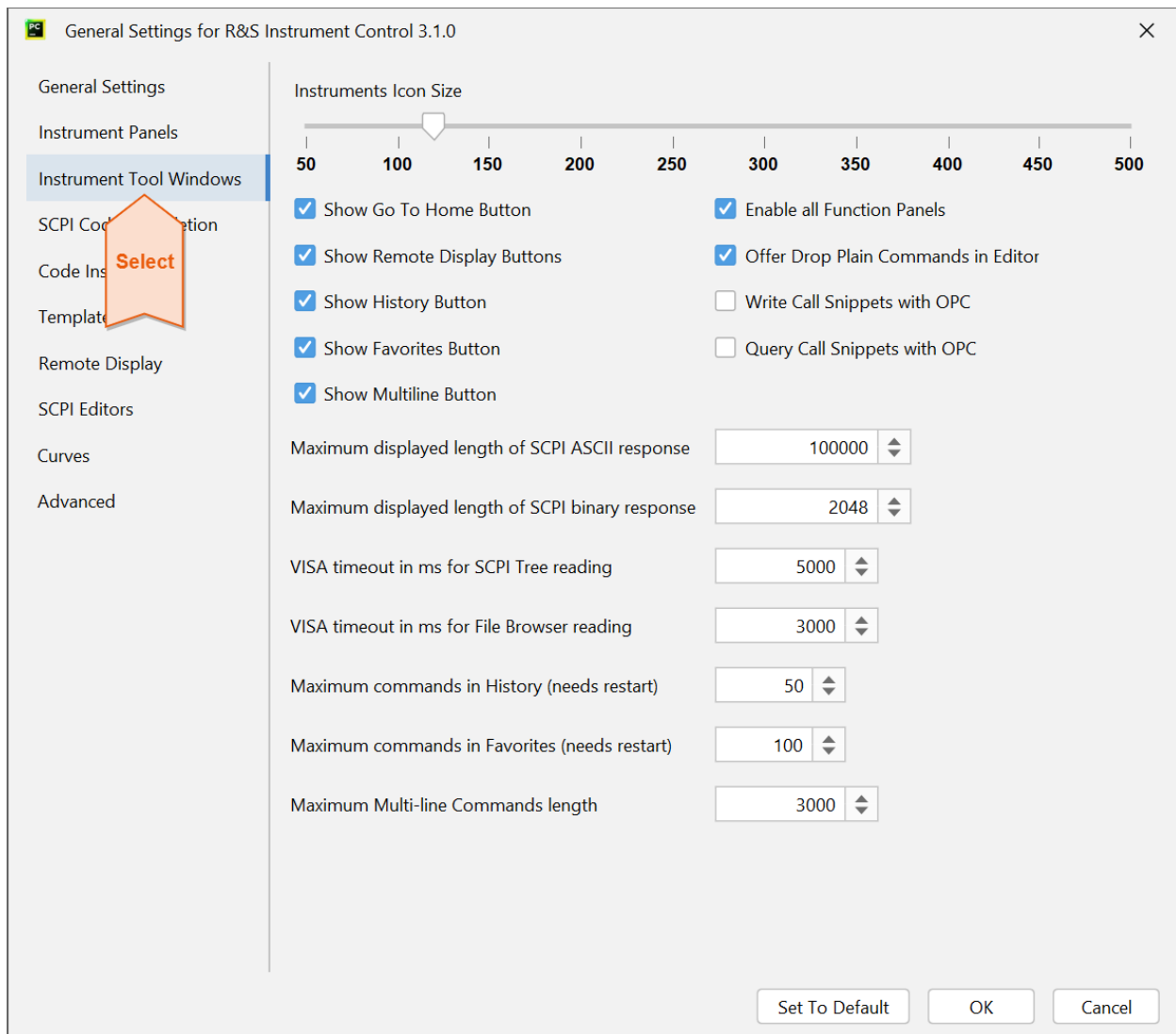
Description of the special controls:

- **Instruments Icon Size** - change the icon size either with the slider or by entering the direct number.
- **Maximum Description Length per Row** - limits the description width, so your panel can have compact width. Use this to prevent for example, long Resource Name strings to hide the action buttons.
- **Easy Remove Mode** - if checked, removing an instrument from the list works with one click without confirmation. The remove icon stays visible even for collapsed panels.

Other checkbox fields names correspond to the Instrument Panel elements. The changes have immediate effect, so you have immediate preview.

