



EEZ Studio User manual

*Low-code embedded GUI development tool
T&M automation and management*

Ver. M14 DRAFT – 05/2023
www.envox.eu
github.com/eez-open

Table of Contents

1. Legal information.....	7
1.1. Definitions.....	7
1.2. Disclaimers.....	7
1.3. Miscellaneous.....	8
1.4. Contact information.....	8
1.5. Revision history.....	8
2. The EEZ Studio overview.....	9
2.1. Introduction.....	9
2.2. Main sections.....	9
2.3. Known issues and issue reporting.....	9
2.4. Donations.....	9
3. Installation.....	11
3.1. System requirements.....	11
3.2. Linux.....	11
3.3. Mac.....	11
3.4. Windows.....	11
3.5. Nix package manager.....	12
3.6. Build and run from source (all operating systems).....	12
3.6.1. Linux only.....	12
3.6.2. Raspbian only.....	12
3.6.3. All platforms.....	12
3.6.4. Raspbian.....	12
3.6.5. Nix.....	12
3.7. USB TMC.....	12
3.7.1. Windows.....	13
3.7.2. Linux.....	13
3.8. FAQ.....	13
4. Key features.....	15
4.1. General.....	15
4.2. EEZ Studio Project.....	15
4.3. EEZ Studio Instrument.....	15
5. Menu options and Settings.....	17
5.1. Home page.....	17
5.2. Menu options.....	17
5.2.1. File.....	17
5.2.2. Edit.....	18
5.2.3. View.....	18
5.2.4. Help.....	19
6. Home page instrument sections.....	21
6.1. History.....	21
6.2. Shortcuts and Groups.....	21
6.3. Notebooks.....	22
6.4. Items purge and restore.....	26
6.5. Instrument Extension (IEXT) Manager.....	27
6.6. Add instrument.....	28
6.7. Establishing a connection with the instrument.....	30
7. Instrument activity bar.....	33
7.1. Start page (EEZ BB3 only).....	33
7.2. Dashboard.....	35

7.3. Terminal.....	35
7.4. Scripts.....	36
7.4.1. Edit script shortcut.....	37
7.5. Shortcuts.....	38
7.6. Lists.....	39
7.6.1. Editing a list using a table.....	40
7.6.2. Editing a list using an envelope.....	41
7.6.3. List view options.....	43
7.6.4. List help.....	43
8. Home page project sections.....	45
8.1. EEZ Studio project types.....	45
8.2. Create new project.....	46
8.3. Additional steps for creating EEZ BB3 projects.....	47
9. Project editor overview.....	49
9.1. Project editor workspace.....	49
9.2. Display of the page in the editor.....	50
9.3. Panel moving and docking.....	50
9.4. Border tabssets.....	52
10. Project editor modes.....	53
10.1. Toolbar overview.....	53
10.2. Toolbar in Edit mode.....	54
10.3. Feature buttons.....	58
11. Project editor panels.....	59
11.1. Panel items.....	59
11.2. Right-click menu.....	60
11.3. Edit mode panels overview.....	61
11.3.1. Search and Replace.....	62
11.3.2. Find and Replace.....	62
11.4. Debug mode panels overview.....	63
12. Project editors/viewers.....	65
12.1. Editors.....	65
12.1.1. Page editor.....	65
12.1.2. User Actions.....	65
12.1.3. User Widgets.....	66
12.1.4. Font editor.....	66
12.1.5. Shortcuts (EEZ-GUI only).....	67
12.1.6. MicroPython (EEZ BB3 only).....	67
12.1.7. Readme.....	67
12.1.8. Settings.....	68
12.2. Viewers.....	69
12.2.1. Page viewer.....	69
12.2.2. Action viewer.....	70
13. EEZ Flow.....	71
13.1. EEZ Flow basic concepts.....	71
13.2. Flow execution.....	72
13.3. Flow examples.....	74
14. Project editing.....	75
14.1.1. Connecting Flow components.....	78
14.1.2. Copy & Paste between two projects.....	80
14.2. Working with Widgets.....	81
14.2.1. Widget component's items.....	81
14.2.2. Creating a User Widget.....	82

14.3. Working with Actions.....	84
14.3.1. Action component's items.....	84
14.3.2. Creating a User Action.....	85
15. Variables.....	87
15.1. Variables usage.....	88
15.2. Variable types.....	89
15.2.1. Basic/Primitive types.....	89
15.2.2. Structures.....	89
15.2.3. Enums.....	89
15.2.4. Objects.....	89
15.2.5. Arrays.....	89
15.2.6. Expressions.....	89
15.2.7. Literals.....	90
15.2.8. Binary Operators.....	90
15.2.9. Logical operators.....	92
15.2.10. Unary operators.....	92
15.2.11. Conditional (ternary) operator.....	93
15.3. Functions.....	93
15.3.1. System.....	93
15.3.2. Flow.....	93
15.3.3. Date.....	94
15.3.4. Math.....	96
15.3.5. String.....	97
15.3.6. Array.....	98
15.3.7. LVGL.....	99
15.4. Expression Builder.....	100

1. Legal information

1.1. Definitions

Draft – A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. Envox does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

1.2. Disclaimers

Limited warranty and liability – Information in this document is believed to be accurate and reliable. However, Envox does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. Envox takes no responsibility for the content in this document if provided by an information source outside of Envox.

In no event shall Envox be liable for any indirect, incidental, punitive, special or consequential damages (including – without limitation – lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, Envox' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of Envox.

Right to make changes – Envox reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use – Envox products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an Envox product can reasonably be expected to result in personal injury, death or severe property or environmental damage. Envox and its suppliers accept no liability for inclusion and/or use of Envox products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications – Applications that are described herein for any of these products are for illustrative purposes only. Envox makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using Envox products, and Envox accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the Envox product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

Envox does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using Envox products in order to avoid a default of the applications and the products or of the application or use by customer's 3rd party customer(s). Envox does not accept any liability in this respect.

Suitability for use in non-automotive qualified products – Unless this data sheet expressly states that this specific Envox product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. Envox accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without Envox' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer

uses the product for automotive applications beyond Envox' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies Envox for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond Envox' standard warranty and Envox' product specifications.

Security – Customer understands that all Envox products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their life cycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by Envox products for use in customer's applications. Envox accepts no liability for any vulnerability. Customer should regularly check security updates from Envox and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by Envox.

To report a security issue, use the EEZ Studio [issue tracker](#).

1.3. Miscellaneous

Open source license and contributions – EEZ Studio uses the *GPL v3* license. To view a copy of this license, please visit <https://www.gnu.org/licenses/gpl-3.0.html>. EEZ Studio uses the [C4.1 \(Collective Code Construction Contract\)](#) process for contributions.

This document is released under *open license FDL v1.3* from GNU.org. Therefore you are entitled to freely copy and redistribute it, with or without modifying it, either commercially or non-commercially. For additional details please consult the content of the [license](#).

Terms and conditions of commercial sale – Envox products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.envox.eu/company/terms-of-use/>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. Envox hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of Envox products by customer.

Translations – A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Export control – This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Trademarks – All referenced brands, product names, service names, and trademarks are the property of their respective owners.

1.4. Contact information

If you have any problem or requirement when using EEZ products or this manual, please contact Envox:

Discord server: <https://discord.gg/dhYMnCB>

E-mail: support@envox.eu

Website: www.envox.eu

1.5. Revision history

Date	Version	Changes
xxxx-xx-xx	1.0	Initial release

2. The EEZ Studio overview

2.1. Introduction

EEZ Studio was initially developed as a companion application for the in-house developed [EEZ H24005](#) programmable power supply and [EEZ BB3](#) T&M chassis to address two important tasks: a) remote programming and management and b) simplifying the development of a feature rich embedded GUI for a color touch-screen display.

The development was inspired by the idea of offering an open source alternative to some existing commercial solutions that are used for the mentioned tasks, all in order to overcome the limitations of their closed code, outdated and complex UI or sometimes awkward UX and licensing, which in our case was not in accordance with the open source of the mentioned devices that we have developed.

2.2. Main sections

EEZ Studio consists of two main sections, which are described separately in the manual:

- **Project** – creating, editing, debugging and building the code for the embedded GUI project for the selected target platform. Generated code can be directly imported into the IDE/toolchain used to build the firmware and accelerate the development process. It enables the rapid development of high quality embedded GUI and also comes with support for the open-source LVGL graphics library. The drag-and-drop editor makes it easy to utilize the many features such as widgets, animations, and styles to create a GUI reducing the coding effort. Additionally flowchart-like *EEZ flow* programming feature will further save development time and complexity.
- **Instrument** – allows access to one or more T&M instruments using several communication interfaces through which it is possible to manage and collect measurement data and screenshots using SCPI commands and queries. Collected data can be analyzed, searched, annotated and exported to other applications. Automation of test and measurement tasks using JavaScript and *EEZ flow* programming allows it to be used in different scenarios from basic development, calibration, troubleshooting and quality control using multiple devices from different manufacturers that can be in different locations connected to LANs.

In the introductory chapters of the two main sections that follow, all important features will be listed and described in detail.

2.3. Known issues and issue reporting

EEZ Studio is continuously developing and improving. A list of known issues can be found on [GitHub](#) where you are also invited to leave your suggestions for improvements and new functionality.

When reporting bugs using the GitHub tracking system, please first check if the issue you want to report has already been reported by someone else. When opening a new ticket, the following information can simplify and speed up the resolution:

- Descriptive/detail name of the issue (avoid general descriptions)
- Installed operating system version
- Installed EEZ Studio version
- Steps to reproduce the problem you are reporting

2.4. Donations

As an open source project, EEZ Studio has been largely developed thanks to donations primarily from [NLnet Foundation](#) as well as a number of smaller individual donors. If you want to contribute to further development with your donation, you can use [Liberapay](#).

3. Installation

3.1. System requirements

EEZ Studio is a 64-bit application. Therefore the minimum requirement for installation is a personal computer with a 64-bit operating system installed which has enough RAM and disk space for smooth operation.

Installation packages for supported operating systems for all versions of EEZ Studio are available for download at <https://github.com/eez-open/studio>

It is the official download page and we recommend that you get the latest version for the first installation. You will be able to check for future updates by using the option provided for that, as described below. If EEZ Studio becomes available on the websites of our partners, this information will be published on the Envox official website.

3.2. Linux

Depending on your linux distribution, choose one of the listed packages (.deb, .rpm) and start the installation using the associated installer.

In addition, there is a self-executing .AppImage version that, after downloading, needs to enable the Allow executing file as program option under file Permissions (Fig. 1) before starting it.

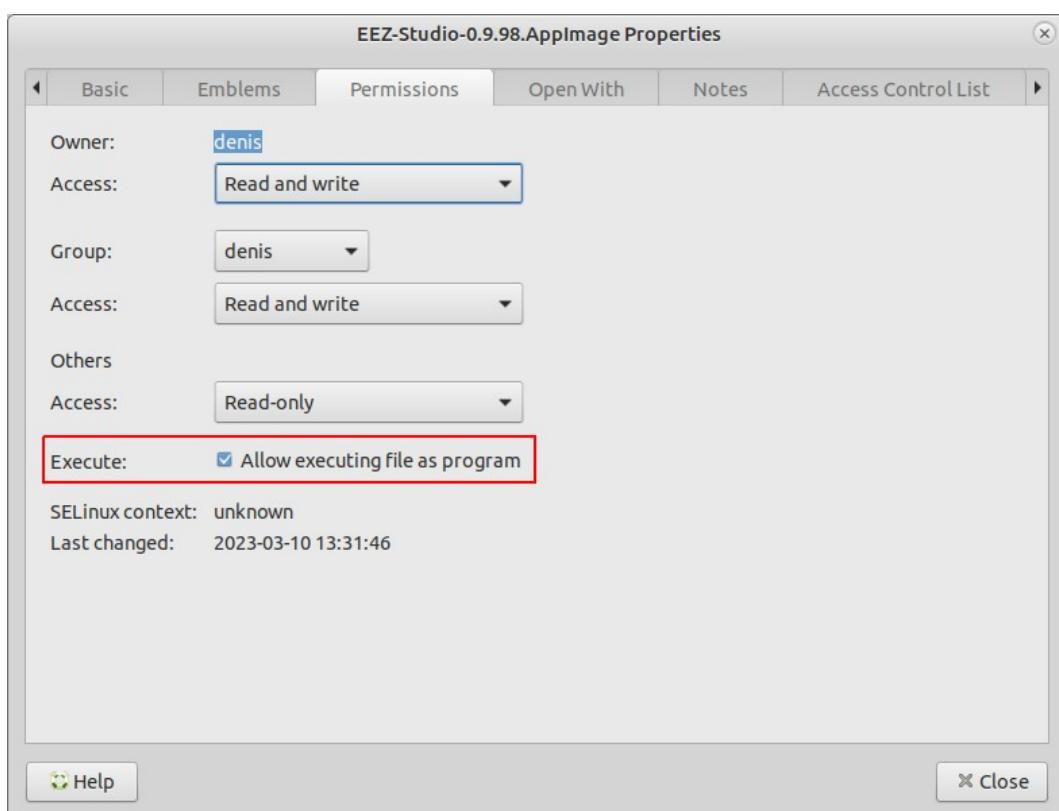


Fig. 1: .AppImage file permission

If you encounter a problem running the .AppImage version on your Linux distribution, try running it using the --no-sandbox option: ./EEZ-Studio-[version].AppImage --no-sandbox

3.3. Mac

Required OS version: macOS 10.10 (Yosemite) or newer

Download *eezstudio-mac.zip*, unpack and move *eezstudio.app* to Applications.

3.4. Windows

Required OS version: Windows 7 (64-bit) or newer

Download and start *EEZ_Studio_setup.exe*.

3.5. Nix package manager

The Nix [flake](#) provides a derivation for EEZ Studio or an overlay that provides that derivation. It can be used to install the project using [Nix package manager](#).

3.6. Build and run from source (all operating systems)

In addition to using ready-made installation packages, it is possible to build and run EEZ Studio directly from the source code located in the GitHub repository. Below is the procedure to be followed:

- Install *Node.JS 14.x or newer*
- Install *node-gyp*, more information at <https://github.com/nodejs/node-gyp#installation>

3.6.1. Linux only

```
sudo apt-get install build-essential libudev-dev
```

3.6.2. Raspbian only

Install *Node.js 16* and *npm* on Raspberry Pi: <https://linode.com/install-node-js-and-npm-on-raspberry-pi/>

```
sudo apt-get install build-essential libudev-dev libopenjp2-tools ruby-full  
sudo gem install fpm
```

3.6.3. All platforms

In the folder where you want to build the project, it will be necessary to clone the GitHub project repository, and start project building as follows:

```
git clone https://github.com/eez-open/studio  
cd studio  
npm install  
npm run build
```

Start with:

```
npm start
```

Create distribution packages (except [Raspbian](#)):

```
npm run dist
```

3.6.4. Raspbian

```
npm run dist-raspbian
```

3.6.5. Nix

To build:

```
nix build 'github:eez-open/studio'
```

To start:

```
nix run 'github:eez-open/studio'
```

3.7. USB TMC

The USB TMC driver must be installed if you want to access the T&M instrument using the USB-TMC interface from EEZ Studio *Instrument* section.

3.7.1. Windows

Download and start [Zadig](#). Select your device, select libusb-win32 and press “Replace Driver” button:

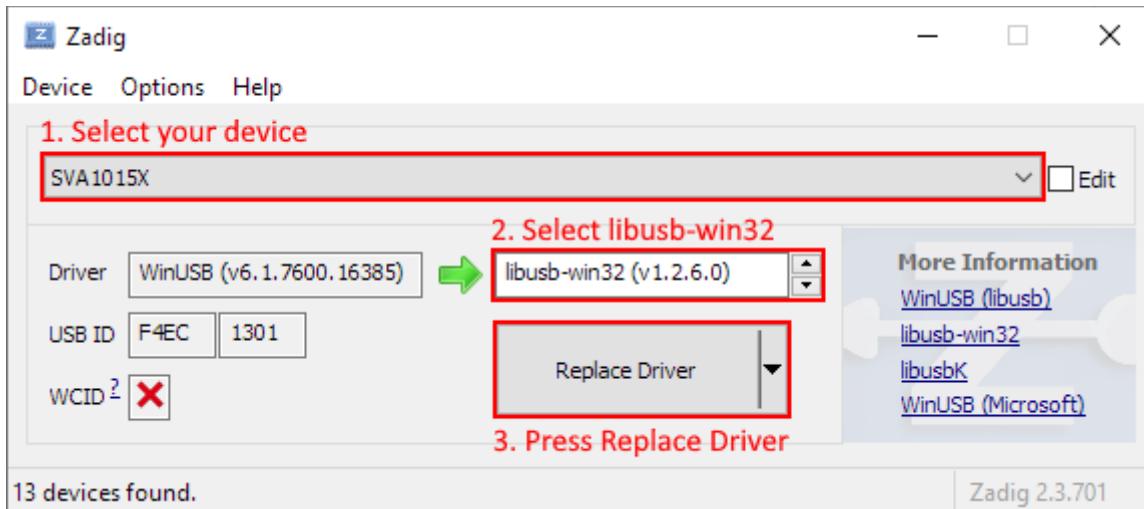


Fig. 2: Zadig driver settings

3.7.2. Linux

You will probably need to add your Linux account to the `usbtmc` group before you can access the instrument using EEZ Studio. Connect your instrument with a USB cable and turn it on. Wait until booting is complete. Now check the instrument group name by entering the following command:

```
ls -l /dev/usbtmc*
```

In case it is *root*, enter the command:

```
sudo groupadd usbtmc
```

Now, add your account (<username>) to the group:

```
sudo usermod -a -G usbtmc <username>
```

A reboot is required. After that, the *gid* of `/dev/usbtmc0` should be set to `usbtmc` and you are ready to use your instrument via USB-TMC interface.

3.8. FAQ

Q: Where is the database file by default?

A: Depending on the operating system, it can be:

- **Linux:** `~/.config/eezstudio/storage.db`
- **Mac:** `~/Library/Application\ Support/eezstudio/storage.db`
- **Windows:** `%appdata%\eezstudio\storage.db`

The default created database as well as its location can be changed later through the options in the *Settings* section of EEZ Studio.

Q: Where are the IEXTs (Instrument EXTensions) used to access T&M instruments stored?

A: Depending on the operating system, it can be:

- **Linux:** `~/.config/eezstudio/extensions`
- **Mac:** `~/Library/Application\ Support/eezstudio/extensions`
- **Windows:** `%appdata%\eezstudio\extensions`

4. Key features

4.1. General

- Modern and attractive UI/UX developed in [Electron](#)
- Light / Dark theme
- Multi-tab support for faster navigation
- Cross-platform run-time (Linux, Windows, macOS)
- Modular design based on plug-ins that can be added/removed depends of scope of the work
- Source/Version control integration ([GitHub](#) and [gitea.io](#))
- Open source project

4.2. EEZ Studio Project

- Modular visual development environment for rich embedded GUI (small display/limited resources) and desktop GUI
- *EEZ Flow*, low-code flowchart programming for both rapid prototyping and creation of complex applications
- [LVGL](#) (Light and Versatile Graphivs Library) support
- Generate C++ code for embedded GUI functionality that can be directly included in [STM32CubeIDE](#) for EEZ BB3 and other STM32 target platforms or [Arduino IDE](#) for EEZ H24005 and other Arduino compatible target platforms
- *Instrument definition file* (IDF) builder with context sensitive SCPI commands help (based on Keysight's [Offline Command Expert command set](#) XML structure) suitable for EEZ Studio *Instrument* and [Keysight Command Expert](#)
- SCPI command help generator based on bookmarked HTML generated directly from .odt file using [EEZ WebPublish](#) extension for OpenOffice/LibreOffice.
- Project templates (using giteo.io repositories) and comparison of projects
- Drag&drop editor for creating instrument's desktop dashboard (for remote control and management)

4.3. EEZ Studio Instrument

- Dynamic environment where multiple instruments can be configured and easily accessed
- Session oriented interaction with each SCPI instrument
- Serial (via USB), Ethernet and VISA (via free [R&S®VISA](#)) T&M instrument interfaces support
- Direct import of EEZ Studio generated IDFs and Keysight's Offline Command Expert command sets
- IEXT (Instrument EXTension) catalog with growing number of supported instruments (Rigol, Siglent, Keysight, etc.)
- History of all activities with search/content filtering
- Quick navigation via calendar ("heatmap") or sessions list view
- Shortcuts (hotkeys and buttons) that can be user defined or come predefined from imported IDF. The shortcut can contain single or sequence of SCPI commands or Javascript code.
- Javascript code for task automation (e.g. logfile, or programming list upload/download, etc.) can be also assigned to the shortcut
- SCPI commands context sensitive help with search
- File upload (instrument to PC) with image preview (e.g. screenshots)
- File download (PC to instrument) automation for transferring instrument profiles
- Simple arbitrary waveform editor (envelope and table mode)
- Displaying measurement data as graphs
- FFT analysis, harmonics and simple math functions (Period, Frequency, Min, Max, Peak-to-Peak, Average)
- Export graphs as .CSV file

5. Menu options and Settings

5.1. Home page

After starting EEZ Studio, the home page is displayed, which is actually the *Home* tab that is always present (it cannot be hidden). *Main tabs* section (1) allows easy navigation between multiple open projects, instruments as well as *Extension Manager* and *Settings* sections (Fig. 3).

The main sections of EEZ Studio are *Extension Manager*, *Settings* which are accessible from the *Main option bar* (2), while the *Projects* (3) and *Instruments* (4) sections are positioned below and have their own option bars whose options when selected also appear in the *Main tabs* section.

The *Projects* section will be described in detail in chapters xx to xx, and the *Instruments* section in chapters xx to xx.

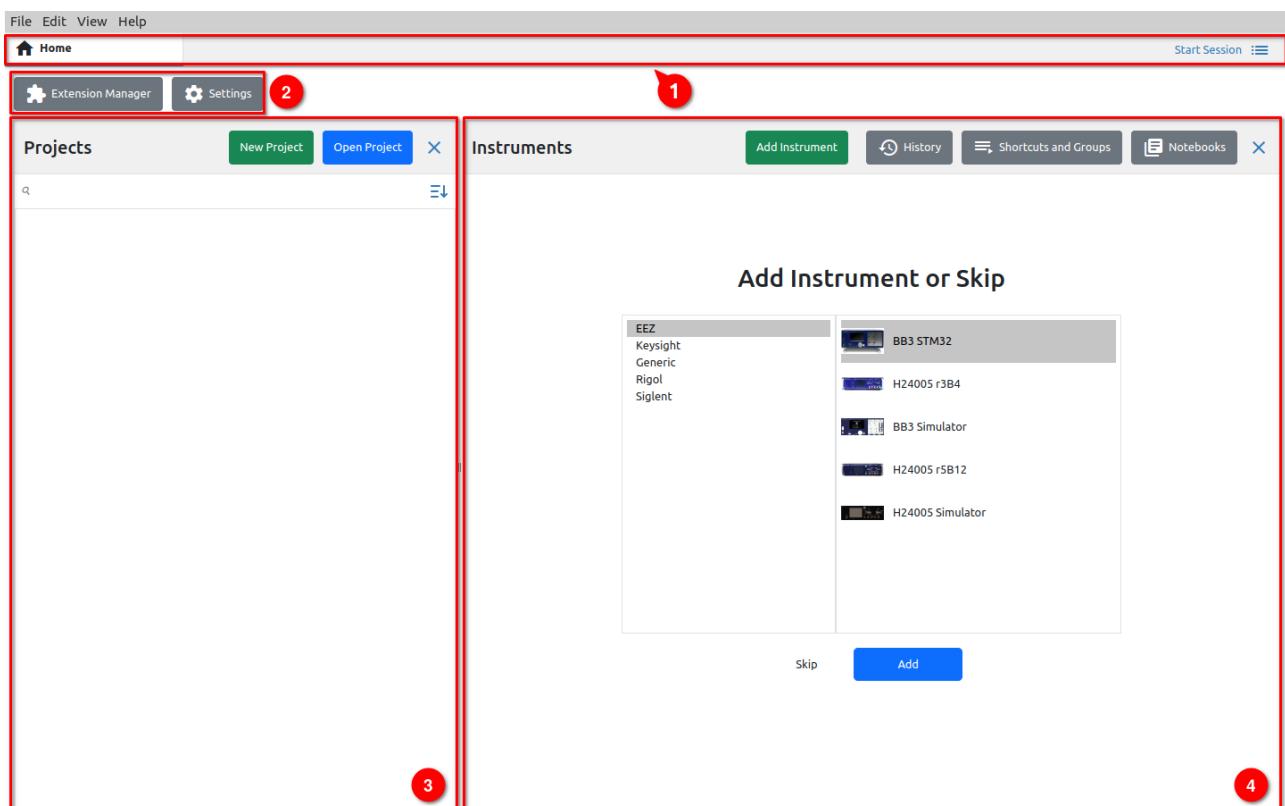


Fig. 3: Home page

5.2. Menu options

Menu options available from all main sections of EEZ Studio are listed below.