## 17.3 16.3 Instrument Tool Windows

Common settings for all the ITWs, that is, non instrument-specific.



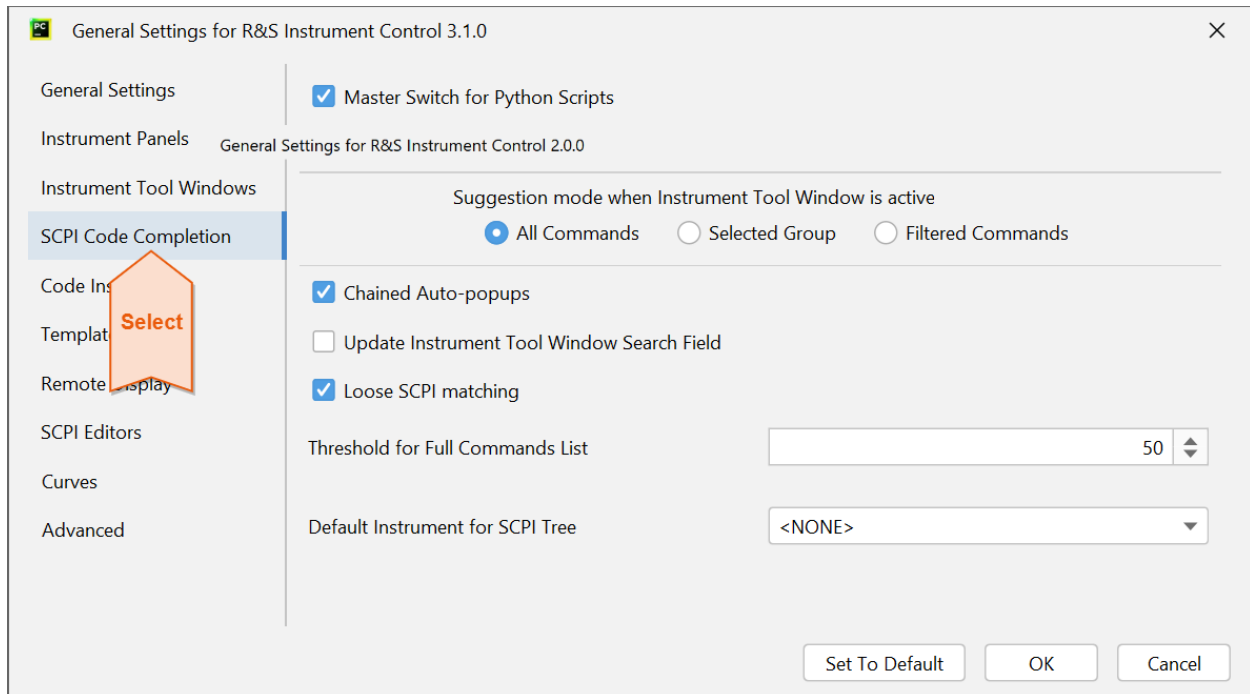
Description of the controls:

- **Instruments Icon Size** - change the ITW header icon size.
- **Show Go To Home Button** - if the instrument has a default web-site, the ITW shows a Home-button.
- **Show Remote Display Buttons** - if the Remote-display is enabled, the ITW shows its enabled buttons.
- **Show History Button** - shows button left of the SCPI Text Field to open the History commands window. You can invoke it also from the SCPI Text Field context-menu.
- **Show Favorites Button** - shows button left of the SCPI Text Field to mark commands as favorites or open the Favorites window. You can also invoke the window from the SCPI Text Field context-menu.
- **Show Multiline Button** - shows button left of the SCPI Text Field to open the Multi-line commands editor window. You can invoke it also from the SCPI Text Field context-menu or with the shortcut **F6**.
- **Enable All Function Panels** - overrides the internal table with the information which instrument supports which feature, and enables all the function panels. Use this on your own risk...
- **Offer Drop Plain Commands in Editor** - when you drag SCPI command(s) to your python editor, on drop you get the option to add them as plain strings. If you want to avoid that drop popup, uncheck this option.

- **Write Call Snippets with OPC** - if unchecked, all the write call snippets use the Write Command template. Check this to use the Write Command with OPC template`.
- **Query Call Snippets with OPC** - if unchecked, all the query call snippets use the Query Command template. Check this to use the Query Command with OPC template`.
- **Maximum Displayed Length of SCPI ASCII response** - if a text response from your instrument is longer than this number, the ITW's Response Field shows only the truncated length.
- **Maximum Displayed Length of SCPI binary response** - if a binary response from your instrument is longer than this number, the ITW's Response Field shows only the truncated length.
- **VISA Timeout in ms for SCPI Tree Reading** - maximum time the plugin waits for the instrument's response when reading the SCPI Tree. You can adjust it higher if your instrument needs more time to respond.
- **VISA Timeout in ms for File Browser Reading** - maximum time the plugin waits for the instrument's response when performing File Browser operations. You can adjust it higher if your instrument needs more time to respond.
- **Maximum Commands in History** - defines maximum amount of commands stored in the Commands History window. The oldest commands are then deleted. Needs restart.
- **Maximum Commands in Favorites** - defines maximum amount of commands stored in the Commands Favorites window. The oldest commands are then deleted. Needs restart.
- **Maximum Multi-line Commands length** - defines maximum length of the Multi-line SCPI Editor text (per bank) that is stored between the Pycharm runs.

## 17.4 16.4 SCPI Code Completion

Settings related to SCPI auto-completion.



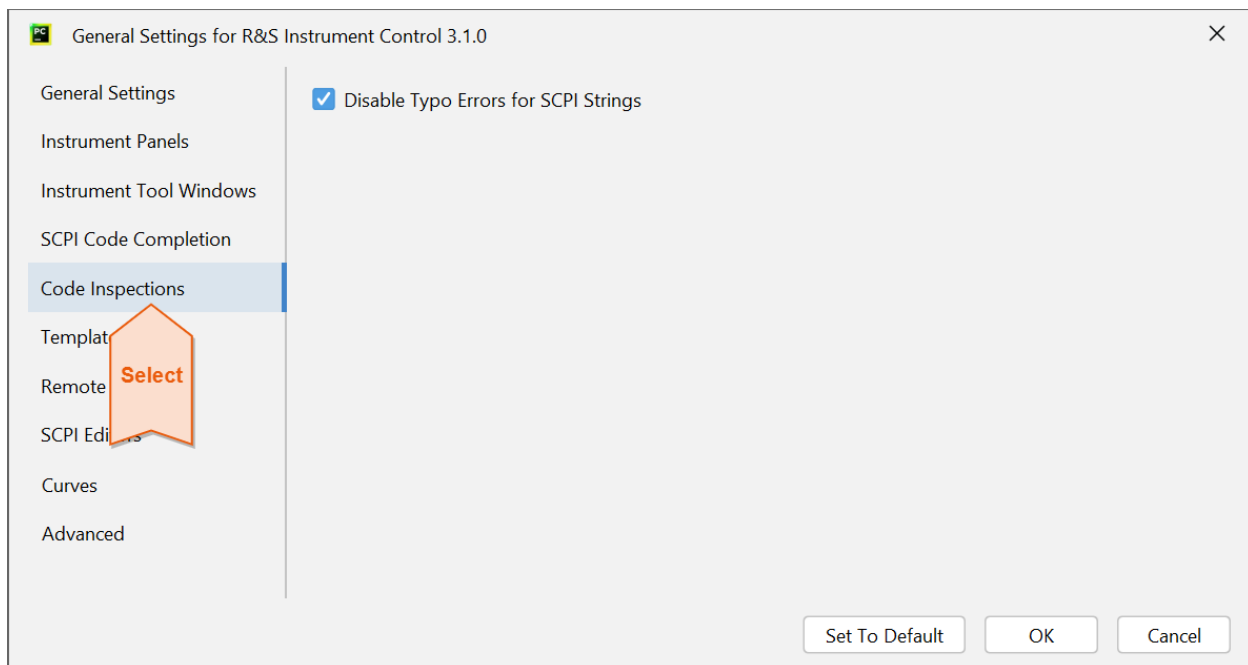
Description of the controls:

- **Master Switch for Python scripts** - switches the SCPI code completion On/Off in Python scripts (\*.py).