5.2.1. File

Option	Shortcut	Description
New project...	CTRL + N	Creates a new project.
Add instrument...	ALT + CTRL + N	Adds an instrument to the EEZ Studio workbench that can be controlled.
New Window	CTRL + SHIFT + N	Opens a new copy of the window.
Open...	CTRL + O	Opens an existing project.
Open Recent	-	List of recently opened projects.
Import Instrument Definition...	-	Import IEXT (Instrument EXTension) file.
Save	CTRL + S	Saving project files.

Exit - EEZ Studio shutdown.

5.2.2. Edit

Option	Shortcut	Description
<i>Undo</i>	CTRL + Z	Undo previous action.
<i>Redo</i>	CTRL + Y	Redo previous action.
<i>Cut</i>	CTRL + X	Move content to Clipboard.
<i>Copy</i>	CTRL + C	Copy content to Clipboard.
<i>Paste</i>	CTRL + V	Paste content from Clipboard.
<i>Delete</i>	DEL	Delete selected content.
<i>Select All</i>	CTRL + A	Select all content.

5.2.3. View

Option	Shortcut	Description
<i>Home</i>	-	Return to the <i>Home</i> tab.
<i>History</i>	-	Opening the Instrument's <i>History</i> tab.
<i>Shortcuts and Groups</i>	-	Opening the Instrument's <i>Shortcuts and Groups</i> tab.
<i>Notebooks</i>	-	Opening the Instrument's <i>Notebooks</i> tab.
<i>Extension Manager</i>	-	Opening the Instrument's <i>Extension Manager</i> tab.
<i>Settings</i>	-	Opening the <i>Settings</i> tab (Fig. 4).
<i>Toggle Full Screen</i>	F11	View EEZ Studio in full screen (select F11 again to restore).
<i>Toggle Developer Tools</i>	CTRL + SHIFT + I	Opening the developer tools in the right part of the window.
<i>Switch to Dark Theme</i>	CTRL + SHIFT + T	Toggle between Light and Dark theme.
<i>Zoom In</i>	CTRL + +	Zoom in (enlargement) of all screen elements. On some Linux distributions you will need to use CTRL + SHIFT + + as a shortcut.
<i>Zoom Out</i>	CTRL + -	Zoom out (reduction) of all screen elements.
<i>Reset Zoom</i>	CTRL + 0	Returning the zoom to the default level.
<i>Reload</i>	-	Reload all content.

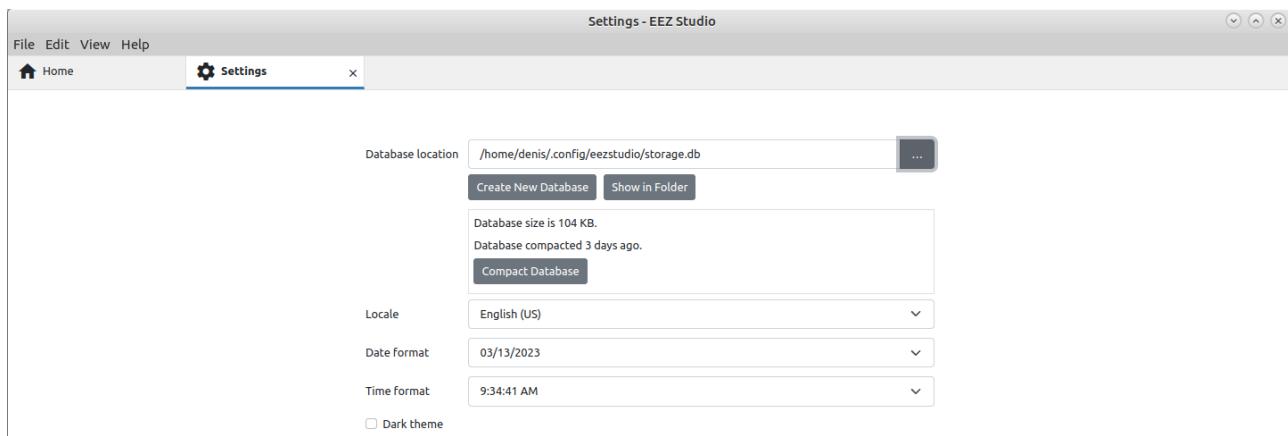


Fig. 4: Settings tab

Database location

A database is used to store the data collected in communication with the instruments. An empty base

is created at first launch and its location can be seen here. You can also change the location here to one of the existing databases (backup, imported from another EEZ Studio, etc.).

Changing the parameters of the database requires a restart of EEZ Studio. The Restart button will be displayed in the lower right corner.

Create New Database

Creating a new database with the name and location you specified.

Show in Folder

View the folder where the database is located.

Locale

Defines the date and time formats for the selected country.

Changing the Locale requires a restart of EEZ Studio. The Restart button will be displayed in the lower right corner.

Date format

Display format of all date values.

Time format

Display format of all time values.

Dark theme

Toggle between Light and Dark theme (same as shortcut CTRL + SHIFT + T).

5.2.4. Help

Option	Shortcut	Description
About	-	Opens the EEZ Studio version information (Fig. 5).

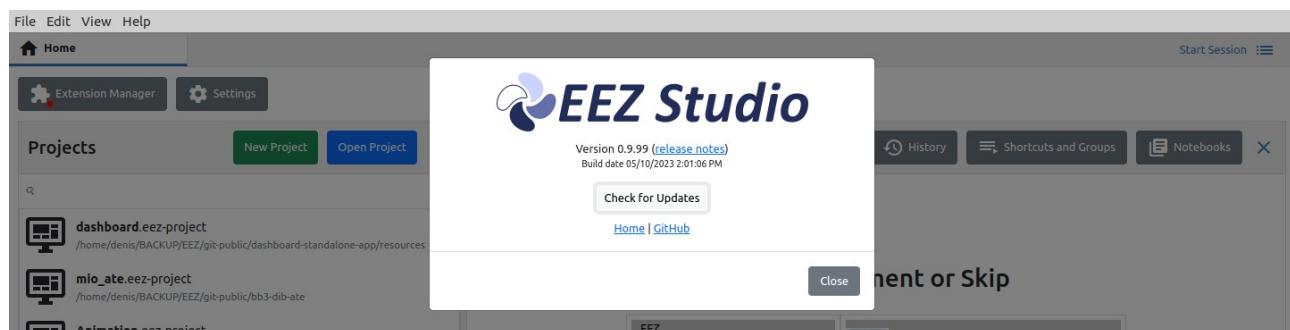


Fig. 5: About page

Check for Updates

This function requires an internet connection in order for EEZ Studio to connect to the GitHub repository and check for a newer version than the one installed.

This function does not take into account versions that have a pre-release status, but only released versions.

Home

Opens the home page of the Envox official site (requires internet browser installed).

Github

Opens Envox's GitHub home page (requires internet browser installed).

*EEZ Studio
Instrument*

6. Home page instrument sections

The top of the home page contains general options (1) for working with instruments (Fig. 6). Instrument specific *History*, *Shortcuts and Groups* and *Notebooks* options can also be accessed through the *Instruments action bar* for the currently selected instrument as described below. The Instruments section can be optionally hidden (2) when the Show Instruments option appears (Fig. 7).

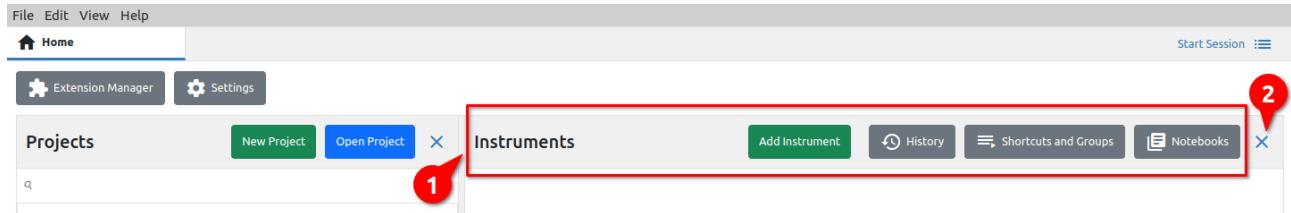


Fig. 6: Home page instrument options



Fig. 7: Home page "Show instruments" option

6.1. History

History displays communication via the *Terminal* option for all instruments in one place. In this way, it will be easier to search all activities as well as to add notes, files and graphs in the same way as in the *Terminal* of the currently selected instrument, as will be described below.

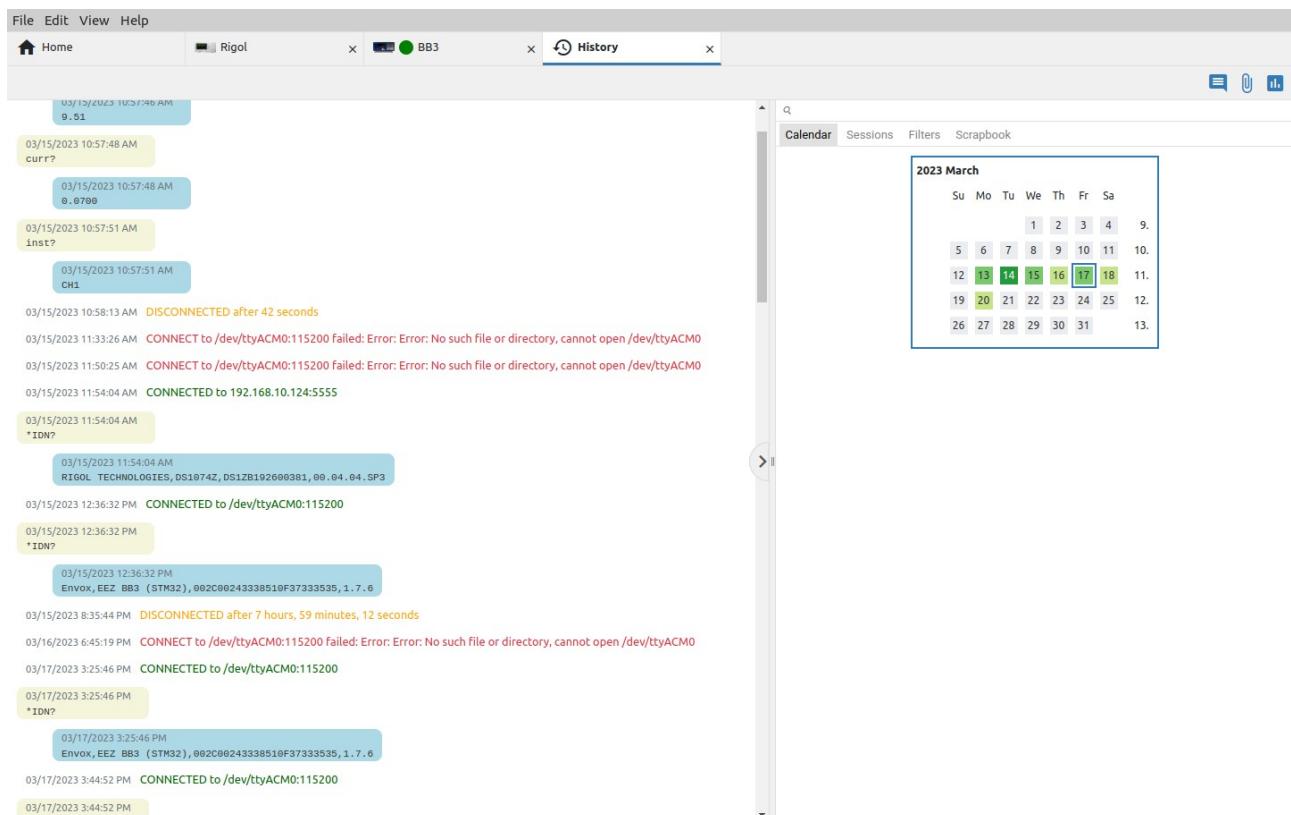


Fig. 8: Instruments History view

6.2. Shortcuts and Groups

Just like with *History*, *Shortcuts and Groups* is not a system feature, but only displays the available shortcuts and their groups in one place for easier searching, editing, deleting and adding new shortcuts and their groups.

Therefore, all operations with shortcuts on this page are possible as via the *Shortcuts* page of the currently selected instrument, which will be described below.

EEZ Studio User manual

Name	Group / Extension	Keybinding	Action	Confirmation	Toolbar	Toolbar position
Abort	EEZ BB3 STM32 EEZ BB3 Simulator EEZ H24005 r3B4	F9	SCPI	✓		9
Clear protections	EEZ BB3 STM32 EEZ BB3 Simulator	F10	SCPI	✓	✓	10
Clear protections	EEZ H24005 r3B4	F10	SCPI	✓	✓	10
Coupling	EEZ BB3 STM32 EEZ BB3 Simulator	F5	JavaScript	✓		5
Dlog abort	EEZ BB3 STM32 EEZ BB3 Simulator EEZ H24005 r3B4	—	SCPI	✓		15
Dlog start	EEZ BB3 STM32 EEZ BB3 Simulator	—	JavaScript	✓		13
Dlog start	EEZ H24005 r3B4	—	JavaScript	✓		13
Dlog upload	EEZ BB3 STM32 EEZ BB3 Simulator	—	JavaScript	✓		14
Dlog upload	EEZ H24005 r3B4	—	JavaScript	✓		14
Init	EEZ BB3 STM32 EEZ BB3 Simulator EEZ H24005 r3B4	F8	SCPI	✓		8
Outputs OFF	EEZ BB3 STM32 EEZ BB3 Simulator	F1	SCPI	✓		1
Outputs OFF	EEZ H24005 r3B4	F1	SCPI	✓		1
Outputs ON	EEZ BB3 STM32 EEZ BB3 Simulator	F2	SCPI	✓		2
Outputs ON	EEZ H24005 r3B4	F2	SCPI	✓		2

Fig. 9: Instruments Shortcuts and Groups view

6.3. Notebooks

The *Notebooks* feature enables data collected from one or more sources (instruments) to be stored and presented in one place. Data stored in this way can be searched as if they belonged to a single source. Notebooks can also be appended, exported and imported, which facilitates the exchange of collected data.

Figure 10 shows the 'Instrument Notebooks' view in EEZ Studio. The interface includes:

- Toolbar: Includes icons for New (1), Open (2), Save (3), and Print (4).
- Left Panel: Shows a list of notebooks, with 'bb3 notebook test' and 'test notebook' currently selected.
- Central Area: Displays three data snippets from the notebook:
 - 03/13/2023 10:47:03 AM: Source: RIGOL TECHNOLOGIES,DS1074Z,DS1ZB192600381,00.04.04.SP3
 - 03/13/2023 10:47:03 AM: Source: RIGOL TECHNOLOGIES,DS1074Z,DS1ZB192600381,00.04.04.SP3
 - 03/13/2023 10:47:06 AM: Source: RIGOL TECHNOLOGIES,DS1074Z,DS1ZB192600381,00.04.04.SP3
- Bottom Area: A screenshot of the Rigol DS1074Z oscilloscope interface, showing waveforms and configuration parameters.
- Right Panel: A calendar for March 2023, with the 13th highlighted.

Fig. 10: Instrument Notebooks view

Option

1 Add / Import notebook

Description

Create a new blank notebook or import a notebook file. When creating a new notebook, you will need to enter a name. To import data into a notebook, use the Notebook option in the instrument's *Terminal*, as shown in Fig. 12: (1) go to the *Terminal* tab in the *Action bar*, (2) select one or more items and (3) export them to a notebook file, a new notebook or an already created notebook.

In the case of exporting to a file, it will be necessary to choose a destination on the local storage, and in the case of exporting to a new notebook, the name of the notebook should be entered.

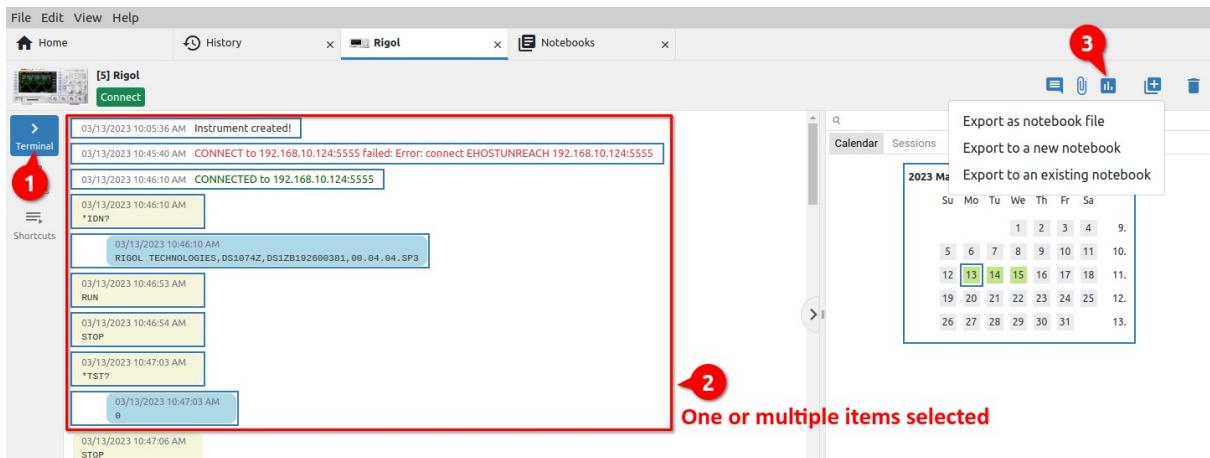


Fig. 11: Adding items to the notebook

2 Remove notebook

Remove the notebook from the list.

3 Change notebook name

Change notebook name.

4 Show deleted notebooks

Notebooks that have been removed from the list are not immediately deleted from the database. This option enables the display of all notebooks (Fig. 12) that have been removed from the list and offers the possibility to restore (return to the list) or permanently delete the notebook.

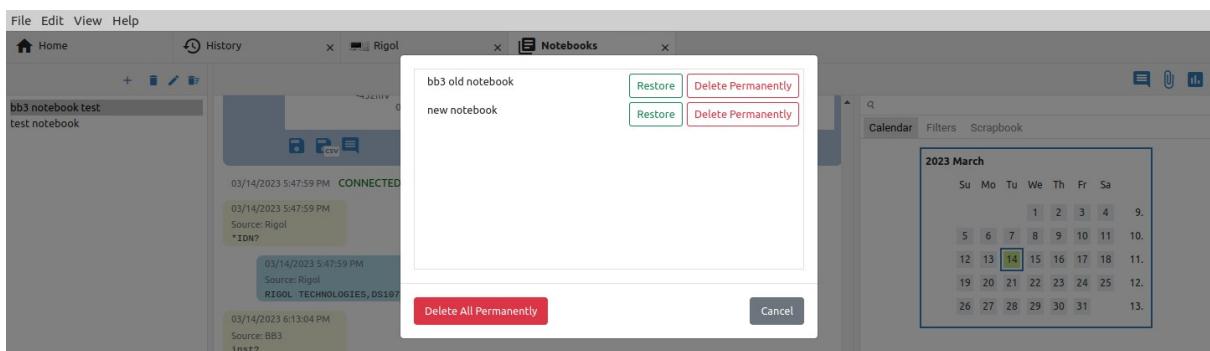


Fig. 12: Deleted notebooks

5 Add note

Adding a note to the notebook (Fig. 13). The number of notes is not limited and the last added note will appear at the bottom of the notebook (Fig. 14).

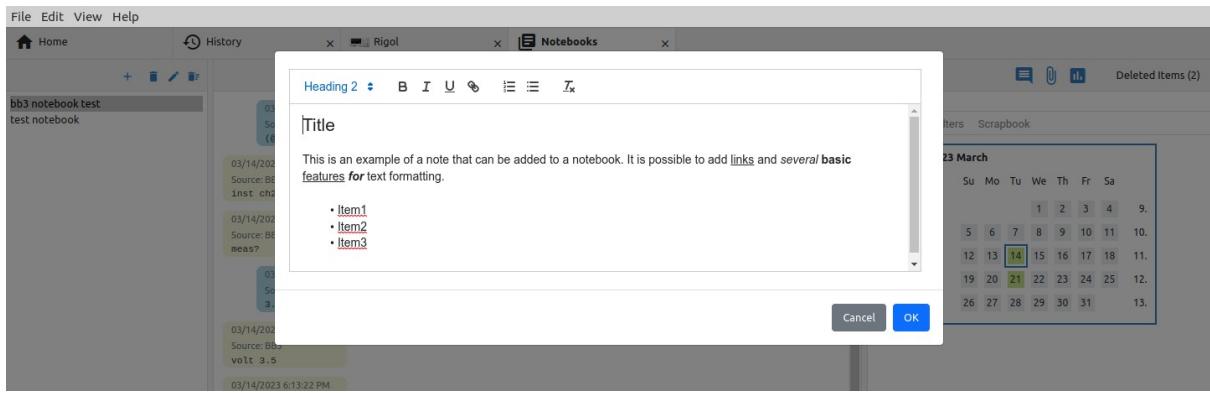


Fig. 13: Adding a new note to the notebook

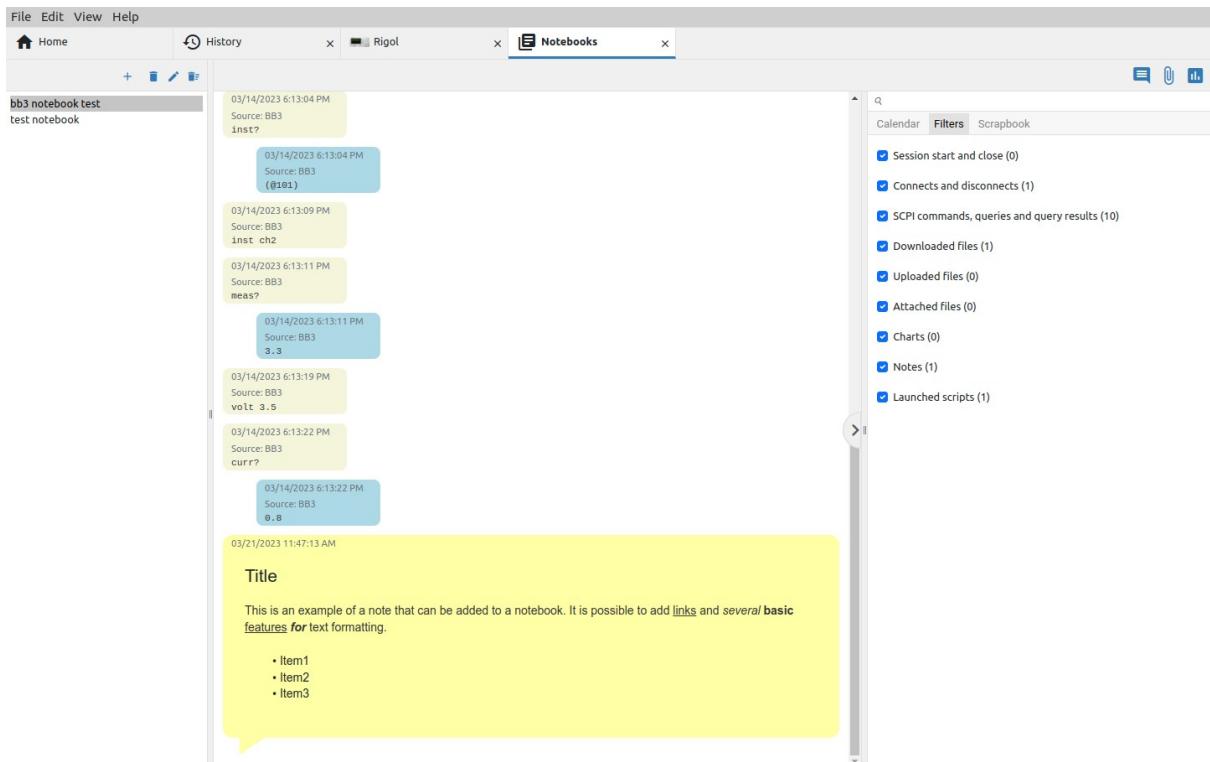


Fig. 14: Newly added note in the notebook

6 Attach file

Different files from local storage can be added to the notebook. In this way, all relevant data collected with the instruments can be combined together with images, recordings, datasheets into a whole that can be searched and further shared.