- **Master Switch for SCPI scripts** - switches the SCPI code completion On/Off for SCPI scripts (\*.scpi).
- **Suggestion Mode** - only has effect if the Instrument Tool Window (ITW) is active. If the ITW is inactive, the suggestion mode is always **All Commands**.
  - **All Commands** - always suggest all the available commands.
  - **Selected Group** - suggest only the commands from the currently selected group. If the instrument does not support group commands, this mode has the same effect as **All Commands**.
  - **Filtered Commands** - only suggest the commands that are currently visible in the SCPI Tree Function panel, also taking Filter feature into account.
- **Chained Auto-popups** - when the SCPI commands suggestion starts, the suggestion popups are invoked automatically until the command is completed.
- **Update ITW Search Field** - as the auto-completion composes the SCPI command, the *Find Field 5 in the ITW SCPI Tree* is updated, which triggers the SCPI Tree item navigation.
- **Loose SCPI matching** - set this to true, if you want the auto-completion to also work on SCPI commands that are missing optional parts.
- **Threshold for Full Commands List** - as you are completing a SCPI command, the amount of the possible suggestions decreases. When it falls below this threshold, the suggestions will contain not only the next header, but the full command up to the end.
- **Default Instrument for SCPI Tree** - there are cases, where you are might use script variables that do not fit any of your instrument's aliases. Such case is for example a common sub-routine serving more than one instrument. If you still want the SCPI code completion to work, you have to tell the RsIC which SCPI Tree to use. You can do it here by defining a default instrument.

## 17.5 16.5 Code Inspections

Settings affecting python code inspections and annotations. Currently only one feature is supported.



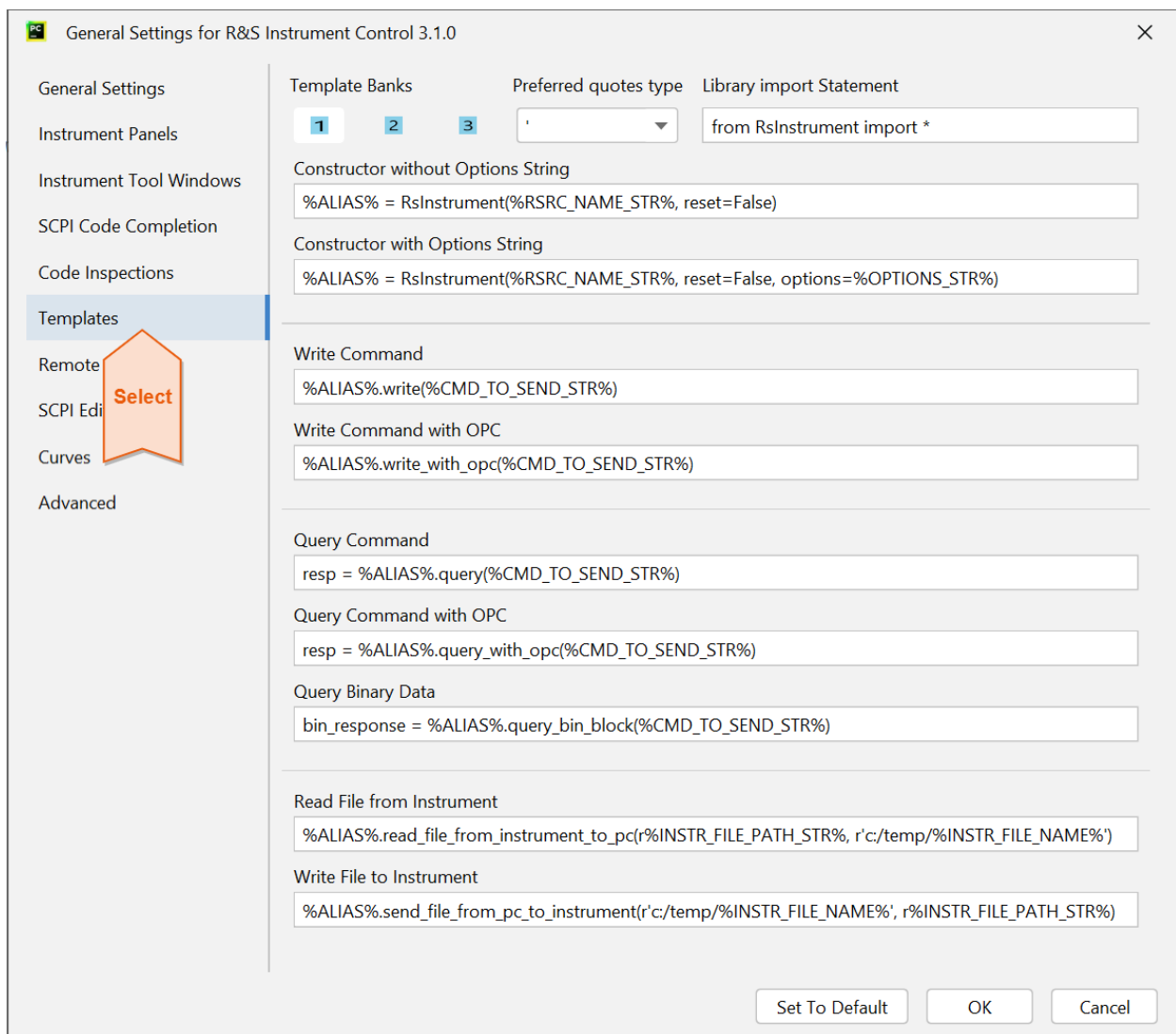
Description of the controls:

- **Disable Typo Errors for SCPI Strings** - removes the distracting green underlines for all the SCPI strings which highlight typo errors. The typo checking still works for other parts of your script.

## 17.6 16.6 Templates

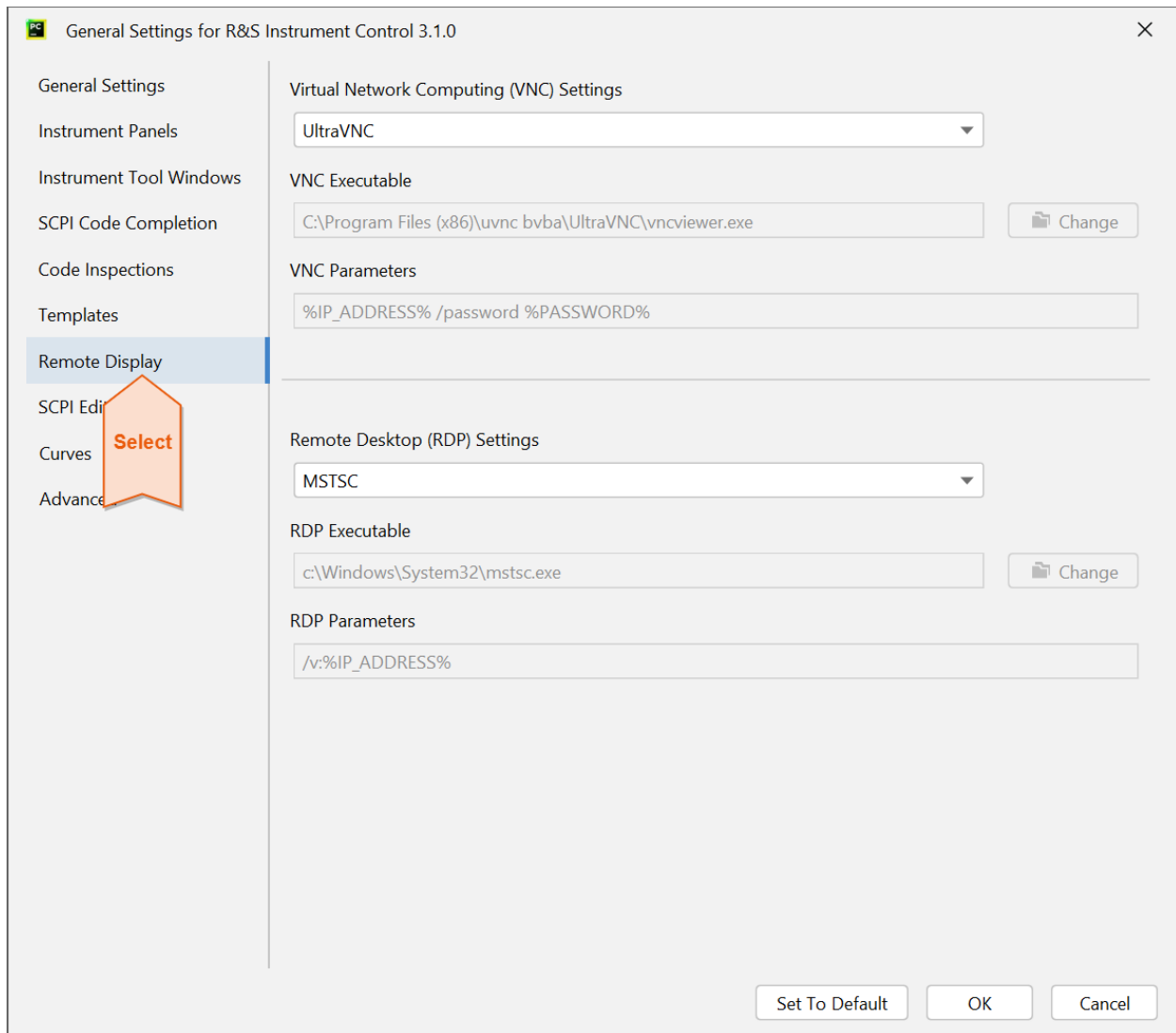
Here, you can customize all the paste snippets. For easier switching between different templates, you have 3 banks (presets) at your disposal. The one selected and visible is the active one. The Templates are split into 4 groups: General, Write, Query, File. For multi-line snippets, use the linefeed mark `<NEWLINE>`. For example, a query snippet with two separate write+read calls: `%ALIAS%.write(%CMD_TO_SEND_STR%);<NEWLINE>resp=%ALIAS%.read();`.