All imported files are marked with a paper clip icon in the upper left corner. It also displays the full path from where the file was imported as well as its size (Fig. 15).

Files whose format EEZ Studio can recognize (.jpeg, .png, etc.) also have a preview. Such files, in addition to the option to save to local storage and to add a note, will also have the option to copy to the clipboard.

6. Home page instrument sections

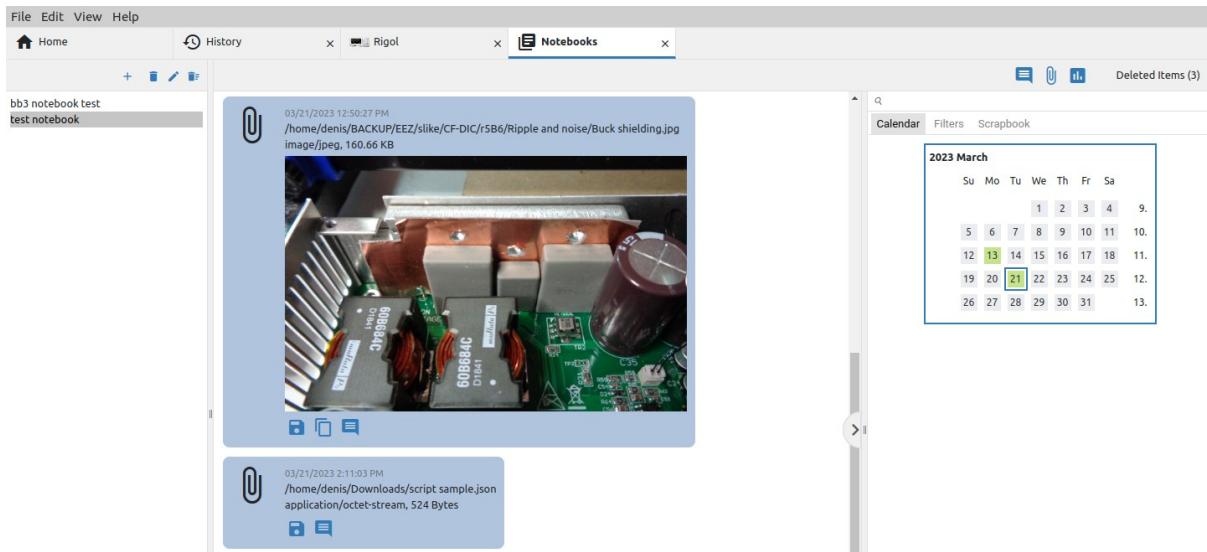


Fig. 15: Files imported into the notebook

7 Add chart

This option allows you to create a new graph from two or more existing ones and add it to the notebook. To create a new graph, you will need to select at least two of the found graphs in the currently selected notebook (1, 2) and add it to the notebook (3) as shown in Fig. 16.

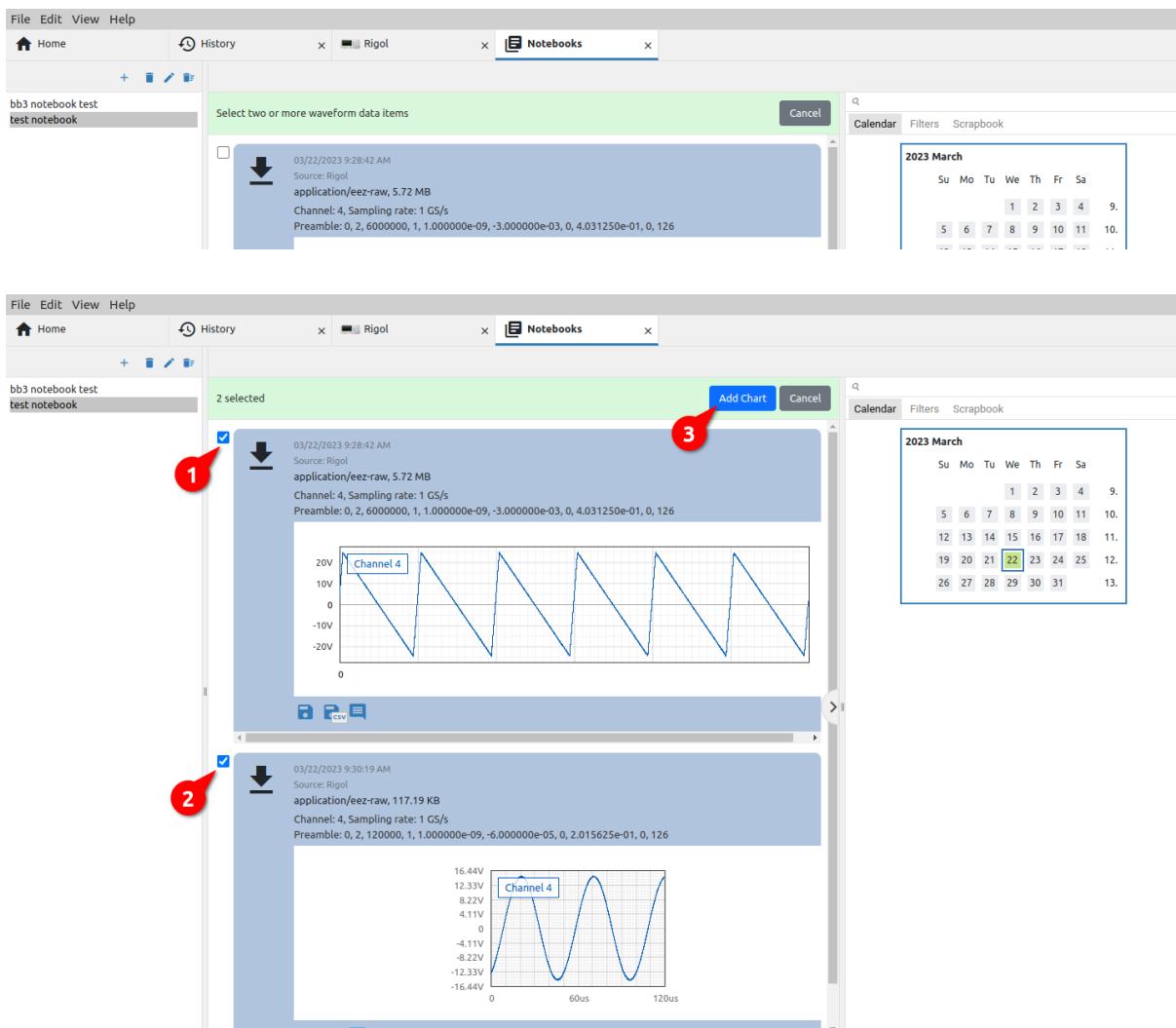


Fig. 16: A selection of graphs to add to the notebook

A successfully created graph will appear at the end of the notebook and will have a graph icon in the upper left corner (Fig. 17).



Fig. 17: Newly created graph added to notebook

6.4. Items purge and restore

Items that are removed from the list are not immediately deleted from the database, which leaves the possibility to restore them if needed. The counter of deleted items that can be restored appears in the right corner as shown in Fig. 18.

The counter can be seen in *Notebooks* but also in the *Terminal* tab of the currently selected instrument, and the same rules apply to restore or purge items in both places.

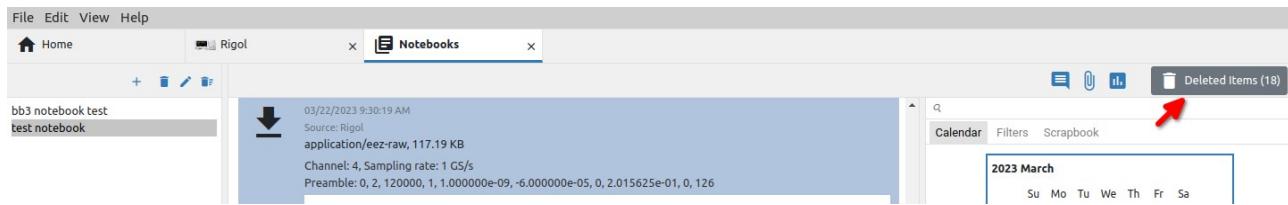


Fig. 18: Deleted items counter

When there are items to delete, they can be accessed by clicking on the counter, when the option to purge all items will first appear (Fig. 19).

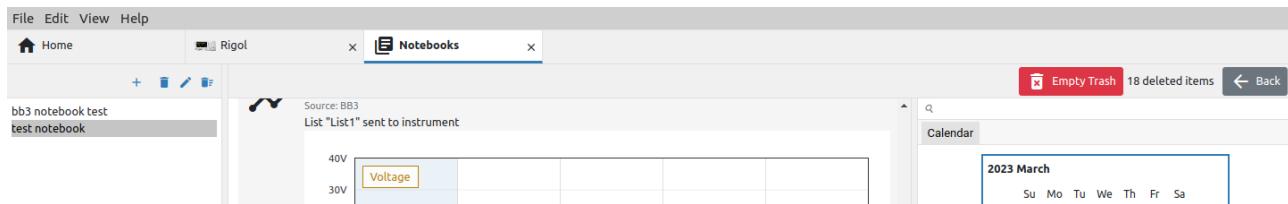


Fig. 19: Empty trash option (no selected items)

If one or more items are selected from the list of deleted items, options for restore (2) or purge (3) will appear (Fig. 20).

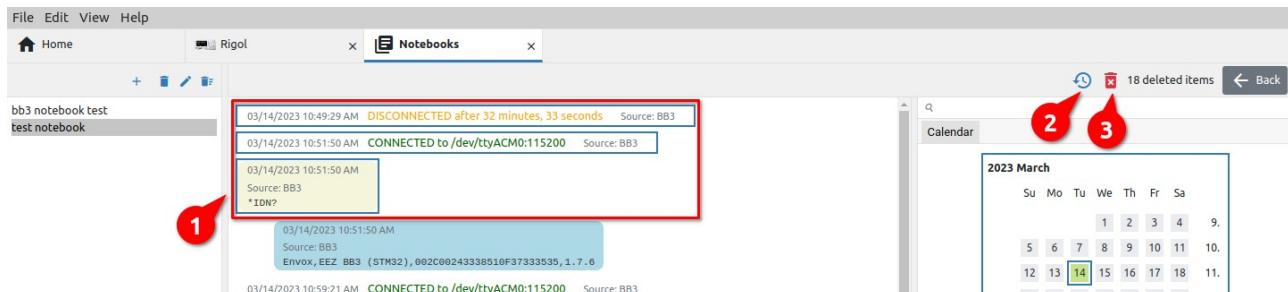


Fig. 20: Selection of deleted items for restore or purge

6.5. Instrument Extension (IEXT) Manager

The EEZ Studio use *Instrument Extensions* (IEXTs) to make communication and control of various instruments easier and more efficient. EEZ Studio comes with IEXTs for several instruments including EEZ H24005, EEZ BB3 as well as Generic SCPI which can be used for basic operations such as connection testing and sending commands and queries. (e.g. *IDN?).

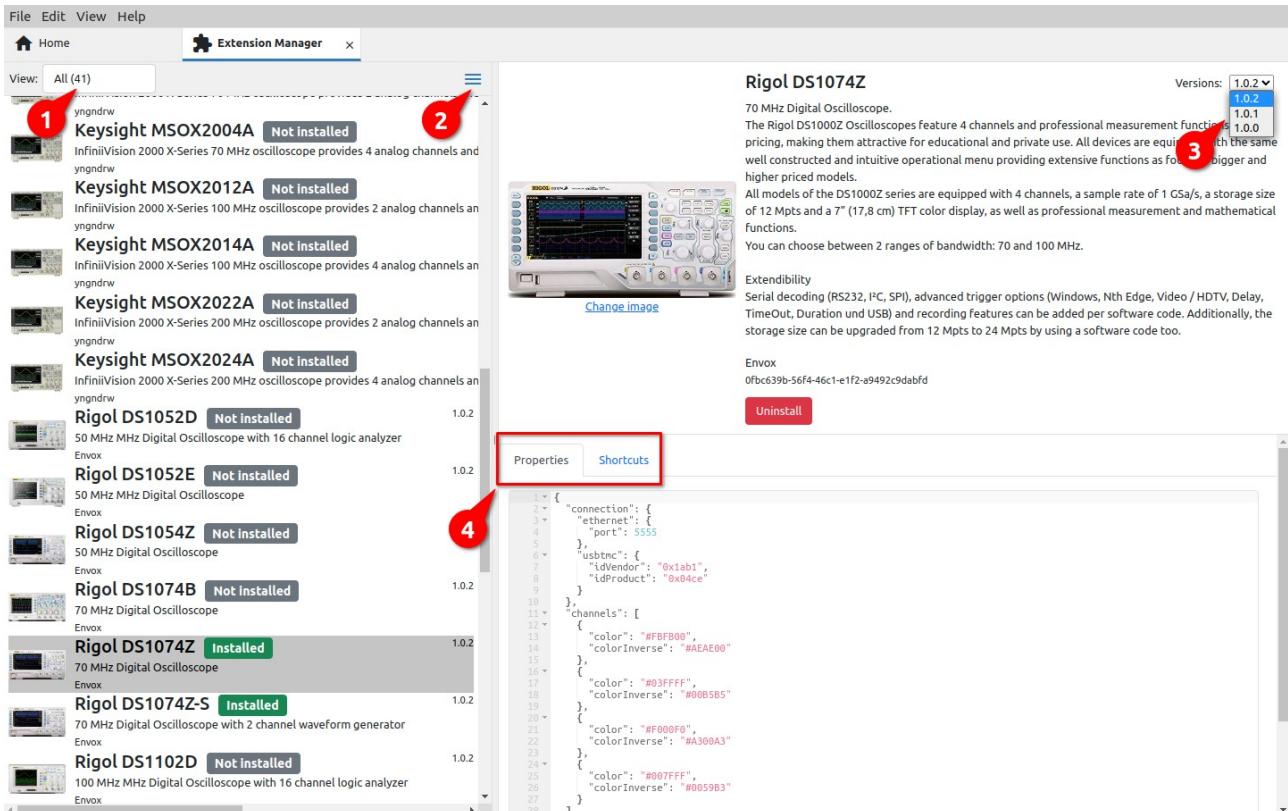


Fig. 21: Instrument extension (IEXT) Manager view

#	Option	Description
1	<i>View</i>	Filters for displaying IEXT in the list: it is possible to display all, only installed or only those that are not installed. The number of filtered IEXTs is displayed next to each option.
2	<i>Update / Install actions</i>	All approved IEXTs are in the catalog on GitHub, with which EEZ Studio synchronizes its catalog every time it is started. Synchronization with the IEXT catalog can also be started manually at any time using the <i>Upgrade Catalog</i> option. The <i>Install extension</i> option allows installing an IEXT that is not in the catalog (from local storage).
3	<i>Versions</i>	IEXT can have multiple versions. If there is more than one, it is possible to change the installed IEXT with one of the versions from the list. In this case, the Replace option will appear as in Fig. 22.



Fig. 22: Changing installed IEXT version

4 Properties

IEXT for a supported instrument can have several properties that will be displayed below the IEXT description.
All displayed properties are for informational purposes and cannot be changed here.

6.6. Add instrument

By using *Add instrument* (Fig. 6), only those instruments for which there is an IEXT in the IEXT catalog can be added to the workbench.

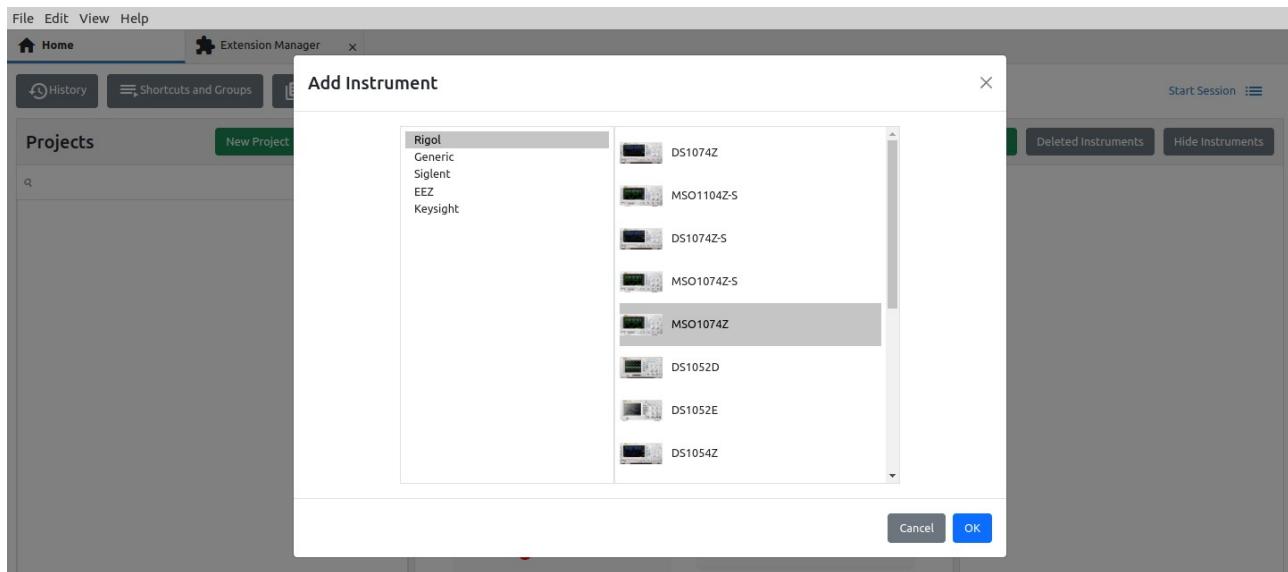


Fig. 23: Add instrument to workbench

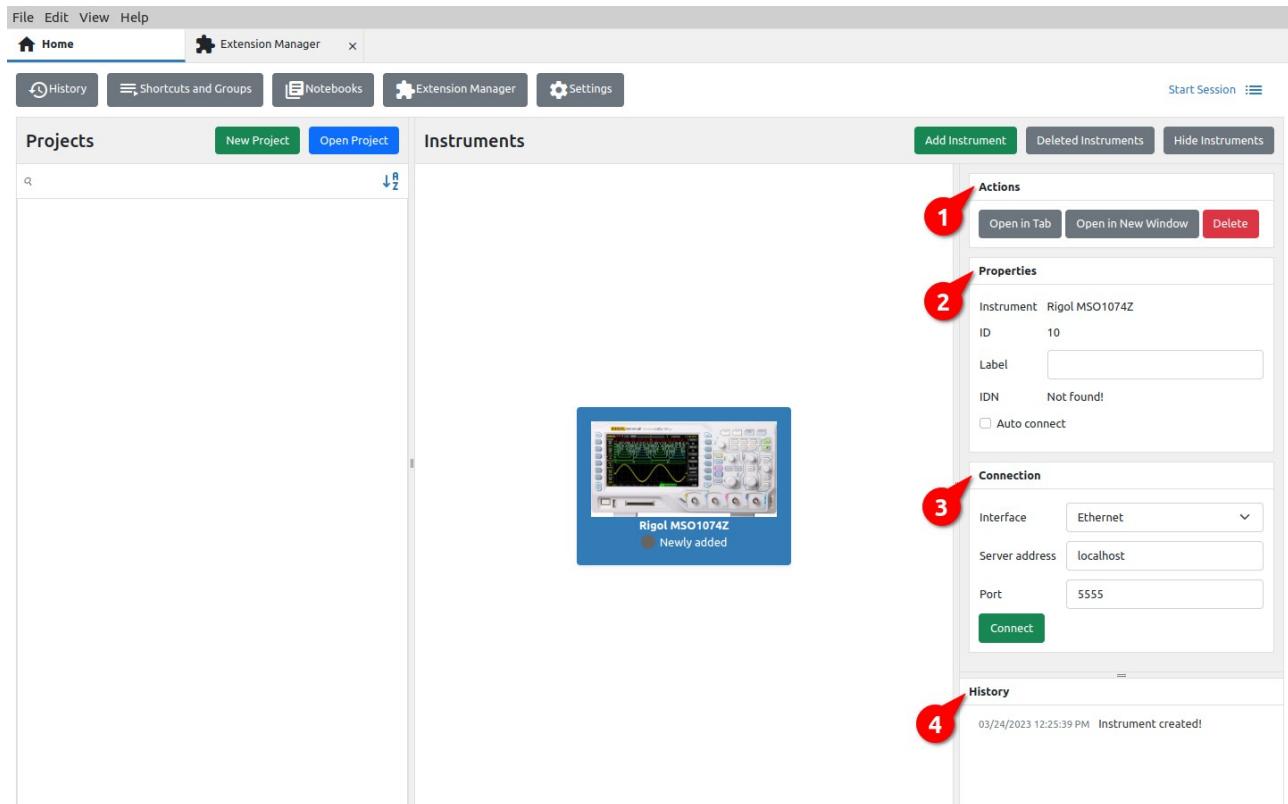


Fig. 24: New added instrument

A successfully added instrument will appear on the workbench (Fig. 24) with the label *Newly added*, and when selected, the sidebar will have the following sections:

#	Option	Description
1	<i>Actions</i>	Basic set of actions for displaying the instrument in a separate tab or new window and for removing it from the workbench.
2	<i>Properties</i>	The properties of the instrument contain information about the IEXT name, the internal ID, the instrument label that can be changed as desired, the identification string that the instrument returns in response to the SCPI query *IDN? and the option to automatically establish a connection with the instrument when starting EEZ studio.
3	<i>Connection</i>	Connection type. Connections to the instrument are defined in IEXT and there can be several of them. Depending on the type of connection (e.g. Serial, Ethernet, USBTMC, VISA), the associated connection parameters will also be displayed.

Please note that the USBTMC and VISA interfaces are experimental and may not work properly on your computer.

For normal communication via the VISA interface, it will be necessary to install a free [R&S®VISA](#) driver. In case it is not installed or there is some problem in communication with it, an error message will appear as in Fig. 25.

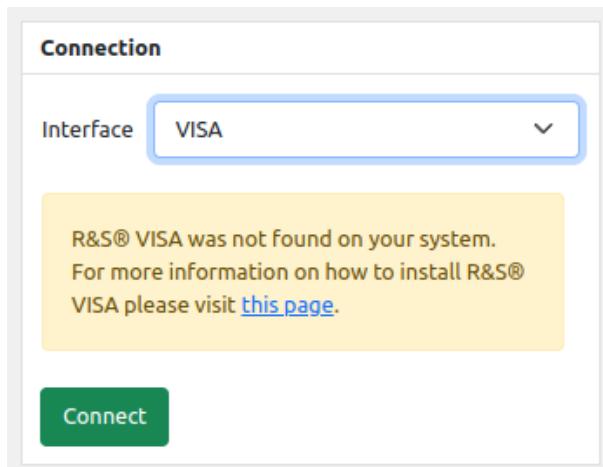


Fig. 25: VISA driver error message

- 4 *History* Preview history of interaction with the instrument using *Terminal*.

6.7. Establishing a connection with the instrument

Connection to the instrument added to the workbench will be possible as shown in Fig. 26: select the instrument from the workbench (1), select the interface in the Connection section (2) and click the Connect button (3).

If the Instrument tab (1) is open, as shown in Fig. 27 to establish a connection, it will be necessary to click on the *Connect* button (2) when a dialogue for choosing an interface will open in which the connection parameters are defined.

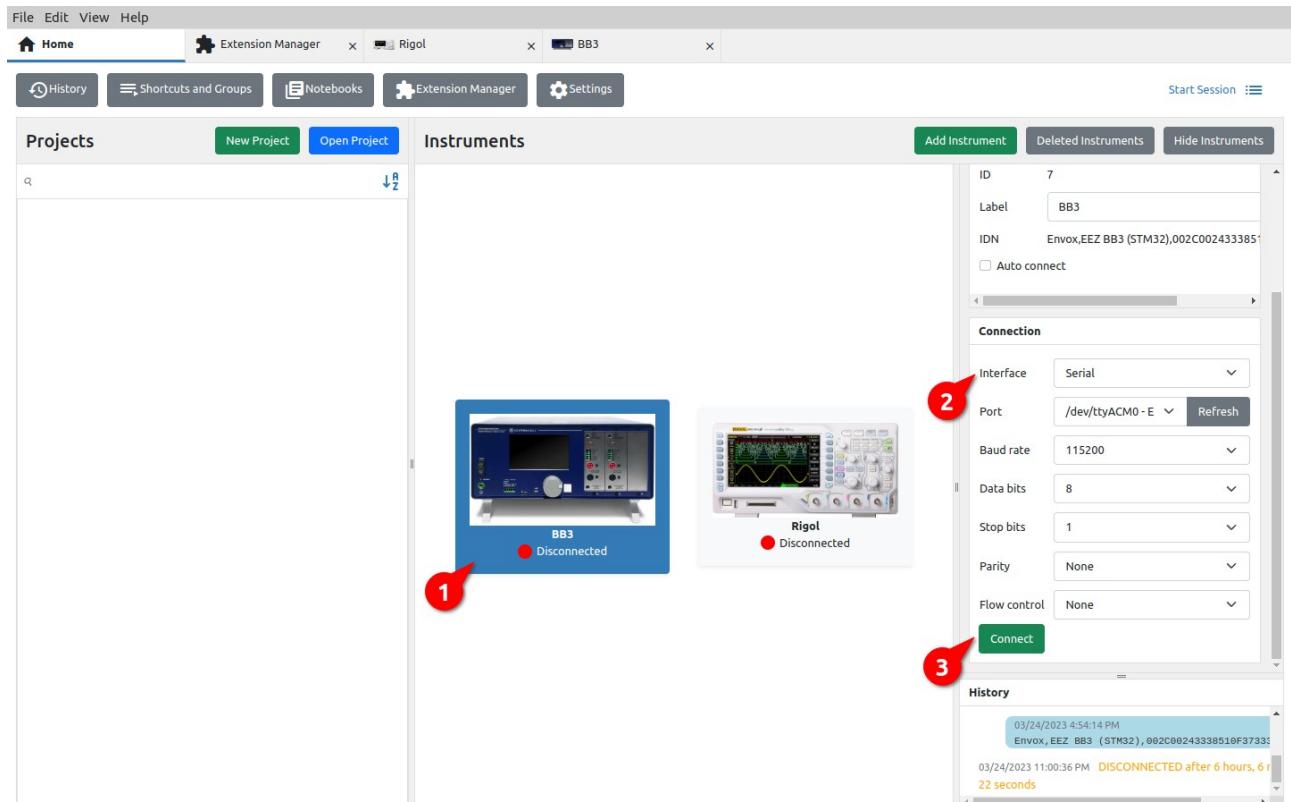


Fig. 26: Selecting an instrument on the workbench to establish a connection

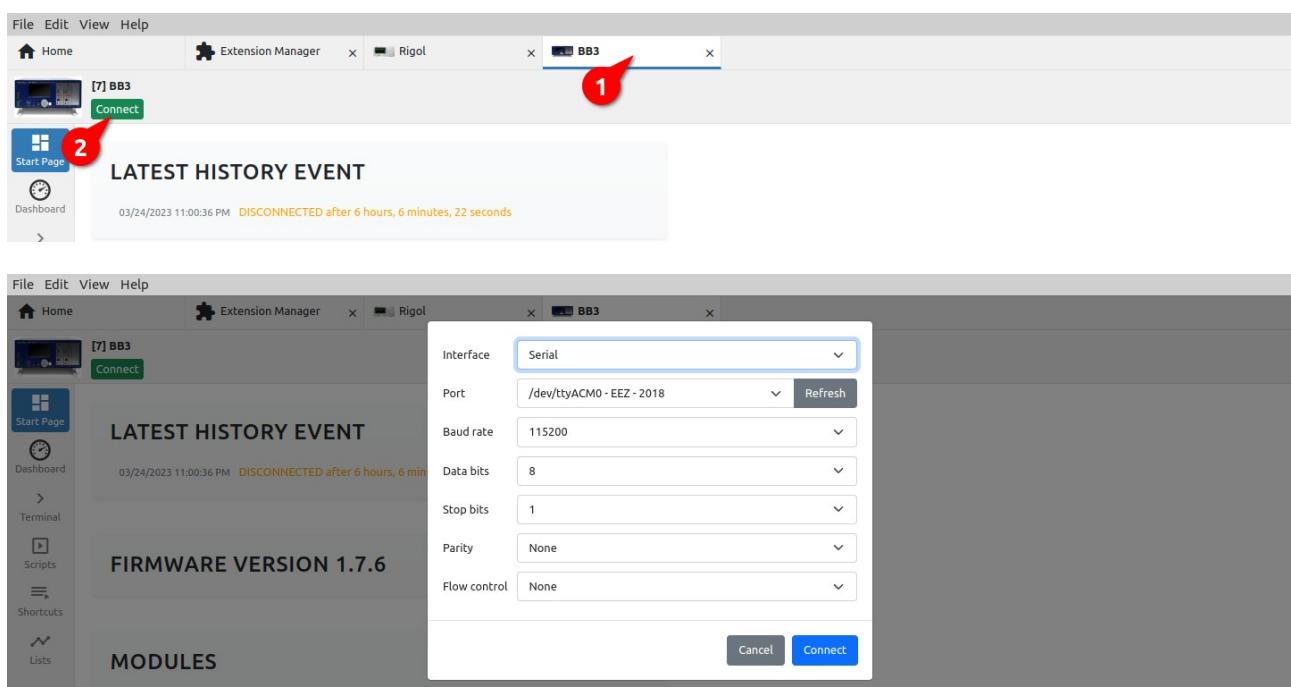


Fig. 27: Establishing a Connection from the Instrument tab

6. Home page instrument sections

Once the connection is established, it will be possible to close the connection by selecting the Disconnect button (Fig. 28).



Fig. 28: Option to close the connection

7. Instrument activity bar

When we open the instrument in its view, an *Activity bar* will be displayed along the left edge. The number of options in the activity bar is defined by IEXT and may vary for different instruments.

7.1. Start page (EEZ BB3 only)

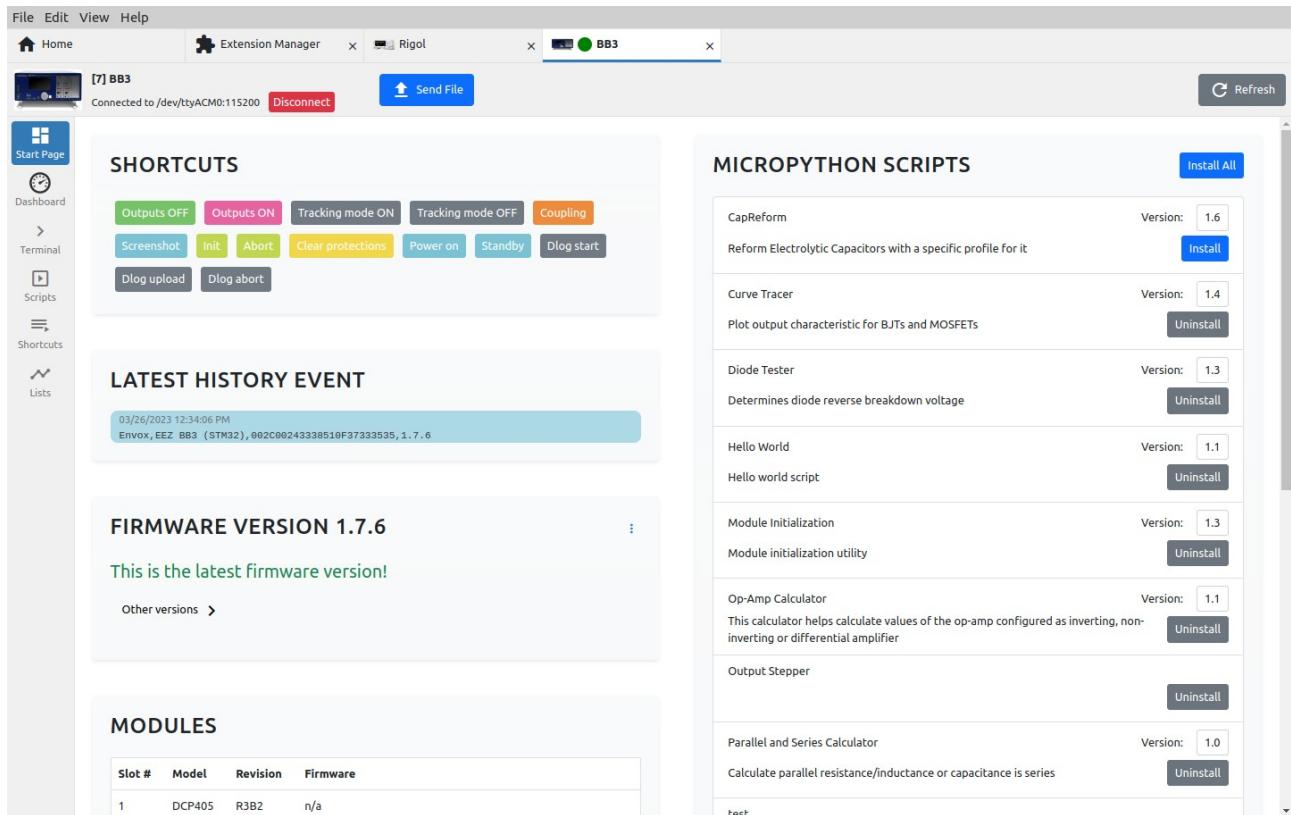


Fig. 29: EEZ BB3 start page

Section / option

Send file

Description

Opens a dialog for sending the file to EEZ BB3. To send, it is necessary to choose the source file, the desired name of the destination file. The destination folder path can be chosen from the offered list or set a new one. The parameters of the send file protocol are predefined and can be viewed and changed via the "gear" button in the lower left corner.

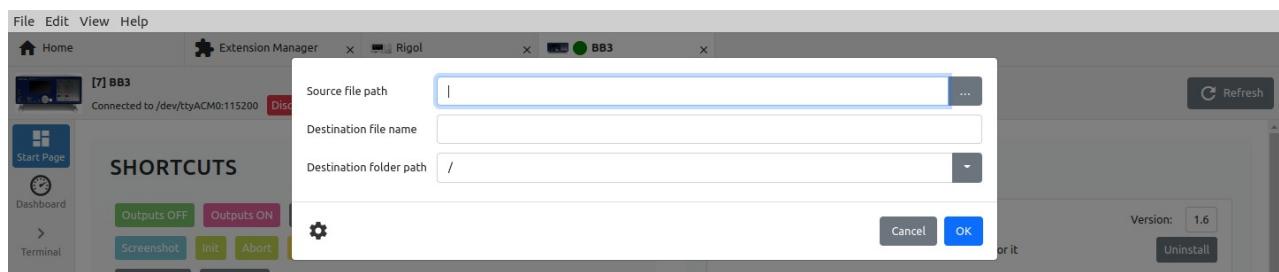


Fig. 30: Sending a file to EEZ BB3

Refresh

Refresh all data displayed on the *Start page*.

Shortcuts

List of available shortcuts from which they can be executed directly.

Latest history event

Shows the last result of interaction with the instrument via the *Terminal* tab.

Firmware version

Displays information about the installed firmware version. If a newer version than the currently installed one is published, an up-

grade option will be offered. It is also possible to manually install another version of the firmware (1) or downgrade the version from the offered list (2) as shown in Fig. 31.



Fig. 31: EEZ BB3 firmware version section

Modules

Display of installed modules. If the module has firmware, information will be displayed as to whether it is up-to-date or not and the possibility to upgrade or install another version.

Upload Pinout Pages is used to update pinout images of all modules.

MODULES			
Slot #	Model	Revision	Firmware
1	DCP405	R3B2	n/a
2	DCP405	R3B2	n/a
3	MIO168	R2B4	0.12 This is the latest firmware version!

[Other versions >](#)

[Upload Pinout Pages](#)

Fig. 32: EEZ BB3 modules section

Micropython scripts

List of all Micropython scripts that are on EEZ BB3. For scripts that are synchronized with the GitHub repository, their versions and options to install or uninstall will be displayed.

For scripts created by the user, versions will not be displayed, only the option to install or uninstall.

Lists

Program lists created by the user (see Section 7.4.). and which are located on EEZ BB3. Lists can be downloaded, uploaded and edited.

LISTS		Download All	Upload All
3.3V stress test		03/14/2023 10:51:38 AM	Download Upload Edit
List1		03/15/2023 5:08:38 PM	Upload Edit
New list		03/15/2023 2:33:07 PM	Upload Edit
Another sample			

Fig. 33: EEZ BB3 program lists

7.2. Dashboard

Fig. 34 shows an example Dashboard that enables simple operations with EEZ BB3 modules.

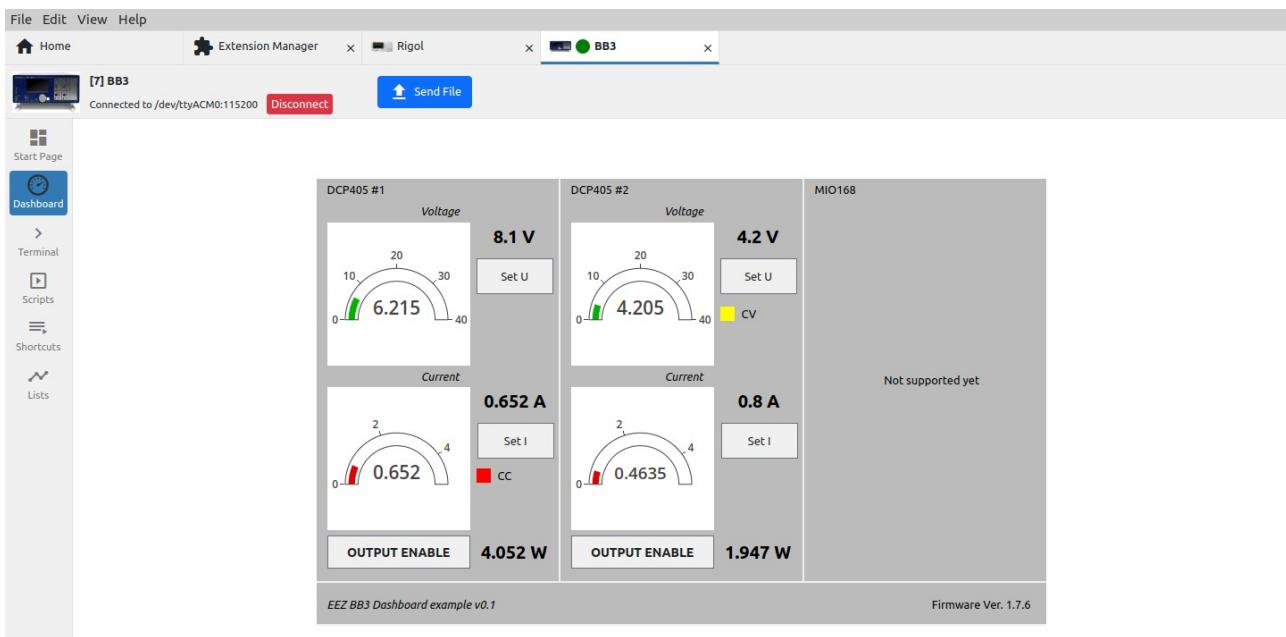


Fig. 34: Instrument dashboard example

7.3. Terminal

The *terminal* allows interaction with the instrument, which is primarily based on the SCPI specification.

The number of SCPI commands varies greatly between instruments, and IEXT can also include help for easier finding of the desired SCPI command or query that will be displayed at the bottom of the screen (7).

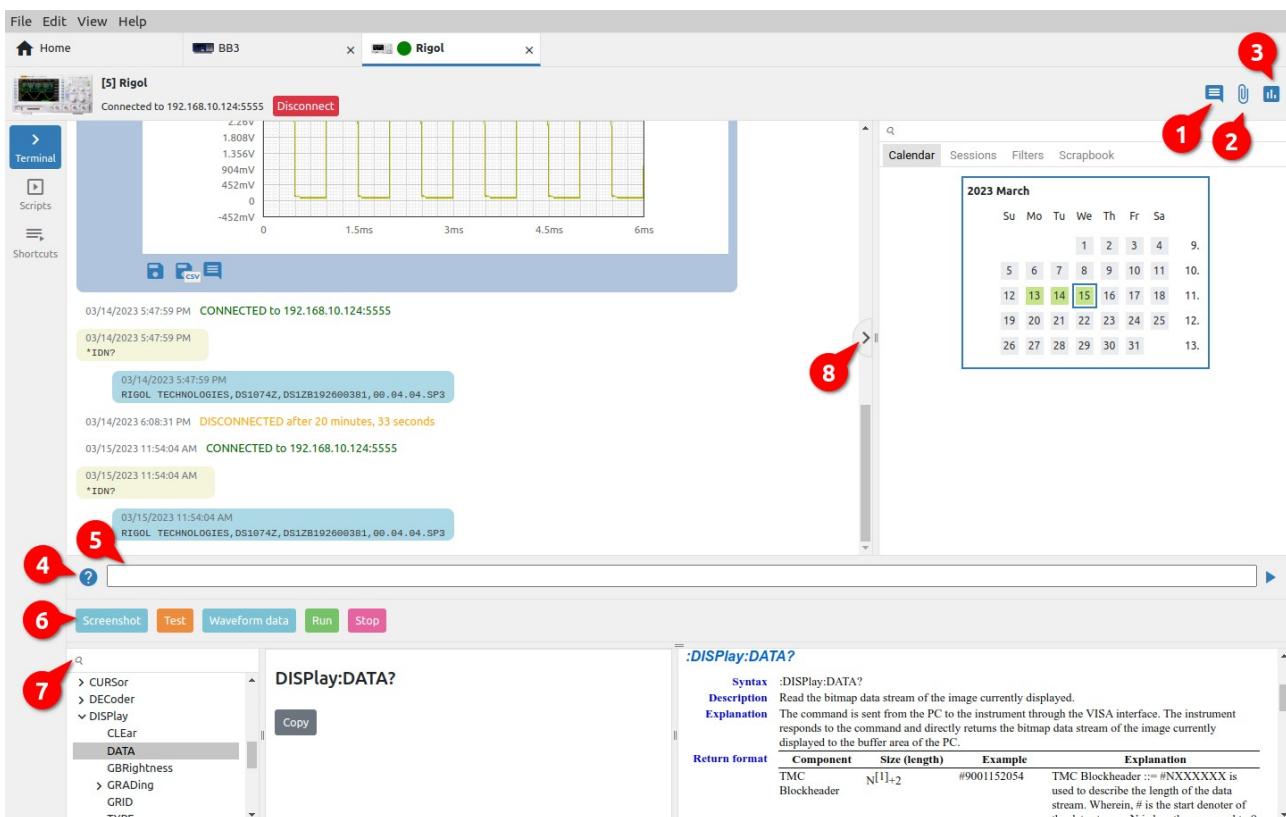


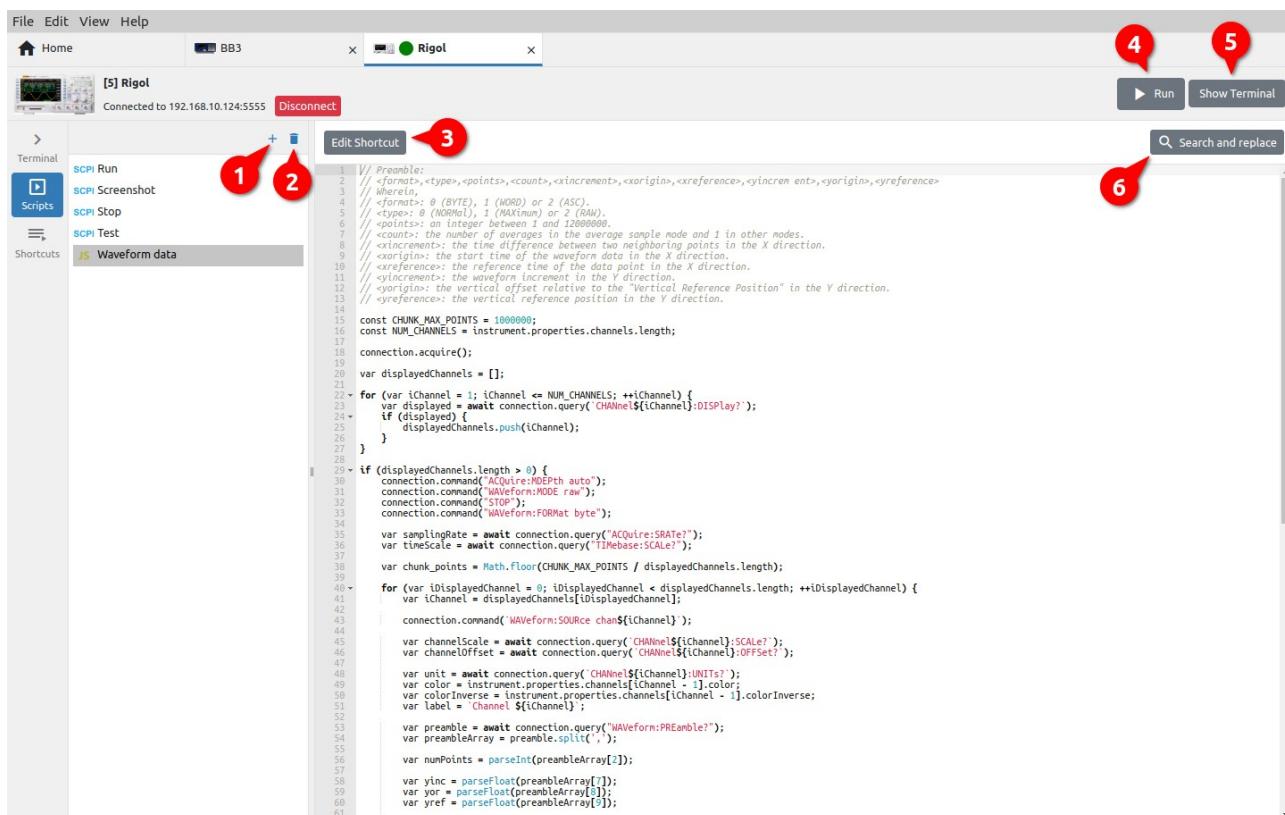
Fig. 35: Instrument terminal

#	Option	Description
1	Add note	Adds a note (see Section 6.3).
2	Attach file	Attaching a file (see Section 6.3).
3	Add chart	Creating a new chart from two or more charts (see Section 6.3).
4	Show/hide commands catalog	Show or hide the help section for instrument commands at the bottom of the <i>Terminal</i> view. Command help will only be displayed if it is defined in IEXT for the selected instrument. Help for each command contains an explanation and syntax of how the command is used with the option to copy it to the command line (5).
5	Command line	Prompt line for sending a command to the instrument.
6	Shortcuts bar	IEXT imported and user defined shortcuts.
7	Command search	Commands help search.
8	Show/hide Side bar	Show or hide sidebar with history search options.

7.4. Scripts

Scripts can be used to automate communication with the instrument (configuration, data collection, test sequences, etc.). Three types of scripts are supported: SCPI commands, JavaScript (JS) code and MicroPython (EEZ BB3 only) script. The number of scripts is unlimited and can be defined in IEXT or created by the user. A shortcut can be added to the script for easier launch.

In addition to containing complex programming procedures, a JS script can also contain GUI elements for communication with the user (entry forms, info or error messages, etc.).



The screenshot shows the EEZ Studio interface with the following numbered callouts:

- 1: A red circle points to the "Scripts" icon in the left sidebar.
- 2: A red circle points to the "+" button in the Scripts panel, which is used to add a new script.
- 3: A red circle points to the "Edit Shortcut" button in the Scripts panel.
- 4: A red circle points to the "Run" button in the top right corner of the interface.
- 5: A red circle points to the "Show Terminal" button in the top right corner.
- 6: A red circle points to the "Search and replace" bar in the top right corner.