- General, Write and Query templates are used by the SCPI Communicator to import package, create the constructors, write and query commands.
- File Templates are used by the File Browser to insert code snippets for reading / writing files to the instrument.



## 17.7 16.7 Remote Display

Default Remote Display settings for VNC and RDP (Remote Desktop). You can customize them per-instrument basis.



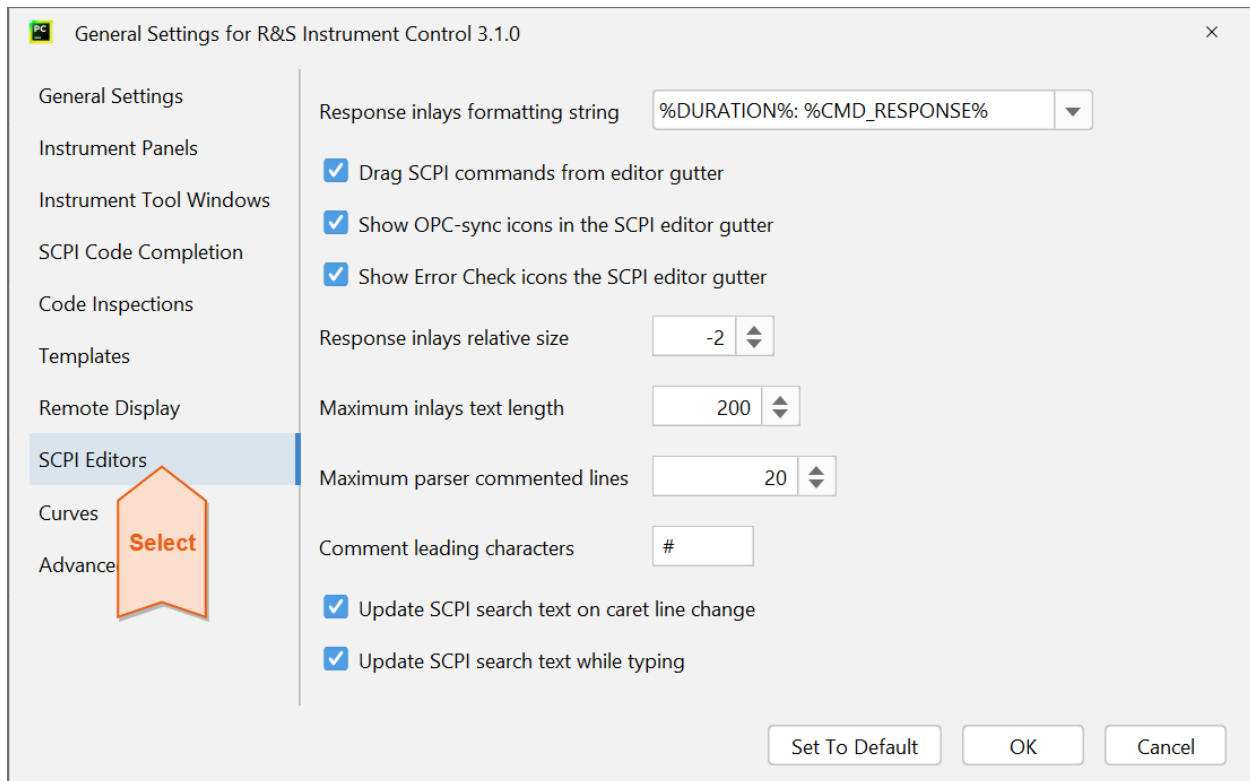
Remote Display control uses external applications that are launched through command line. They differ for Windows, Linux, and macOS. This description is for Windows OS:

- **VNC Settings** - you can choose standard settings for UltraVNC or TightVNC, or select custom settings for any VNC application you prefer.
- **RDP Settings** - for Windows, the only pre-configured application is Microsoft's MSTSC. Because of the security policy, you have to manually enter your credentials every time you open the RDP session.