The main area displays a JS script titled "Waveform data". The script is as follows:

```

1 // Preamble:
2 // <format>, <type>, <points>, <count>, <xincrement>, <xorigin>, <xreference>, <yincrement>, <yorigin>, <yreference>
3 // <format>: 0 (BYTE), 1 (WORD) or 2 (ASC).
4 // <type>: 0 (NORMAL), 1 (MAXIMUM) or 2 (RAW).
5 // <points>: an integer between 1 and 12800000.
6 // <count>: the number of data points to average sample mode and 1 in other modes.
7 // <xincrement>: the time difference between two neighboring points in the X direction.
8 // <xincrement>: the start time of the waveform data in the X direction.
9 // <xorigin>: the reference time of the data point in the X direction.
10 // <xreference>: the waveform increment in the X direction.
11 // <yincrement>: the vertical offset relative to the "Vertical Reference Position" in the Y direction.
12 // <yreference>: the vertical reference position in the Y direction.
13
14 const CHUNK_MAX_POINTS = 1000000;
15 const NUM_CHANNELS = instrument.properties.channels.length;
16
17 connection.acquire();
18
19 var displayedChannels = [];
20
21 for (var iChannel = 1; iChannel <= NUM_CHANNELS; ++iChannel) {
22     var displayed = await connection.query(`CHANnel${iChannel}:DISPlay?`);
23     if (displayed) {
24         displayedChannels.push(iChannel);
25     }
26 }
27
28 if (displayedChannels.length > 0) {
29     connection.command("ACQuire:DEPth auto");
30     connection.command("WAveform:MODE raw");
31     connection.command("STOP");
32     connection.command("WAveform:FORMAT byte");
33
34     var samplingRate = await connection.query("ACQuire:SRATE?");
35     var timeScale = await connection.query("TImebase:SCALE?");
36
37     var chunk_points = Math.floor(CHUNK_MAX_POINTS / displayedChannels.length);
38
39     for (var iDisplayedChannel = 0; iDisplayedChannel < displayedChannels.length; ++iDisplayedChannel) {
40         var iChannel = displayedChannels[iDisplayedChannel];
41
42         connection.command(`WAveform:SOURce chAnel${iChannel}`);
43
44         var channelscale = await connection.query(`CHANnel${iChannel}:SCALE?`);
45         var channelOffset = await connection.query(`CHANnel${iChannel}:OFFSet?`);
46
47         var unit = await connection.query(`CHANnel${iChannel}:UNITS?`);
48         var color = instrument.properties.channels[iChannel - 1].color;
49         var colorInverse = instrument.properties.channels[iChannel - 1].colorInverse;
50         var label = `CHANnel ${iChannel}`;
51
52         var preamble = await connection.query("WAveform:PREamble");
53         var preambleArray = preamble.split(",");
54
55         var numPoints = parseInt(preambleArray[0]);
56
57         var yinc = parseFloat(preambleArray[1]);
58         var yor = parseFloat(preambleArray[2]);
59         var yref = parseFloat(preambleArray[3]);
60
61         if (unit === "V") {
62             yinc *= 1000000;
63             yor *= 1000000;
64             yref *= 1000000;
65         }
66     }
67 }

```

Fig. 36: Instrument scripts

#	Option	Description
1	Add script	Creating a new script. It will be necessary to define the name and type: SCPI, JS or MicroPython (EEZ BB3 only).

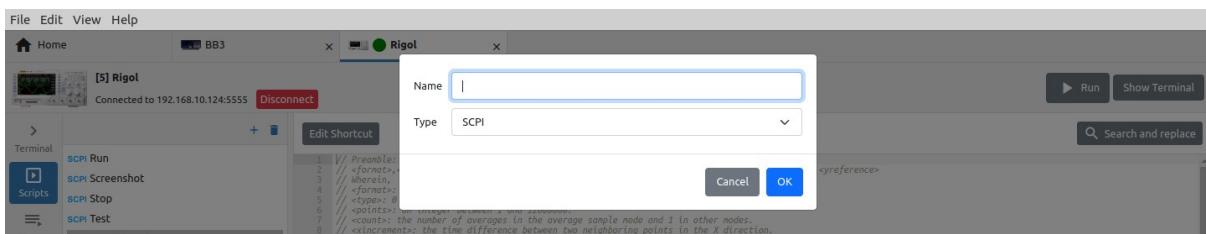


Fig. 37: Adding a new script

The content of the script is entered in the editor (Fig. 38).

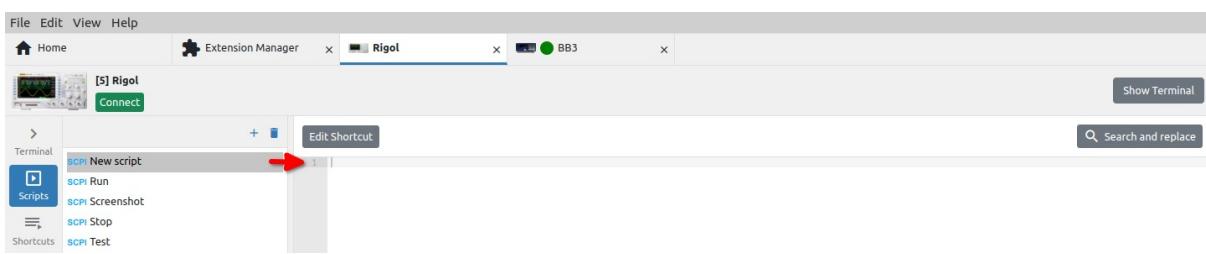


Fig. 38: Script editing

2	Delete script	Deleting the selected script.
3	Edit shortcut	Editing a script shortcut (see Section 7.4.1.)
4	Run	Runs the script on the instrument. This option is only displayed if the connection to the instrument is established.
5	Show / Hide terminal	Show / hide Terminal on the right.
6	Search and replace	Script editor function for searching and replacing text in the script. By default, only the search field is displayed. To replace the found text, it will be necessary to click on the "+" sign.

7.4.1. Edit script shortcut

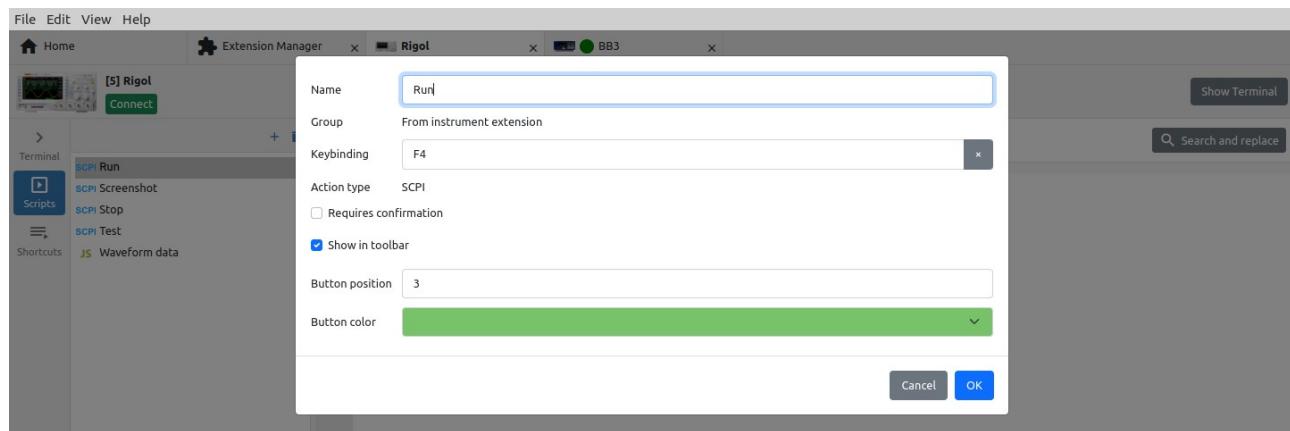


Fig. 39: Script shortcut editing

Option	Description
Name	The name of the script shortcut as it will be displayed in the shortcut bar.
Group	The name of the group to which the shortcut belongs. If the shortcut is defined in IEXT, the label <i>From instrument extension</i> will be displayed.

<i>Keybinding</i>	A key or a combination of several keys (e.g. with SHIFT, ALT, CTRL) that will start the execution of the script.
<i>Action type</i>	Script type: SCPI, JS or MicroPython (EEZ BB3 only).
<i>Requires confirmation</i>	Displays a dialog box to confirm the execution of the script.
<i>Show in Shortcuts bar</i>	Determines whether the shortcut button will be displayed in the <i>Terminal's Shortcuts bar</i> .
<i>Button position</i>	The position of the shortcut button in the <i>Shortcuts bar</i> . When displaying, the shortcut with a lower value will be displayed first. If there are multiple shortcuts with the same value, they will be sorted alphabetically.
<i>Button color</i>	Color coding of shortcut button.

7.5. Shortcuts

Shortcuts are used to simplify the execution of scripts and can be defined in IEXT or user defined.

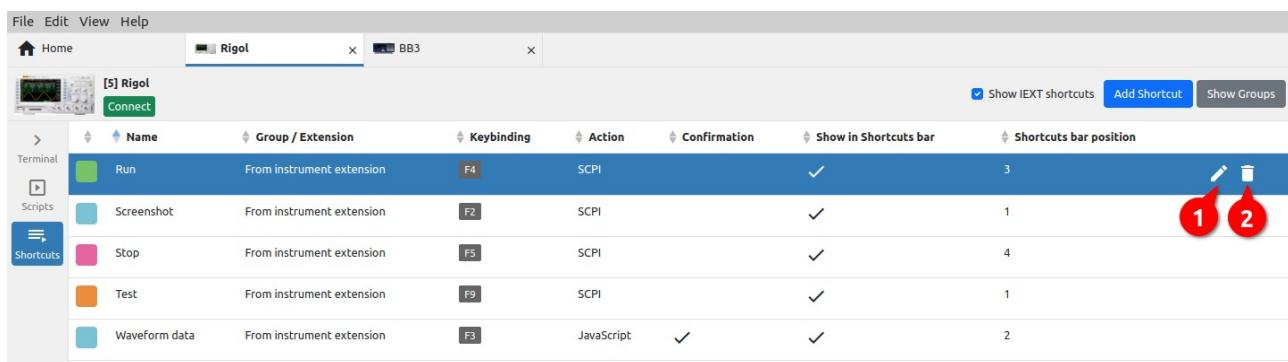


Fig. 40: Instrument shortcuts

#	Option	Description
1	<i>Edit shortcut</i>	Editing the shortcut (see Section 7.4.1.)
2	<i>Delete shortcut</i>	Deleting an existing shortcut.
	<i>Show IEXT shortcuts</i>	Filters the display of Shortcuts belonging to the installed Instrument Extension (IEXT).

Add Shortcut Adding a new shortcut opens the entry form as shown in Fig. 41.

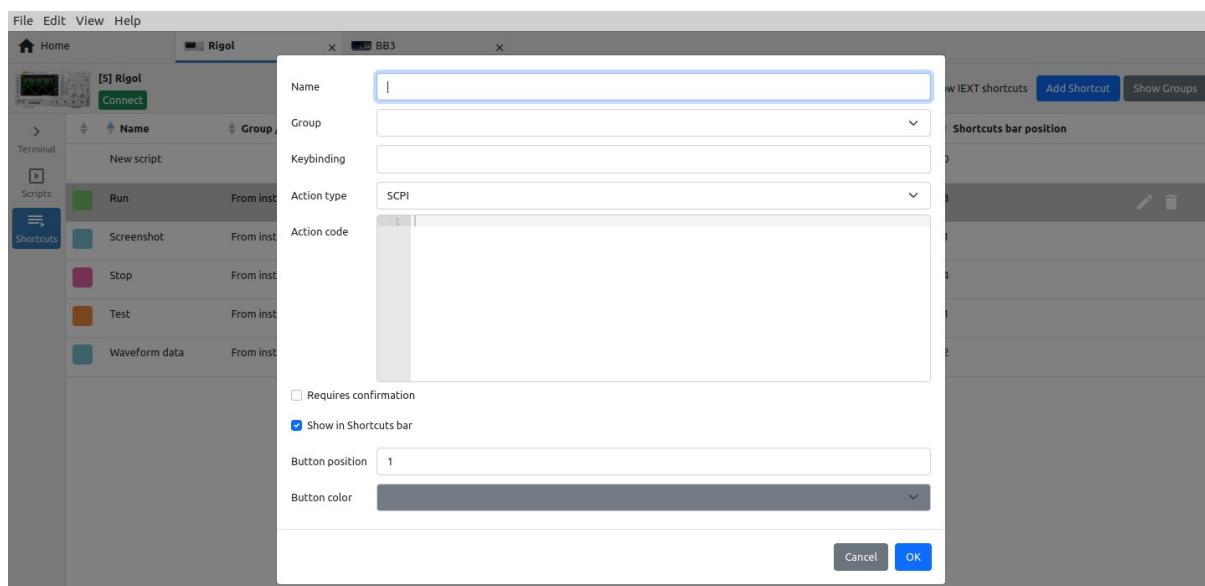


Fig. 41: Add new shortcut

Show Groups / Show Shortcuts

Toggle between displaying a list of shortcuts and groups (Fig. 42) of shortcuts.



Fig. 42: Instrument shortcut groups

7.6. Lists

Lists are used to program parameters for instruments that support SCPI list commands. Lists for programming value and duration of output voltage and current for EEZ BB3 will be described below.

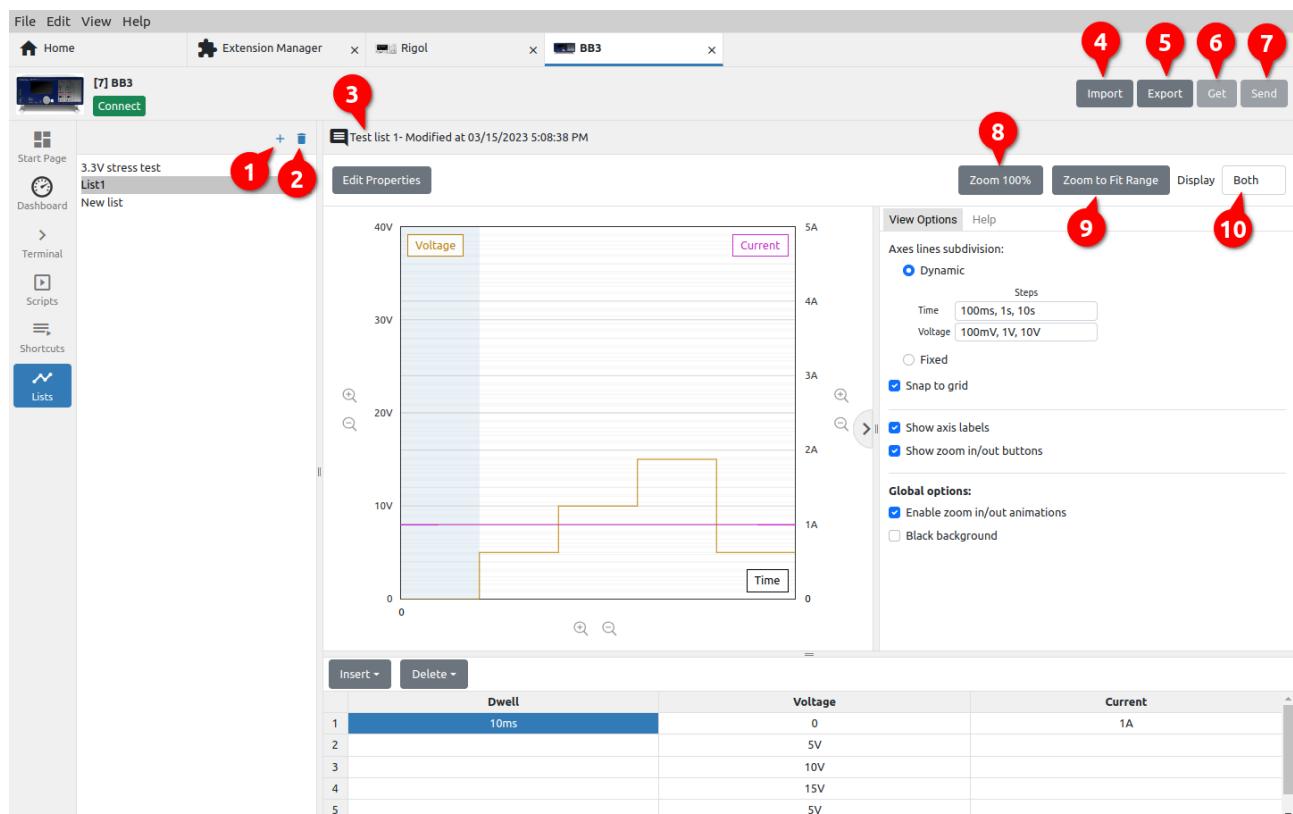


Fig. 43: Instrument programming lists

Option**Description**1 *Add list*

Creating a new list. The parameters of the list can be specified through a table (Fig. 43) or by defining envelope points that show the change of the parameter value over time.
In addition to the list *Type*, it will be necessary to define a *Name* and optionally a *Description*.

2 *Remove list*

Deleting the list (use *Undo* from the *Edit* menu to restore).

3 *List info*

List description and datetime of last changes.

4 *Import*

Import list from local storage. Opens a new dialog box for selecting the folder and name.

5 *Export*

Export list to local storage. Opens a new dialog box in which a list file can be selected.

6 Get

Receiving a list from the instrument. The option will be disabled if connection is not established with the instrument.
Opens a menu (Fig. 44) where you can choose the source (e.g. channel) from which the list will be received. For the imported list, it is necessary to enter the name and description (Fig. 45).

If the selected source does not have a defined list, an empty list will be imported.

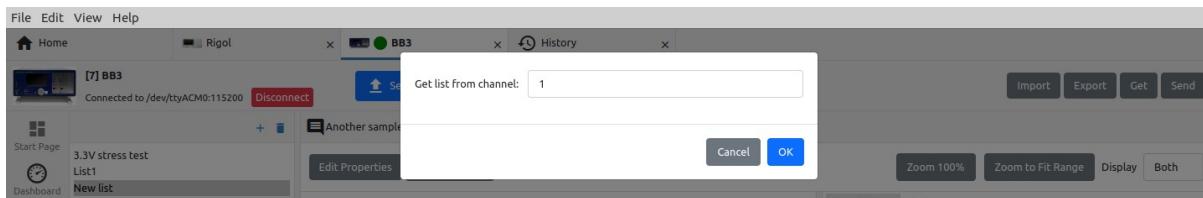


Fig. 44: List source selection

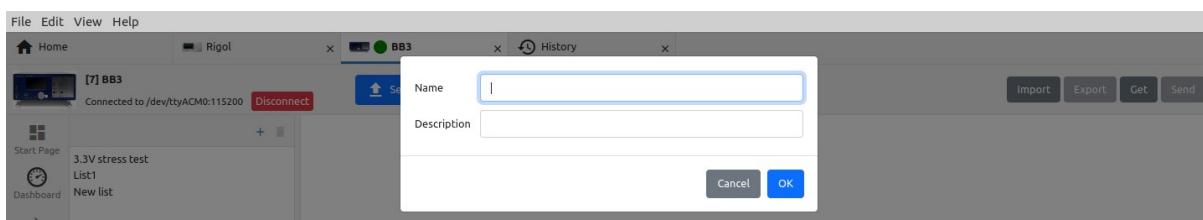


Fig. 45: Imported list parameters

7 Send

Sending the list to the instrument. The option will be disabled if connection is not established with the instrument.

8 Zoom 100%

Display graph without scaling.

9 Zoom to Fit Range

Graph display scaled according to the largest defined value.

10 Display

Selection of graphs to be displayed (e.g. voltage only, current only, both).

7.6.1. Editing a list using a table

Editing the list via the table is shown in Fig. 43. The program parameters graph is drawn simultaneously with editing the table at the bottom of the graph. In the case shown, the list contains two program parameters: *Voltage* and *Current*, for which values should be entered as well as duration (*Dwell*). To define the value, it is possible to use the units prefix, e.g. ms for dwell, mV for voltage, and mA for current.

In Fig. 46 and Fig. 47 shows all options for inserting new lines and deleting existing ones.

Insert	Delete	Dwell	Voltage	Current
Insert row above		10ms	0	1A
Insert row below			5V	2A
			10V	1A
			15V	1A
			5V	2A

Fig. 46: Table insertion options

Insert	Delete	Voltage	Current
	Delete row	0	1A
	Clear column from cursor down	5V	2A
	Delete all from cursor down	10V	1A
	Delete all	15V	1A
		5V	2A

Fig. 47: Table deletion options

7.6.2. Editing a list using an envelope

In contrast to the previously mentioned editing of the list, where it is necessary to define program points through a table, envelope mode allows program points to be defined directly on the curve of the parameter being edited. This can simplify and speed up the whole process.

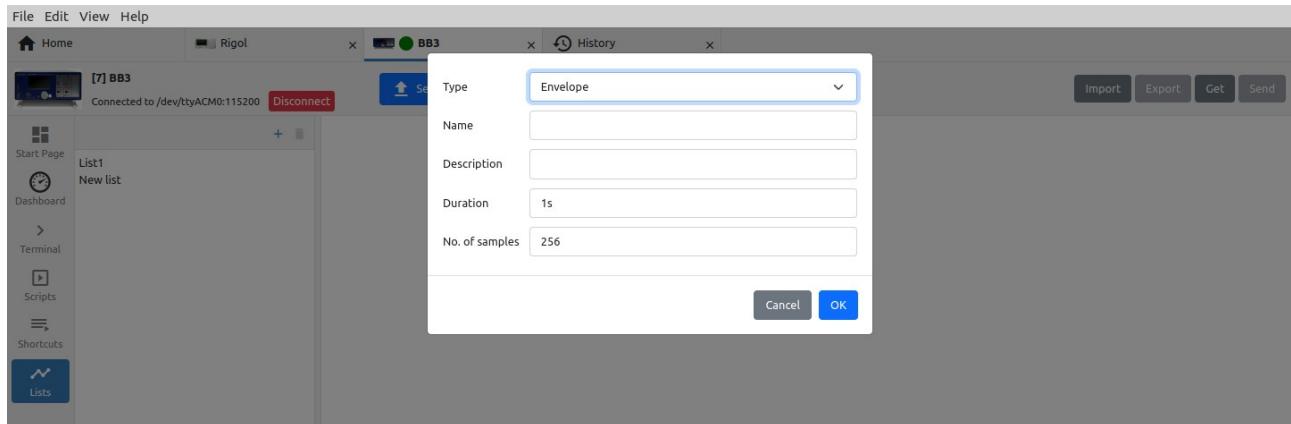


Fig. 48: Adding a new list in envelope mode

When creating a new list in envelope mode, it will be necessary to set two more parameters: the total duration of the program sequence and the number of samples (Fig. 48). The former is needed to be able to display the duration in the graph, and the later is needed to know how many points should be generated in total when sending the list to the instrument.

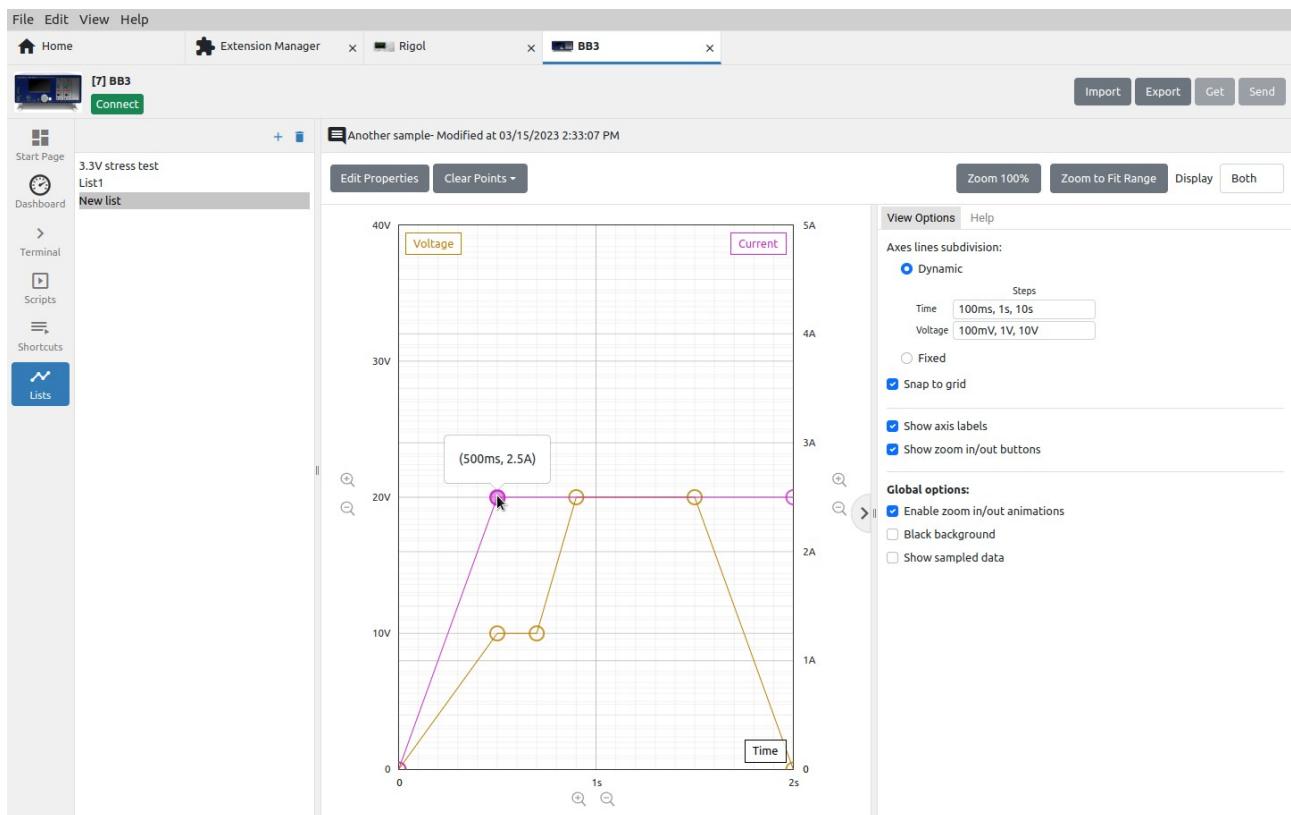


Fig. 49: Graph editing in envelope mode

The example in Fig. 49 contains 6 programming points for setting the voltage (light brown) and 3 for setting the current (magenta).

Adding a new point is simple: you only need to position the cursor somewhere in the graph and click, and a new point will appear, which will be automatically connected to two adjacent ones.

If we want to move the point in any direction, it will be necessary to position the cursor on it again and drag&drop it to a new position somewhere in the graph.

EEZ Studio User manual

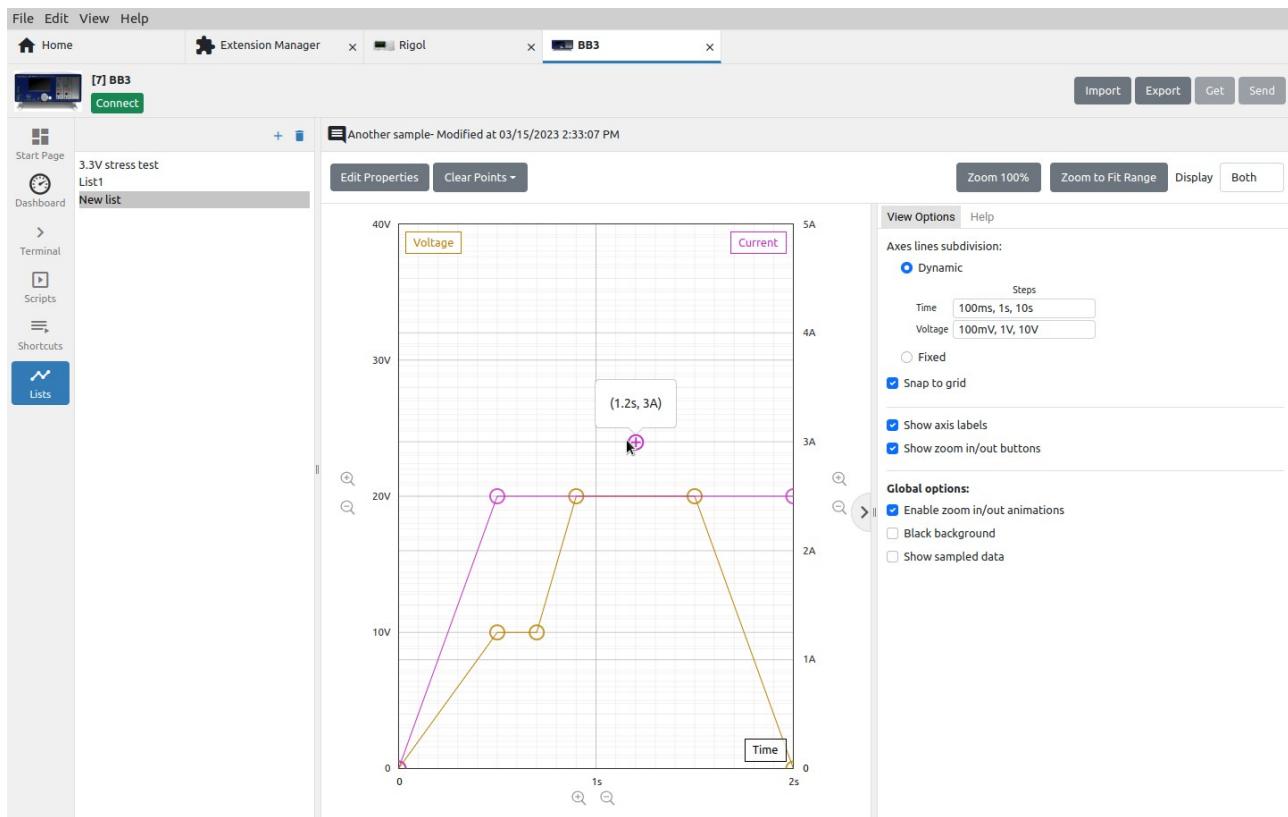


Fig. 50: Adding a new point in envelope mode

If you want to delete an existing point or manually edit its parameters after you have positioned yourself on it, you only need to click once more with the mouse when a dialog box will appear as shown in Fig. 51.

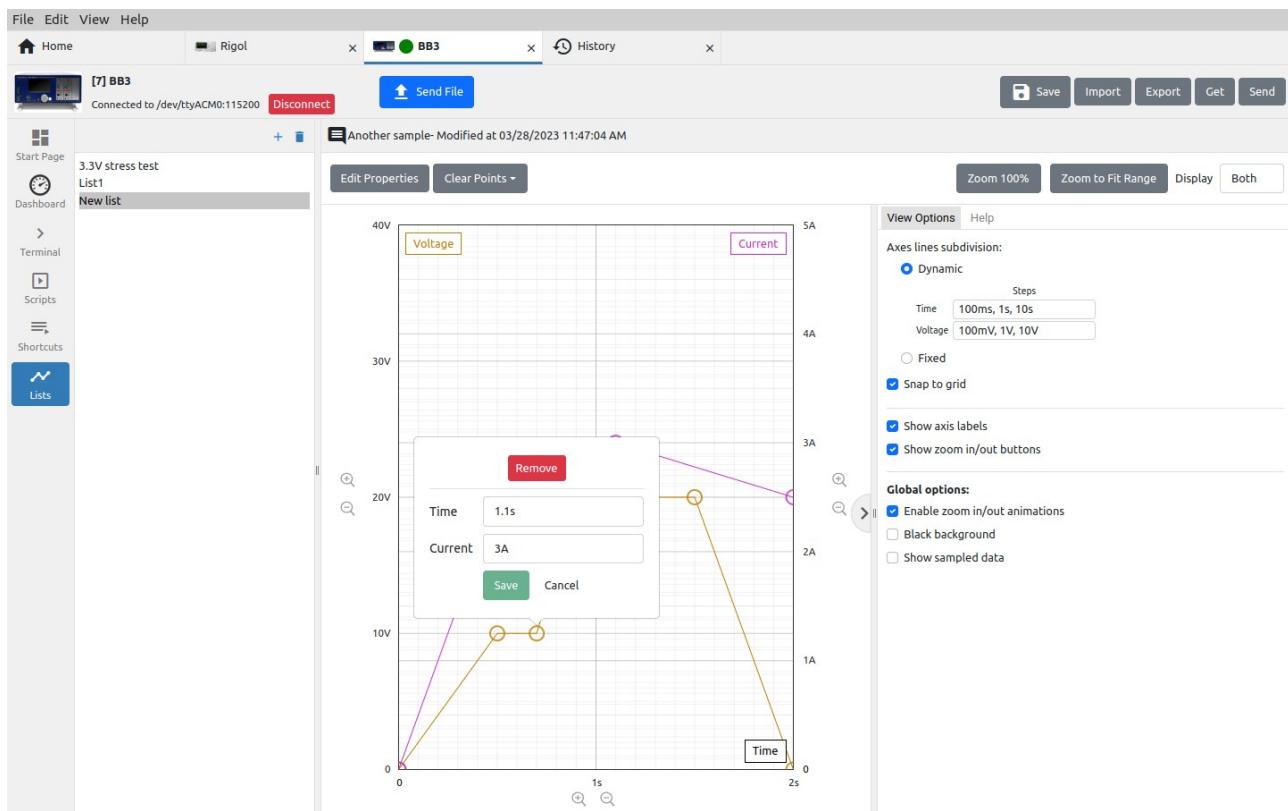


Fig. 51: Program point editing in envelope mode

7.6.3. List view options

The display of the graph can be dynamically changed (Fig. 43) depending on the resize of the window or the number of graticules can be fixed (Fig. 52).

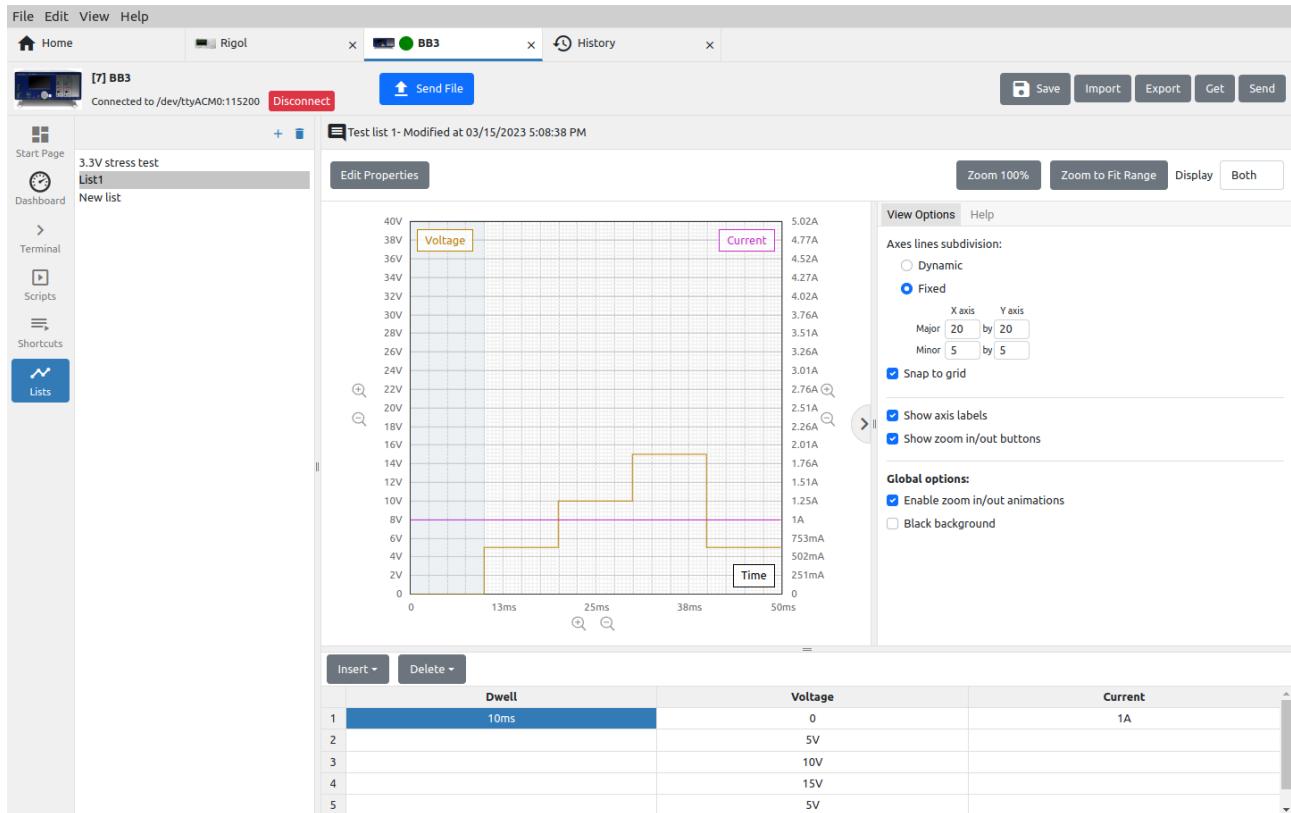


Fig. 52: Fixed graph view

7.6.4. List help

For zooming and navigating the graph, in addition to the zoom options located next to the x- and y-axes of the graph ("+" and "-" magnifier signs), a combination of mouse keys and control keys can be used. These additional options are shown in the Help tab as in Fig. 53.

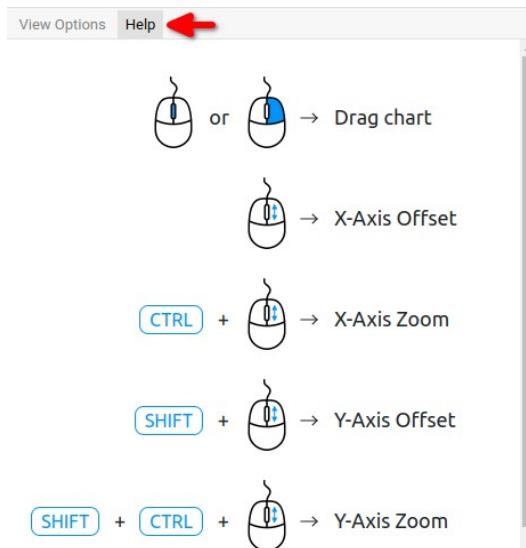


Fig. 53: Graph navigation and zoom help

*EEZ Studio
Project*

8. Home page project sections

One of the important features of EEZ Studio is that it enables the creation of projects for different target platforms using different technologies, which will be described below. The *Projects* section of the home page is shown in Fig. 54. which displays a searchable Recent Project List (RPL).

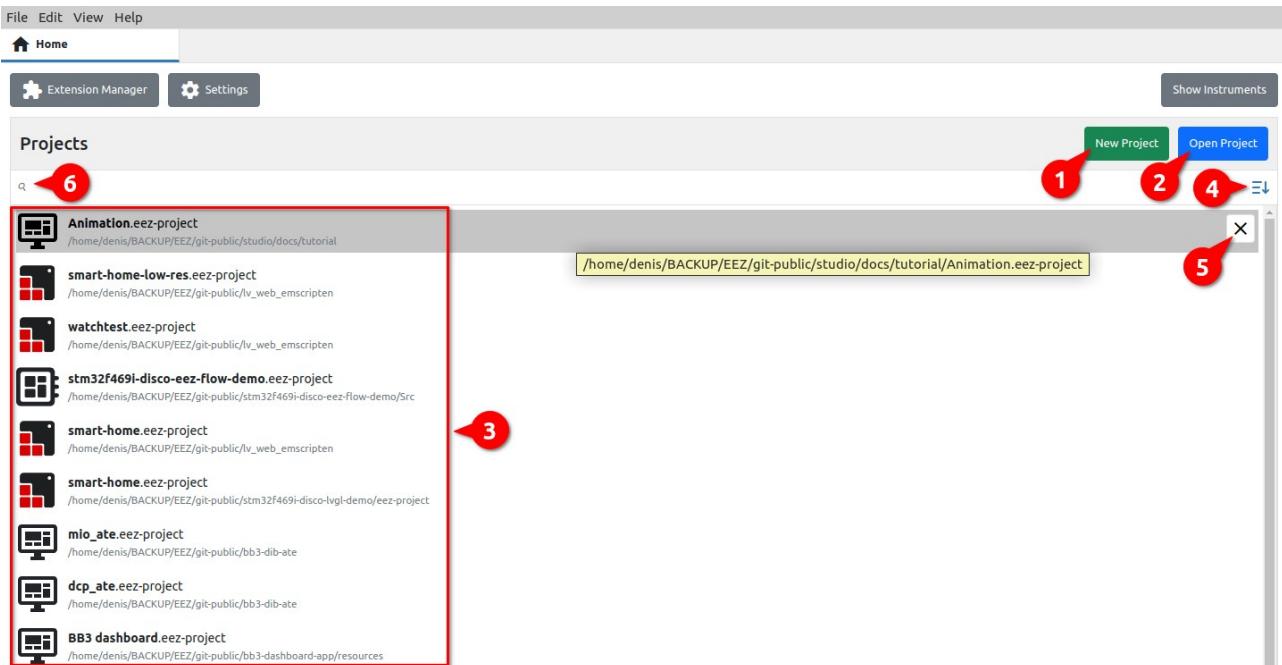


Fig. 54: Home page project options

#	Option	Description
1	New project	Creating a new project.
2	Open project	Opening an existing project (will be added to RPL after successful loading).
3	Recent Project List (RPL)	List of all successfully loaded projects after the first run.
4	RPL sort order	Sorting order of projects in RPL: It can be <i>Show most recent first</i> or <i>Sort alphabetically</i> .
5	Remove from RPL	Removing the project from RPL.
6	Search RPL	RPL search by project name.

8.1. EEZ Studio project types

EEZ Studio offers the creation of the following project types:

- **Dashboard** – desktop application. GUI applications can be quickly and easily created thanks to the drag & drop of available widgets and the import of multiple fonts and ready-made bitmaps prepared by the designer. The animation editor allows adding simple animations to the desired sections of the page or navigation between pages. Finally, the flowchart method of defining program logic instead of programming in one of the programming languages will further speed up prototyping and creation of the final application. The implemented debugger will shorten the application development process and help in more efficient error detection.
- **EEZ-GUI** – embedded GUI application that uses the EEZ-GUI framework. This is a native EEZ Studio framework that was initially developed to speed up and simplify embedded GUI development for [EEZ H24005](#) and [EEZ BB3](#) firmware.
- **LVGL** – embedded GUI application that uses LVGL (Light and Versatile Graphics Library) framework. LVGL is a popular open source project that supports a large number of target platforms. For more information visit <https://lvgl.io/>
- **Applet** – GUI application that can be run on EEZ BB3. Program logic is created using EEZ Flow (flowchart-based programming).

- **MicroPython script** – GUI application that can be run on EEZ BB3. Program logic is created using MicroPython scripting.
- **Templates from gitea repository** – Completed projects located in the gitea.io repository (mostly based on the EEZ-GUI framework). They can be used as a starting point for creating new projects.
- **Empty** – Creating an empty project for advanced users who want to configure everything themselves from the start.

8.2. Create new project

When creating a new project, the first step is a dialog as shown in Fig. 55.

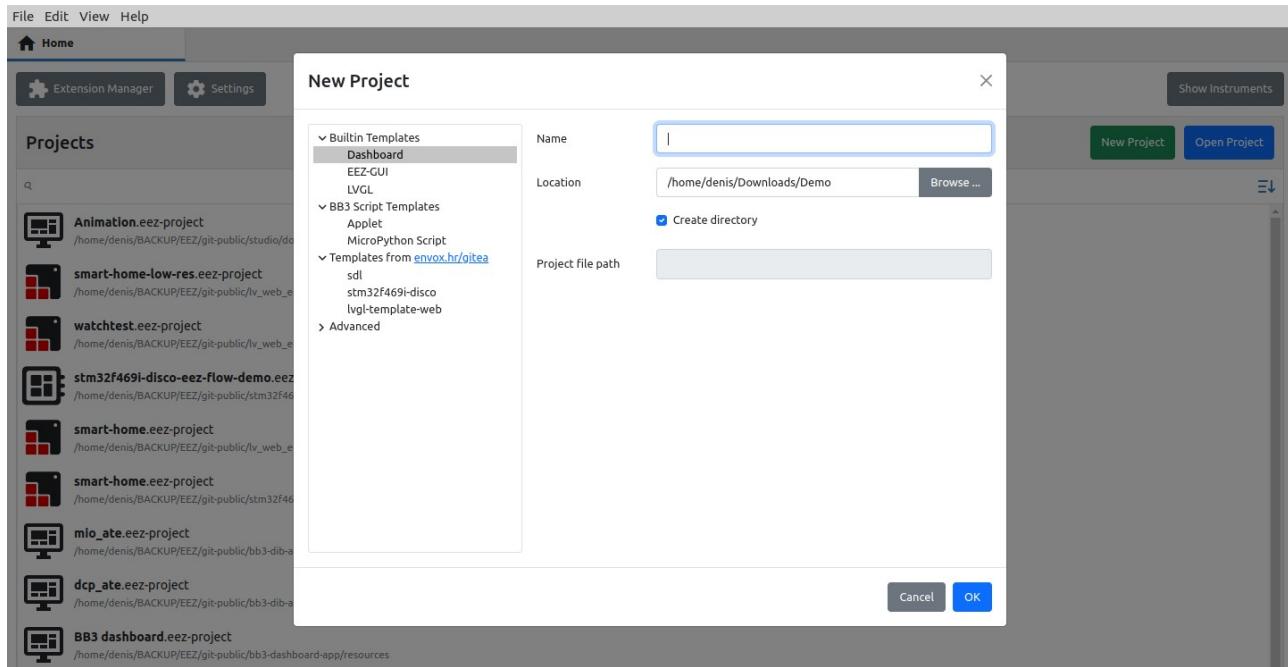


Fig. 55: Create new project

New project type

List of available project types as mentioned in the previous section.

Name

The name of the new project.

Location

The location where the project files will be stored.

Create directory

If selected, a subdirectory (at Location) with the name of the project will be created.

Project file path

Information field (read-only) showing the resulting path in which the new project will be created.

8.3. Additional steps for creating EEZ BB3 projects

New *Applet* and *MicroPython script* projects require access to the EEZ BB3 firmware master project from which exported styles, fonts and themes are used to make the GUI of the newly created application compatible with the EEZ BB3 on which it will be executed.

The necessary EEZ BB3 master project can be downloaded from GitHub (Fig. 56) when creating a new project or set a reference to a local copy of the repository (Fig. 57).

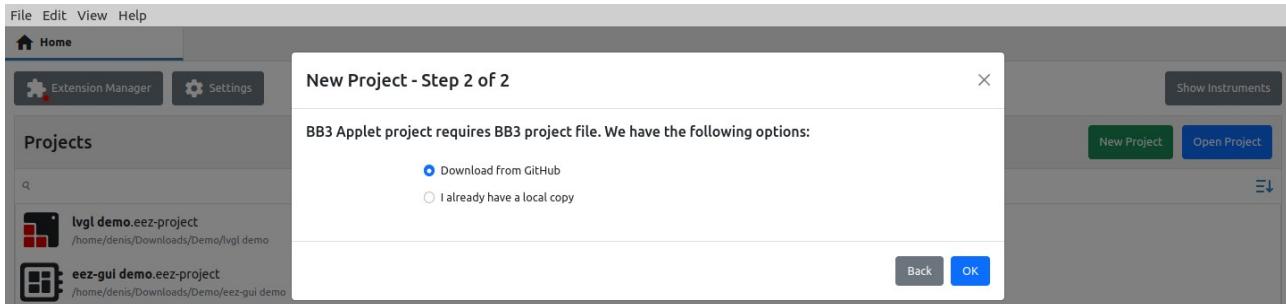


Fig. 56: Selection of EEZ BB3 project for a new project from GitHub

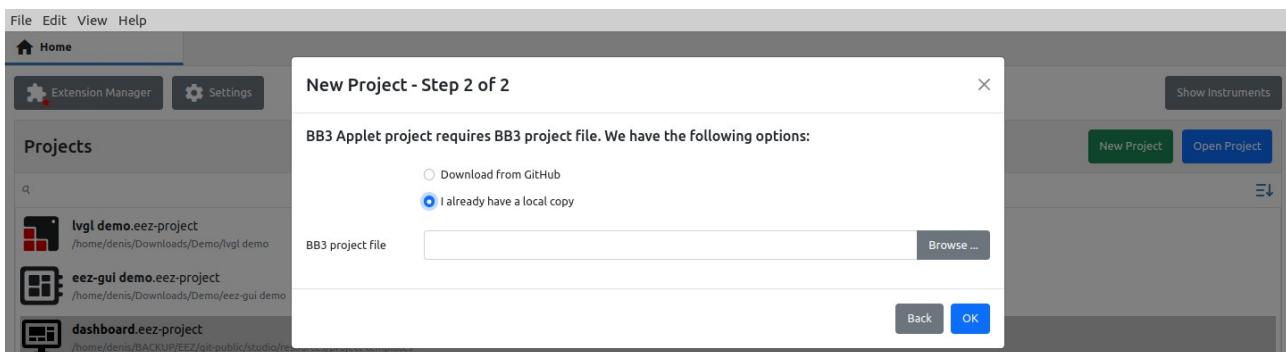


Fig. 57: Selection of EEZ BB3 project for a new project from the local repository

When creating a *MicroPython script* project, it will be necessary in the third step to define which firmware version is used on the target EEZ BB3 in order to create the corresponding resource file during the build (Fig. 58).