## 17.8 16.8 SCPI Editors

Settings related to SCPI Editors that handle \*.scpi files.



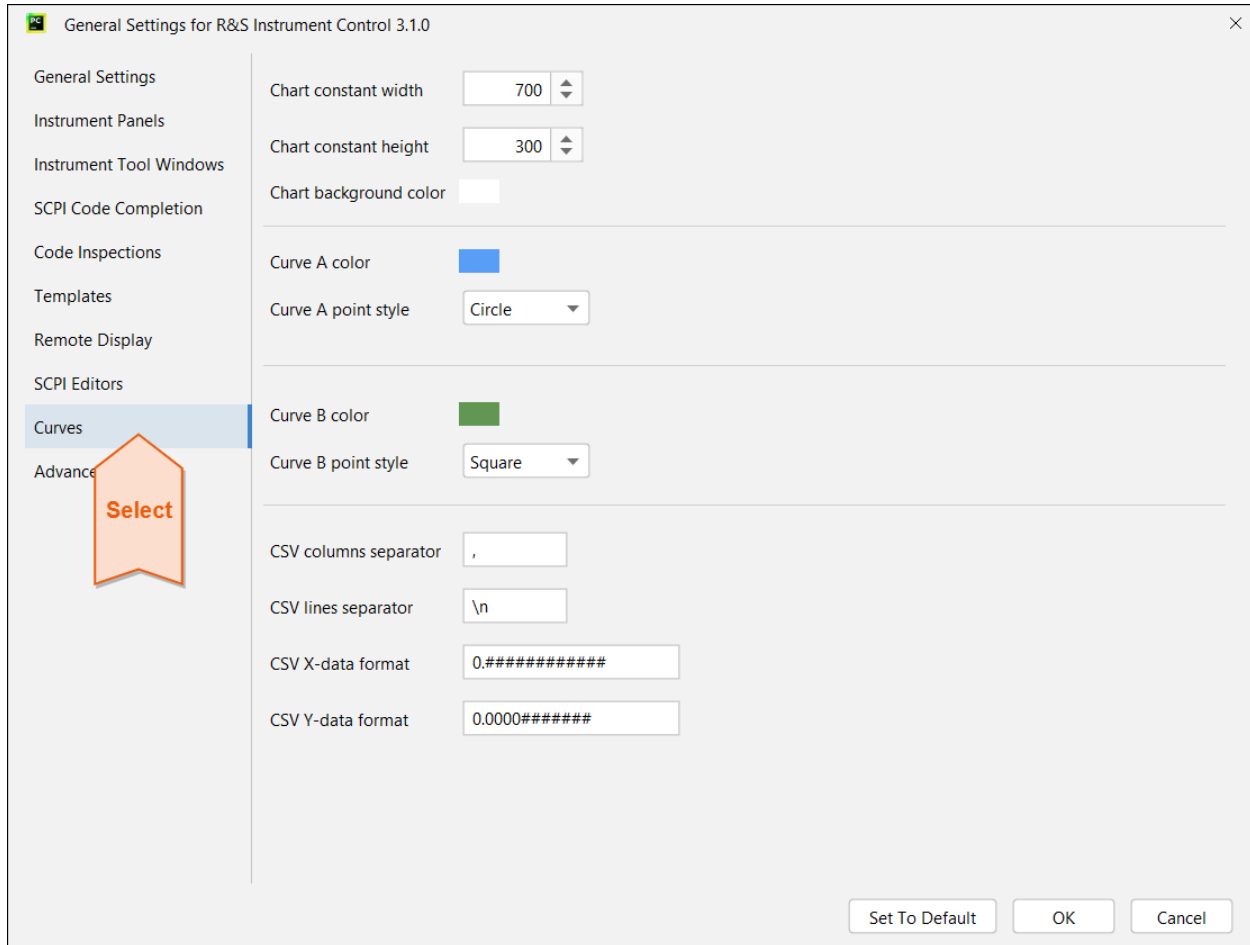
Description of the controls:

- **Response inlays formatting string** - select the format of the SCPI query response inlays. You can select the predefined formats or define your own.
- **Drag SCPI commands from editor gutter** - useful feature allowing to drag the selection or the actual line to another place.
- **Show OPC-sync icons in the SCPI editor gutter** - gutter icons show red sand clock icons for \*OPC commands, and green sand clock icons for \*OPC? queries.
- **Show Error Check icons the SCPI editor gutter** - gutter icons show blue icons for all error-checking commands.
- **Response inlays relative size** - relative font size of the inlay text. 0 means same size as the current editor font size.
- **Maximum inlays text length** - limit the length of the inlays.
- **Maximum parser commented lines** - in the SCPI parser, when you use the option to show non-parsed lines as comments, the parser shows maximum of this many lines.
- **Comment leading characters** - characters to use when you toggle comment/uncomment of line(s). Notice that this value contains one leading space by default.
- **Update SCPI search text on caret line change** - if the Search Window is active (shortcut F3), its search text follows the current editor caret position.