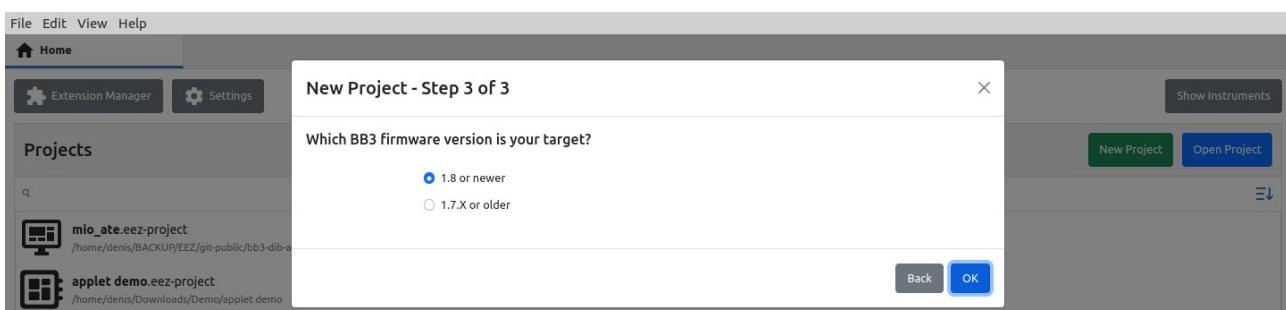


Fig. 58: Selection of EEZ BB3 firmware version for a new project

After all steps have been confirmed, the new project will be displayed in the project editor in *Edit* mode. Fig. 59 shows a new project for the *EEZ BB3 applet*. An overview of the project editor can be found in the next chapter.

The newly created project has the minimum required to be able to execute it in simulation (*Run* or *Debug* mode) or after build on the target platform.

The basic project settings set by the New project wizard can be seen by clicking on the *Settings* option (1) when the project *Settings* will open in a new tab (2) as shown in Fig. 60.

There you can also see *Project features* that have been added and are mandatory, so the *Remove* option is disabled (3), added and can be removed (4) and others that have not yet been added (5).

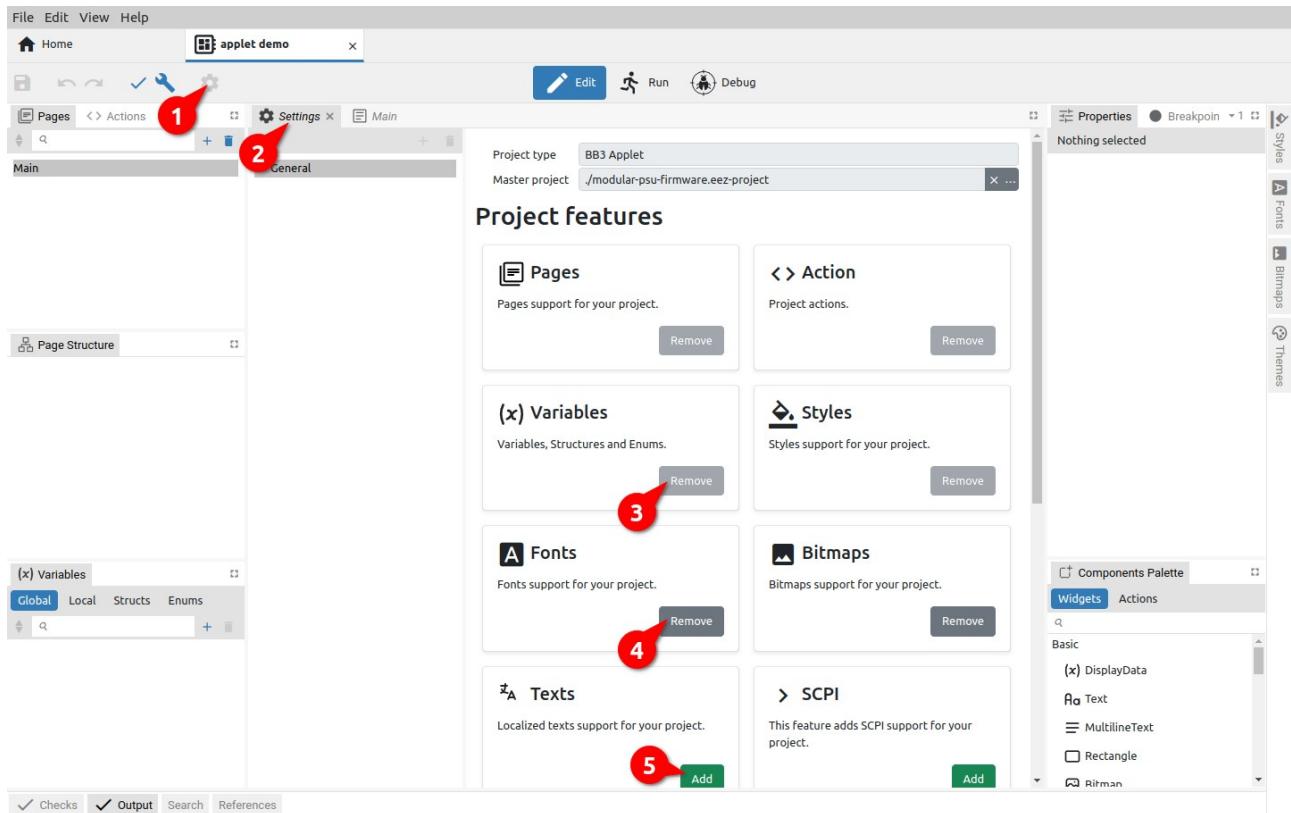


Fig. 60: Newly created project settings

9. Project editor overview

This chapter provides an overview of the basic elements and functions of the Project editor. Their detailed description and content is described in other chapters.

9.1. Project editor workspace

Fig. 61 shows a typical arrangement of Project editor elements. Thanks to its modern design, the Project editor offers users the freedom to rearrange them according to their own needs and taste.

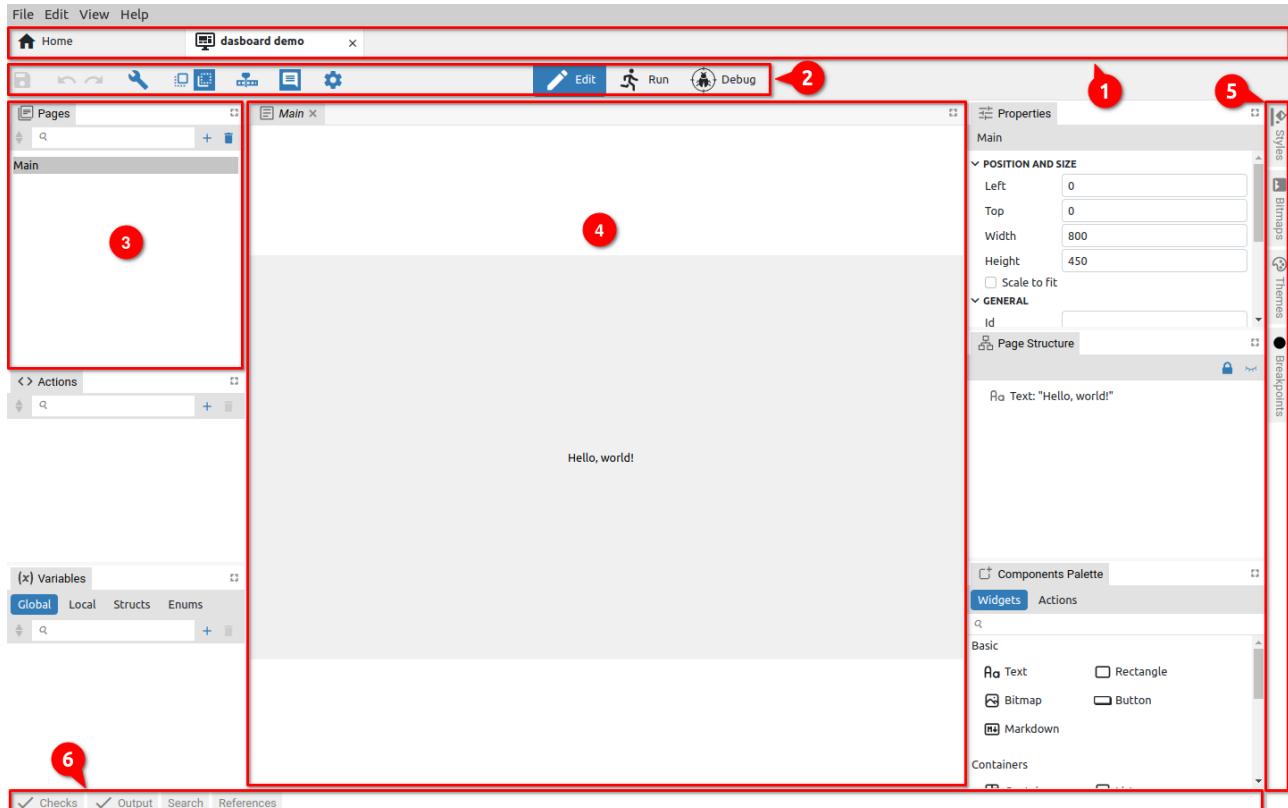


Fig. 61: Project editor sections

All elements of the project editor can be classified into three main groups:

- *Toolbar* – contains icons of basic editor functions, the number of which varies depending on the type of project.
- *Panel windows* – can contain groups of project elements, components and reports e.g. *Pages*, *Actions*, *Styles*, *Fonts*, *Bitmaps*, *Variables*, *Checks*, *Output* (*Build results*), *Search* and *References*. Panels can be grouped within a tabset when they are accessible via tabs labeled with their names.
- *Page editors/viewers* is used to display the page being edited (*Page editors* in Debug mode are *Page viewers* because then the content of the page cannot be edited).

Panels and editors can be grouped within one or more tabssets. Tabssets are dockable and can be placed in the workspace e.g. (3) and (4) or along borders e.g. (5) and (6). The elements of the project editor shown in Fig. 61 are explained below.

#	Section / option	Description
1	<i>Main tabs</i>	Allows easy navigation between multiple open projects (as well as other options that do not belong to the <i>Projects</i> section, i.e. instruments, etc.).
2	<i>Toolbar</i>	List of the main functions of the Project editor and modes (<i>Edit</i> , <i>Run</i> and <i>Debug</i>).
3	<i>Tabset</i>	A dockable section that contains one or more panels.

4 Editor tabset

The place where Pages and Actions are edited. Unlike Actions, Pages also contains GUI elements (Actions can only contain program logic created in EEZ Flow).

5 Right border tabset

An example of a border tabset placed along the right border. By default, it contains panels for styles, bitmaps, themes and breakpoints.

6 Bottom border tabset

An example of a border tabset placed along the bottom border. By default, it contains panels for error checking, build and search lists.

9.2. Display of the page in the editor

In Fig. 62 shows how it is possible to work with multiple editors. To display a page in the editor, click on the desired page (1). A new editor tab will appear, with the name of the selected page in italics (2). This indicates that the tab is not locked and if you choose another page from the list, it will replace the currently displayed one. If we want to lock the page, we will use the right click when the option *Keep Tab Open* (3) will appear. When the page is locked, its name will no longer be displayed in italics (4).

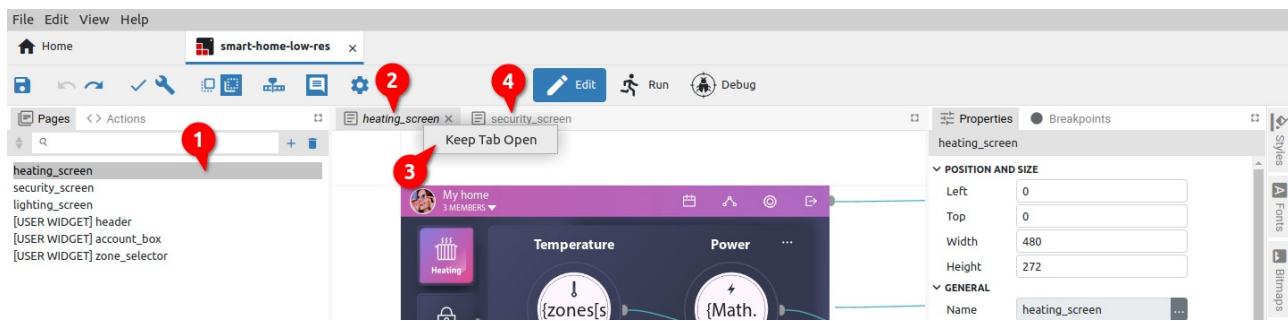


Fig. 62: Page editor tab locking

9.3. Panel moving and docking

Panels and editors can be freely positioned within the workspace or borders and grouped into tabsets.

The key difference between panels and editors is that panels cannot be closed/hidden, unlike editors that open and close as needed depending on how many pages we want to have in the workspace.

Below is an example of how to move the *Actions* panel to another tabset. To begin, click and hold the *Actions* tab (Fig. 63).

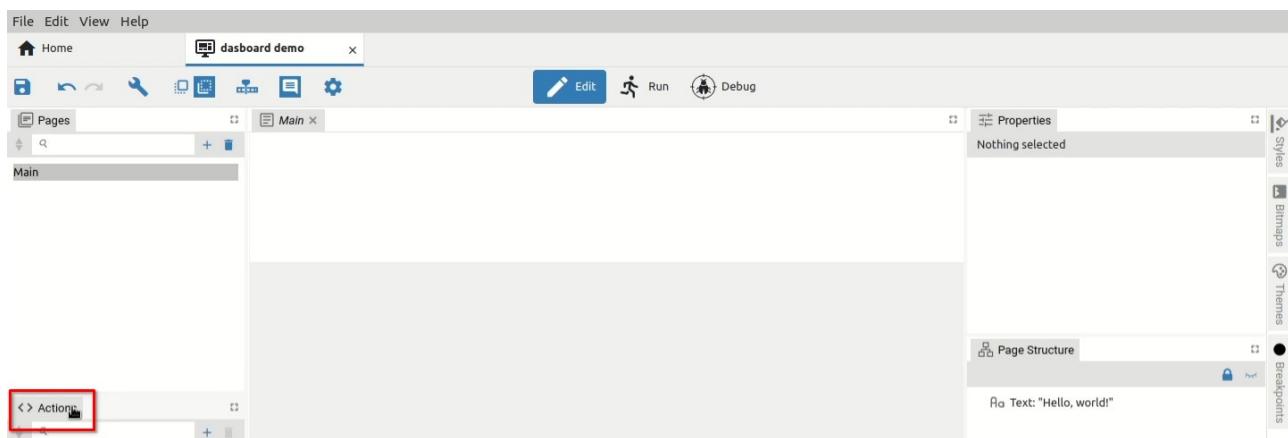


Fig. 63: Panel selection

The panel is now ready to move to another location. The cursor will change and marks will also appear on all four sides indicating the ability to dock into border tabs (Fig. 64).

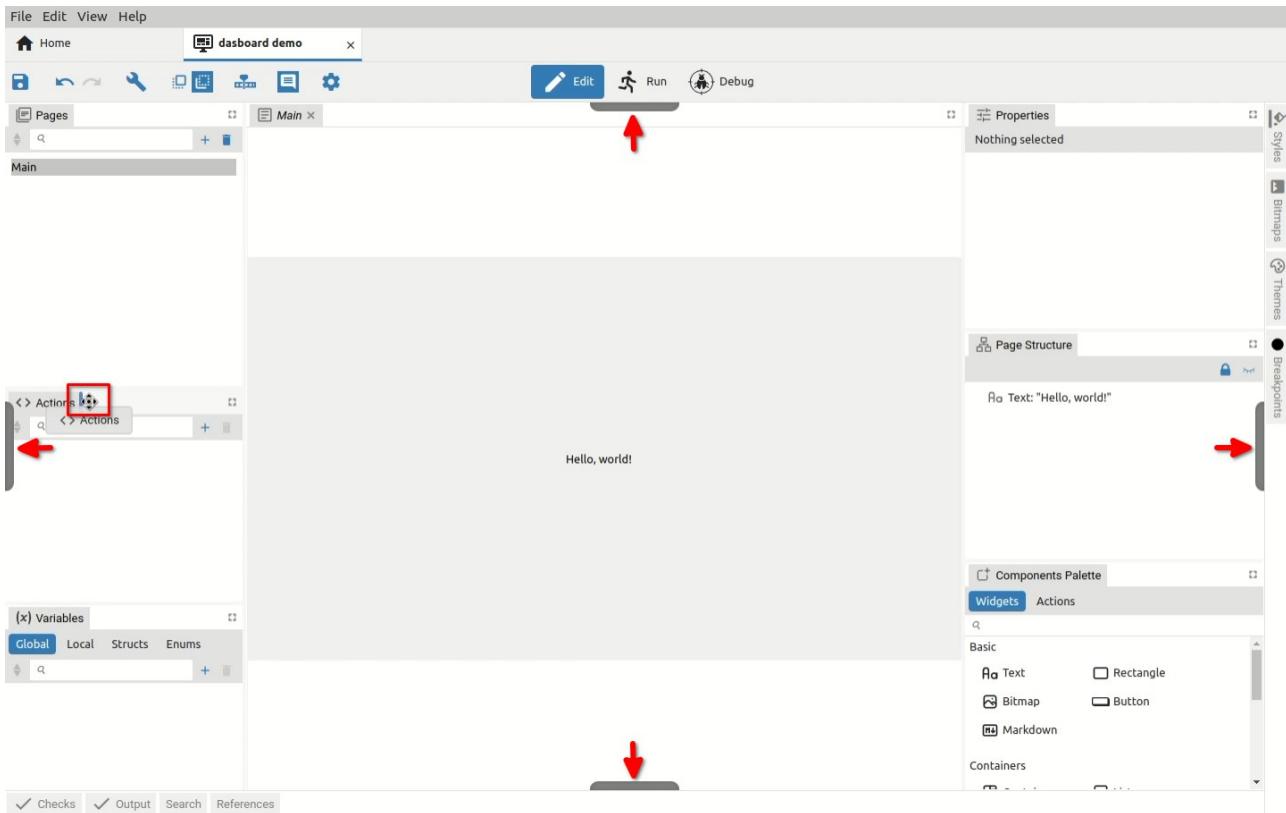


Fig. 64: Docking indicators for border tabs

Now we can choose where we want to dock the panel and whether we want it to become a new tab in the tabset or share the space occupied by an existing tabset. For example, if we want the selected panel to share horizontally the lower part of the space occupied by *Pages*, we will need to move the cursor to the lower part of the *Pages* panel when a rectangle will be displayed as in Fig. 65. Similarly, if we want the selected panel to divide vertically the right part of the space occupied by *Pages*, we will need to move the cursor to the right part of the *Pages* panel until a rectangle is displayed as in Fig. 66.

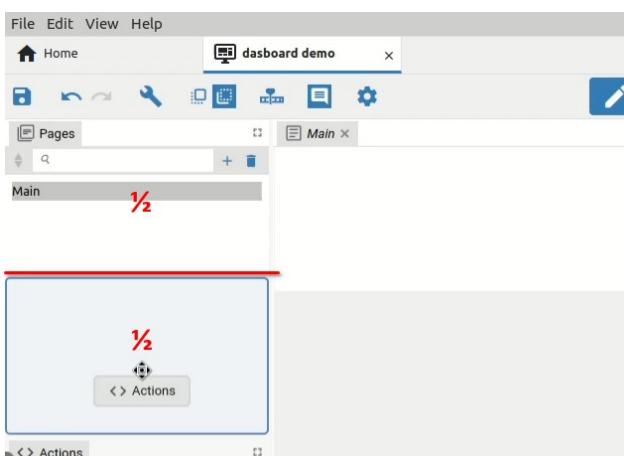


Fig. 65: Panel horizontal positioning

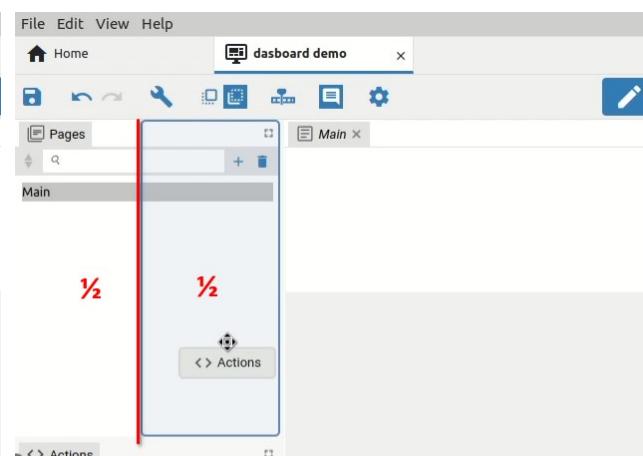


Fig. 66: Panel vertical positioning

The panel can also become a new tab within the existing tabset. This can be done in two ways: by moving the cursor next (left or right) to the existing tab in the tabset as shown in Fig. 67 or to place the cursor approximately in the middle of the existing tab so that a rectangle appears as in Fig. 68.

Note: if we move the cursor closer to the edges of the existing tab, smaller rectangles will appear indicating that the space of the existing tab will be split as shown in Fig. 65 and Fig. 66.

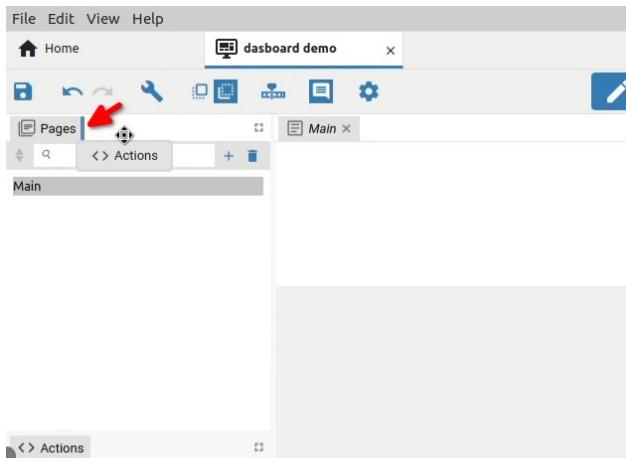


Fig. 67: Positioning in another tabset (1st method)

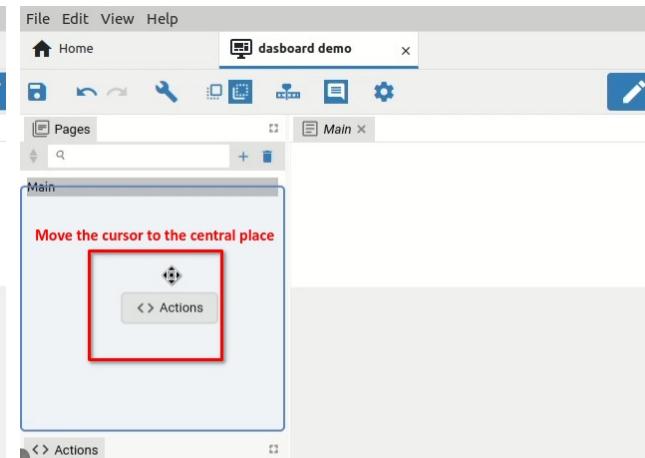


Fig. 68: Positioning in another tabset (2nd method)

Finally, when we have chosen where we want the selected panel to be displayed for docking, it will be necessary to release the mouse button. In our example, Actions will become a new tab in the tabset with Pages (Fig. 69).