- **Update SCPI search text while typing** - if the Search Window is active (shortcut F3), its search text is updated as you type.

## 17.9 16.9 Curves

Common settings for the Curves Tool Windows:

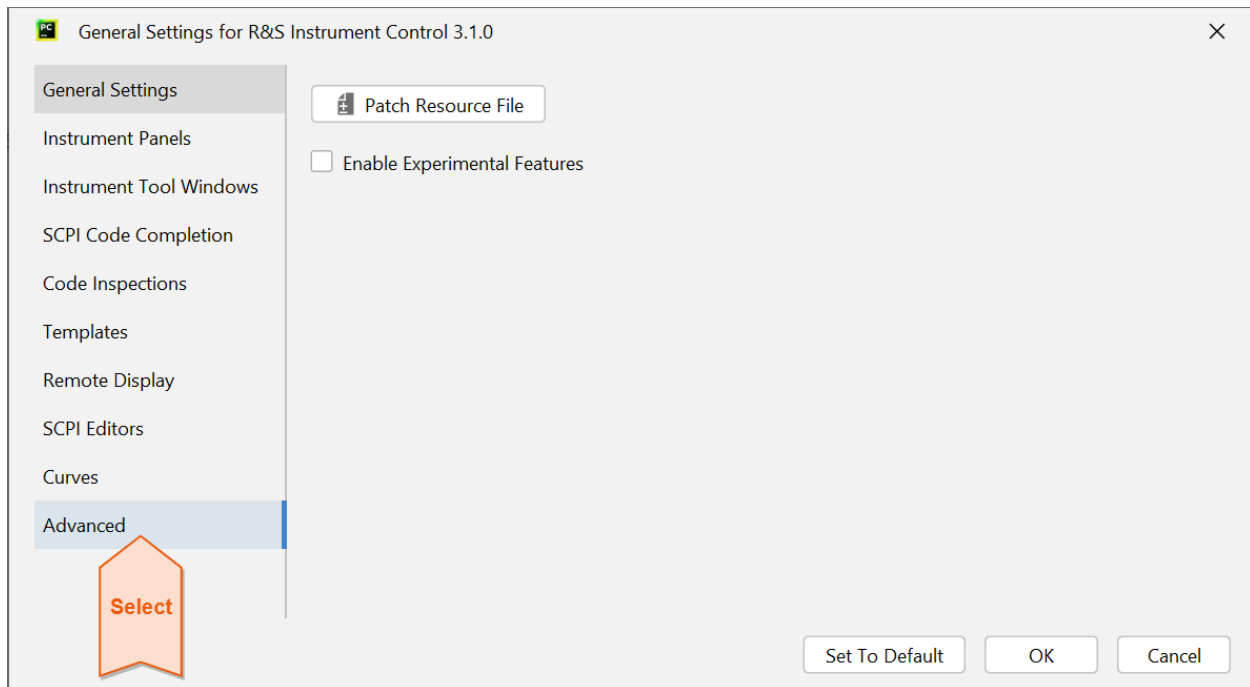


Description of the controls:

- **Chart constant width / height** - If you've selected 'Keep the chart size constant' in your Curves Tool Window, the chart keeps the dimensions defined here.
- **Chart background color** - by default it is white, even for Darcula Theme. Change it to your liking.
- **Chart A/B color + Style** - you can adjust the color and points style of curves A/B.
- **CSV columns/lines separator** - relevant, if you work with CSV export feature, by default the columns are separated by comma, and lines by Linefeed.
- **CSV X/Y-Data format** - relevant for CSV-data export. The formatting is Java DecimalFormat. A good explanation of the DecimalFormat can be found for example here: <https://jenkov.com/tutorials/java-internationalization/decimalformat.html> >`\_.

## 17.10 16.10 Advanced

Danger Zone!!! Tweaking the functionalities and patching feature definition files.





## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`