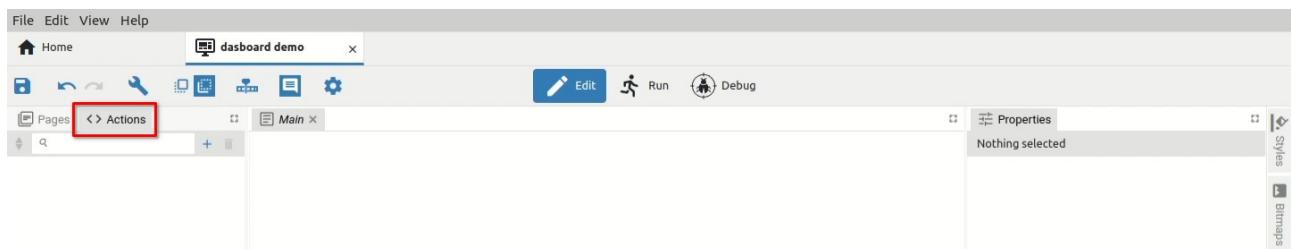


Fig. 69: Panel docking completed

9.4. Border tabs

Panels from border tabssets, unlike panels in the workspace, are displayed by clicking the tab (Fig. 70) and closed by clicking the tab again. Only one panel within a border tabsset can be open at any time (Fig. 71).

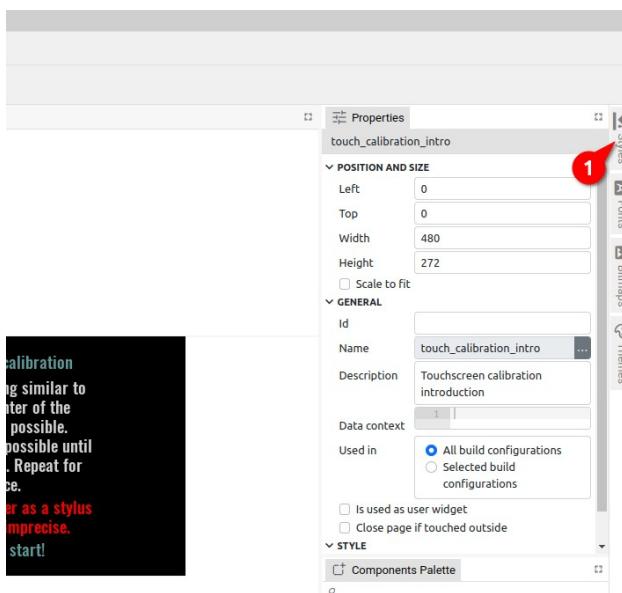


Fig. 70: Border tabs are closed

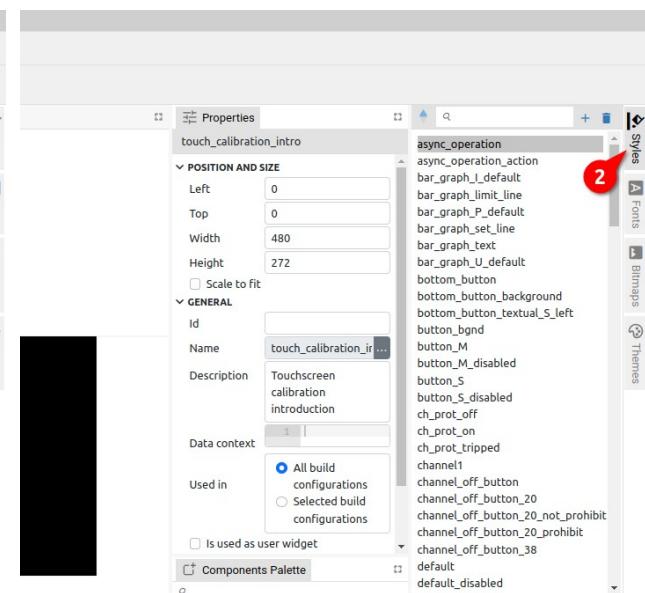


Fig. 71: The border tab is selected and opened

Panel docking is possible in Edit and Debug mode, but in Debug mode it is not possible to dock into border tabssets.

10. Project editor modes

Project editor has three modes: *Edit*, *Run* and *Debug*. The mode selection buttons (i.e. the Mode switcher) are located in the toolbar of the Project editor and will only be displayed if EEZ Flow is used in the project.

While the *Dashboard* project type includes EEZ Flow by default, for *EEZ-GUI* and *LVGL* projects it will be necessary to explicitly set whether or not to use EEZ Flow. To add EEZ Flow to such projects, use the *Flow support* option (Fig. 72).

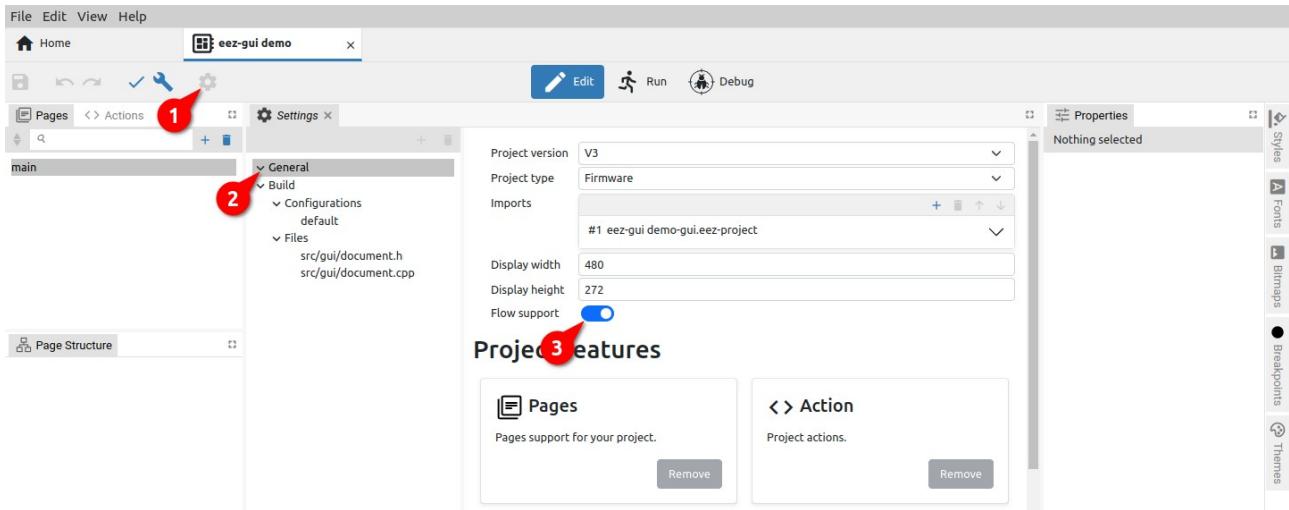


Fig. 72: Enabling EEZ Flow in project Settings

10.1. Toolbar overview

The appearance of the Toolbar depends on the Project editor mode. Certain options are common to all modes, while some depend not only on the current mode but also on the selected *Project features* in *Settings* or the use of global variables when their status will be displayed.

Fig. 73 shows the toolbar with all options displayed. Their availability in each mode is shown in Table 1.

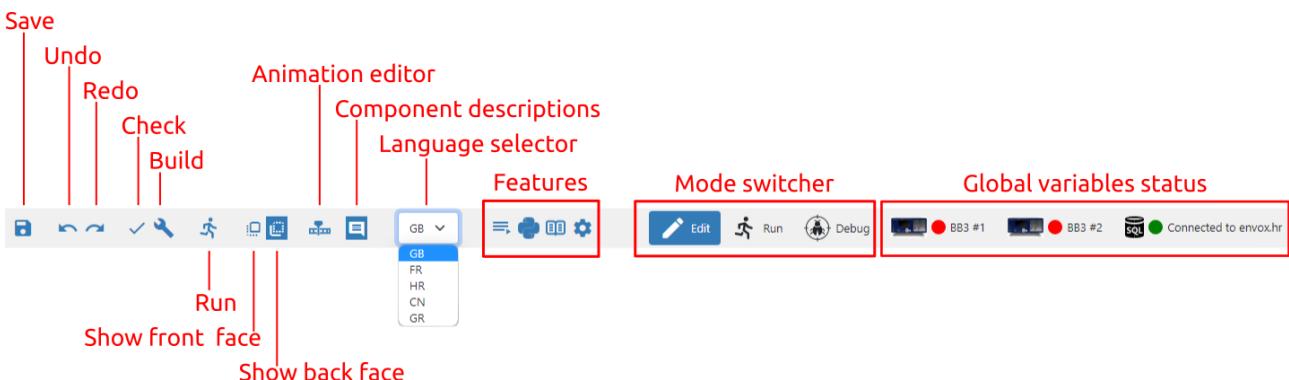


Fig. 73: Project toolbar (all options)

What editors are in *Edit* mode, viewers are in *Run* and *Debug* mode. So we have Page viewer (in *Run* and *Debug* mode) and Action viewer (*Debug* mode only). For example, page viewer displays the page in the same way as the editor, but editing is not possible.

When the Project editor is in *Run* mode, it displays only the toolbar and the active page viewer.

In *Run* and *Debug* mode, it is not possible to change the project, but only to monitor the execution of the project.

Statuses of global variables are present only in *Run* or *Debug* mode and if the project has at least one global variable of type *object*, e.g. *Instrument connection* or *PostgreSQL connection*. Status shows icon, connection state (connected / disconnected) and title. By clicking on the status of the global variable, you can e.g. change the connected instrument or PostgreSQL connection parameters.

Function / Group	Edit	Run	Debug
Save	✓		
Undo	✓		
Redo	✓		
Check	✓		
Build	✓		
Run MicroPython Script (EEZ BB3 only)	✓		
Show front face	✓	✓	
Show back face	✓	✓	
Show / Hide animation timeline editor	✓		
Show / Hide component descriptions	✓		✓
Language selector	✓		
Features	✓		
Mode switcher	✓	✓	✓
Global variables status		✓	✓

Table 1: Toolbar options in all modes

10.2. Toolbar in Edit mode

Undo / Redo

Undo / Redo recent editor Action. If any changes have been made to the project since the last save, a * sign will appear in the project tab next to the name (Fig. 74).



Fig. 74: Indication of unsaved changes

Check

All project elements are checked without building the executable code. The Results are displayed in the *Output* panel (Fig. 75).

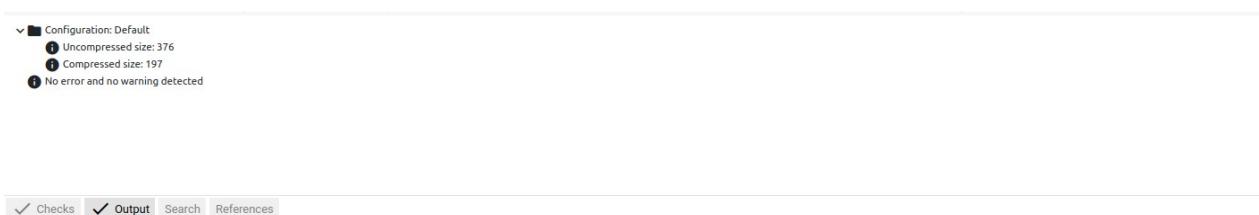


Fig. 75: Results of project checking

Build

Build the executable code after checking all the elements of the project. The results are displayed in the *Output* panel (Fig. 76).

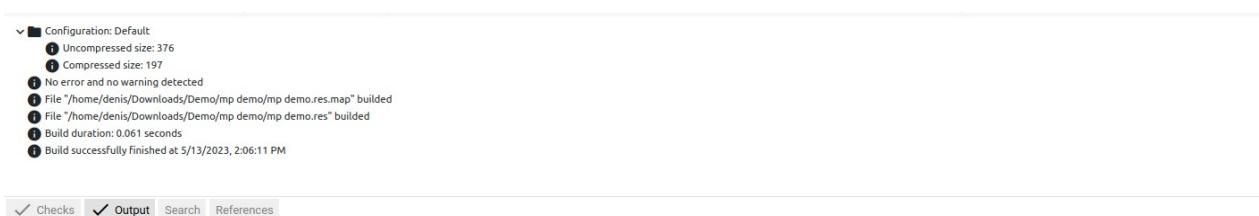


Fig. 76: Project build results

Run MicroPython Script (EEZ BB3 only)

The option is available for a *MicroPython Script* type project that can be executed on an EEZ BB3 device. The *MicroPython* feature should also be selected in the project's general settings.

It starts the build of the project when the accompanying resource file (.res extension) is generated, which will be transferred together with the MicroPython script (.py extension) to the selected EEZ BB3 where it will be started.

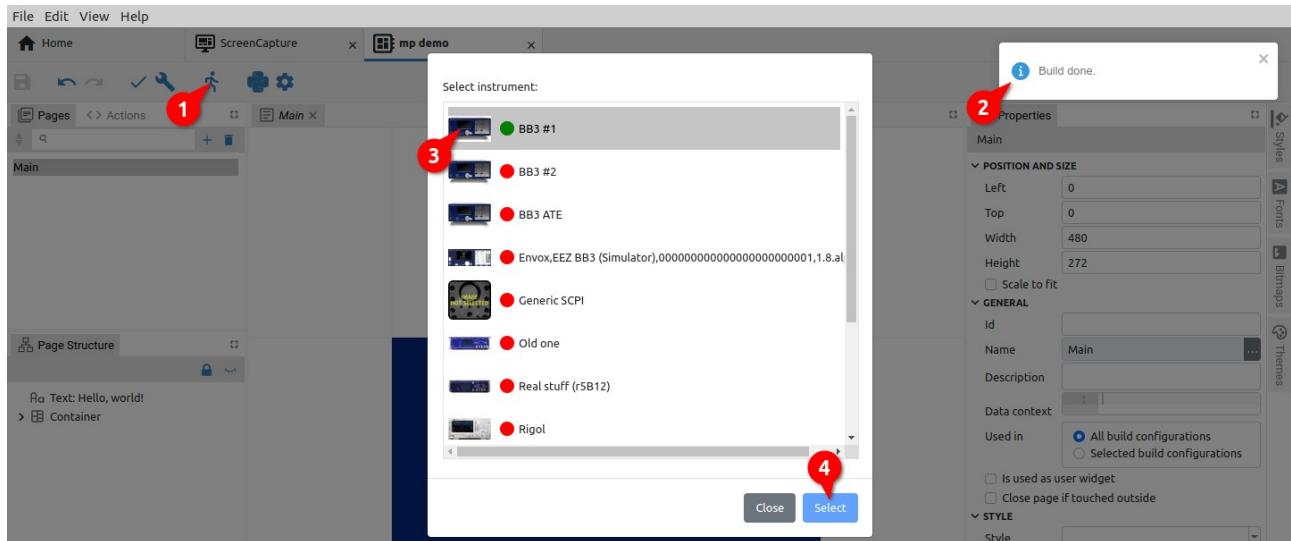


Fig. 77: Selection of target EEZ BB3 for project execution

In case there is no active connection with the selected EEZ BB3, an additional dialog box for establishing the connection will appear. For example in Fig. 78 the serial interface is selected.

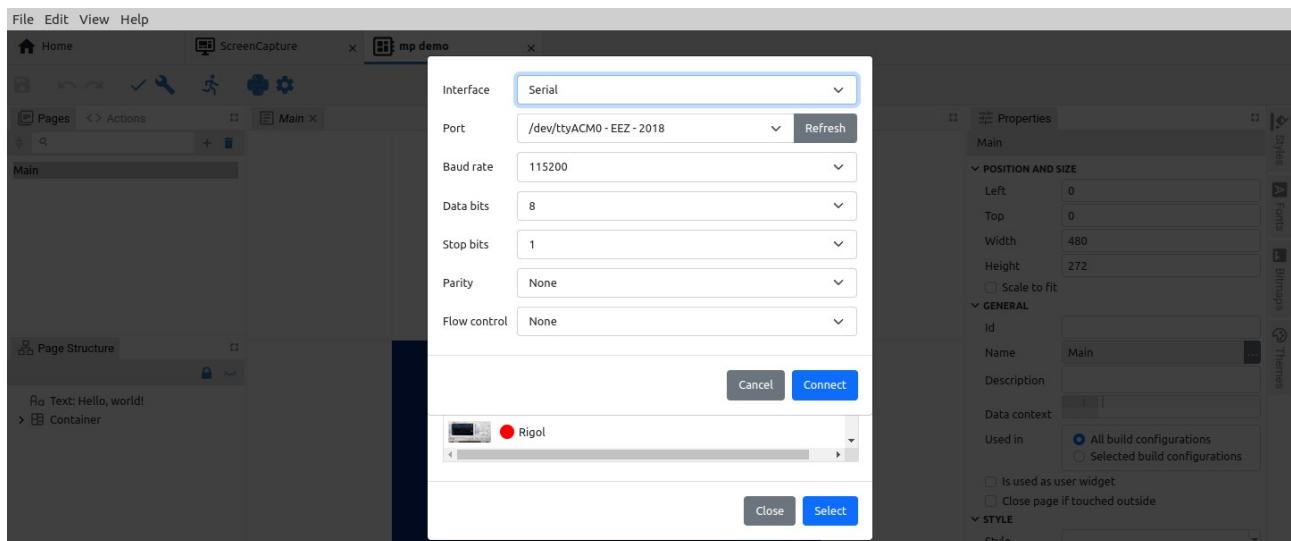


Fig. 78: Selection of target EEZ BB3 for project execution

Finally, after the project files (.py and .res) have been successfully transferred to the selected EEZ BB3, an indication will appear when the MicroPython script has been started (Fig. 79).



Fig. 79: Project execution indication

Show front face

Shows only Widgets without Action components and lines in the page editor for better readability. This button is present if EEZ Flow is enabled in the project and if the page editor is in focus (Fig. 80).

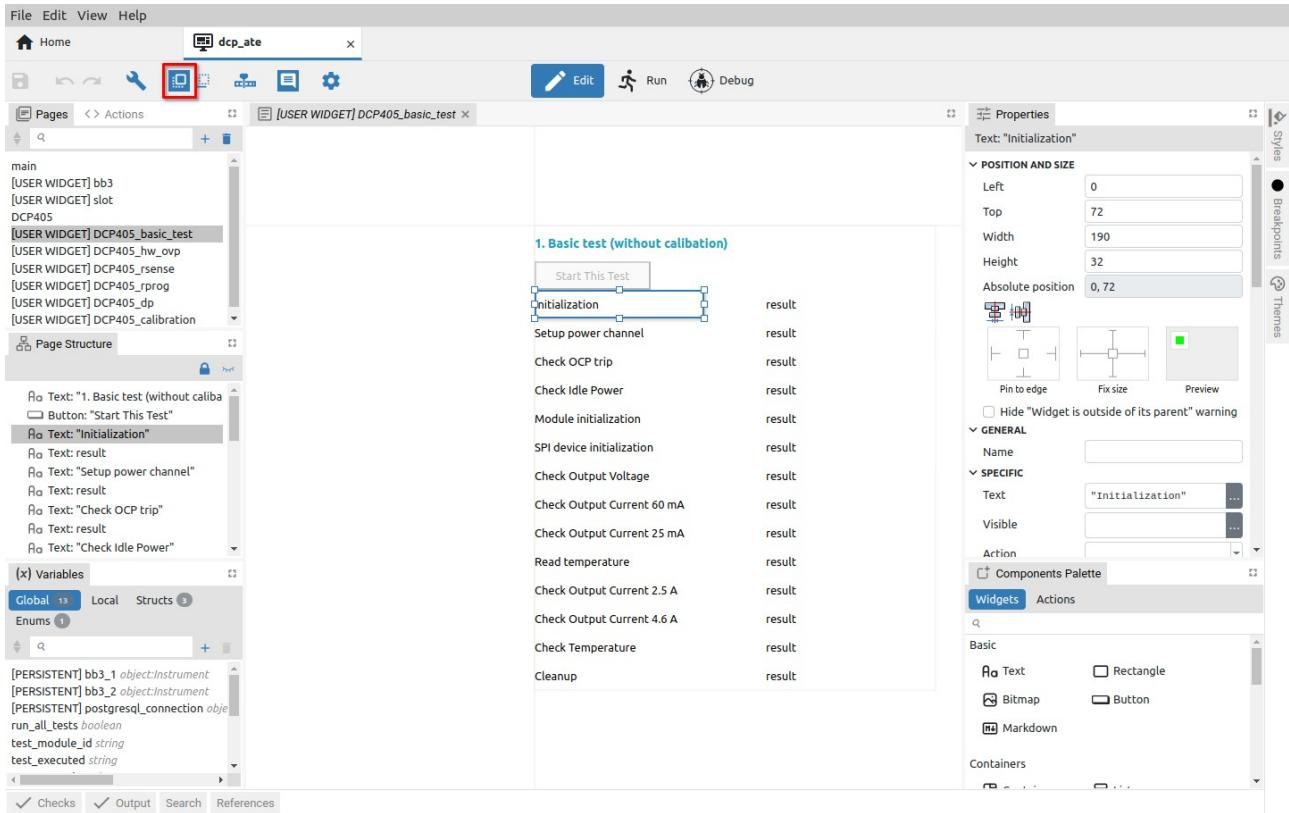


Fig. 80: Project page in front face view

Show back face

Shows all components (both Widgets and Actions) and lines in the page editor. This button is present if Flow is enabled in the project and if the page editor is in focus (Fig. 81).

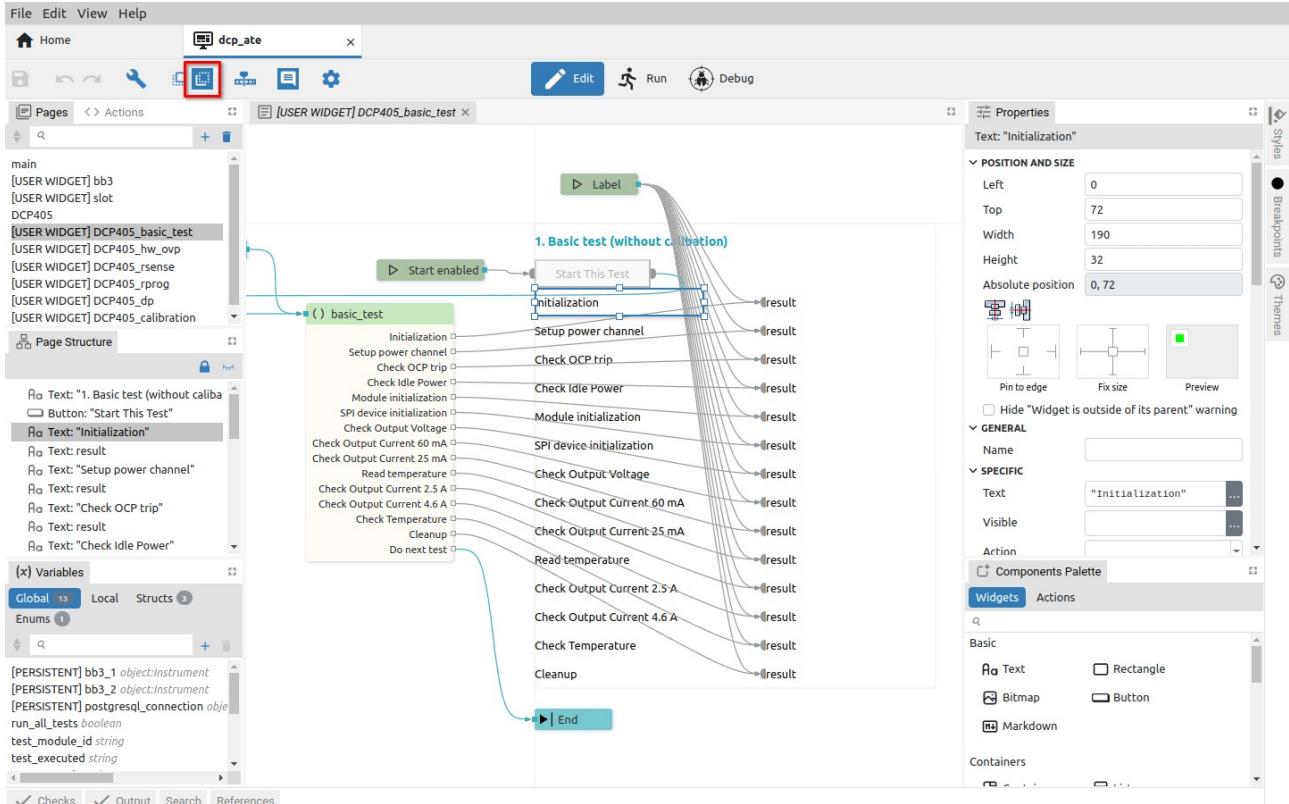


Fig. 81: Project page in back face view

Show / Hide animation timeline editor

EEZ Flow supports animation of page content, for which the Animation timeline editor is used.

An icon in the toolbar to show and hide it will appear when the page editor is in focus. EEZ Flow should also be enabled in the general settings of the project (Fig. 53).

The animation timeline editor is displayed in the space below the page editor (Fig. 82).

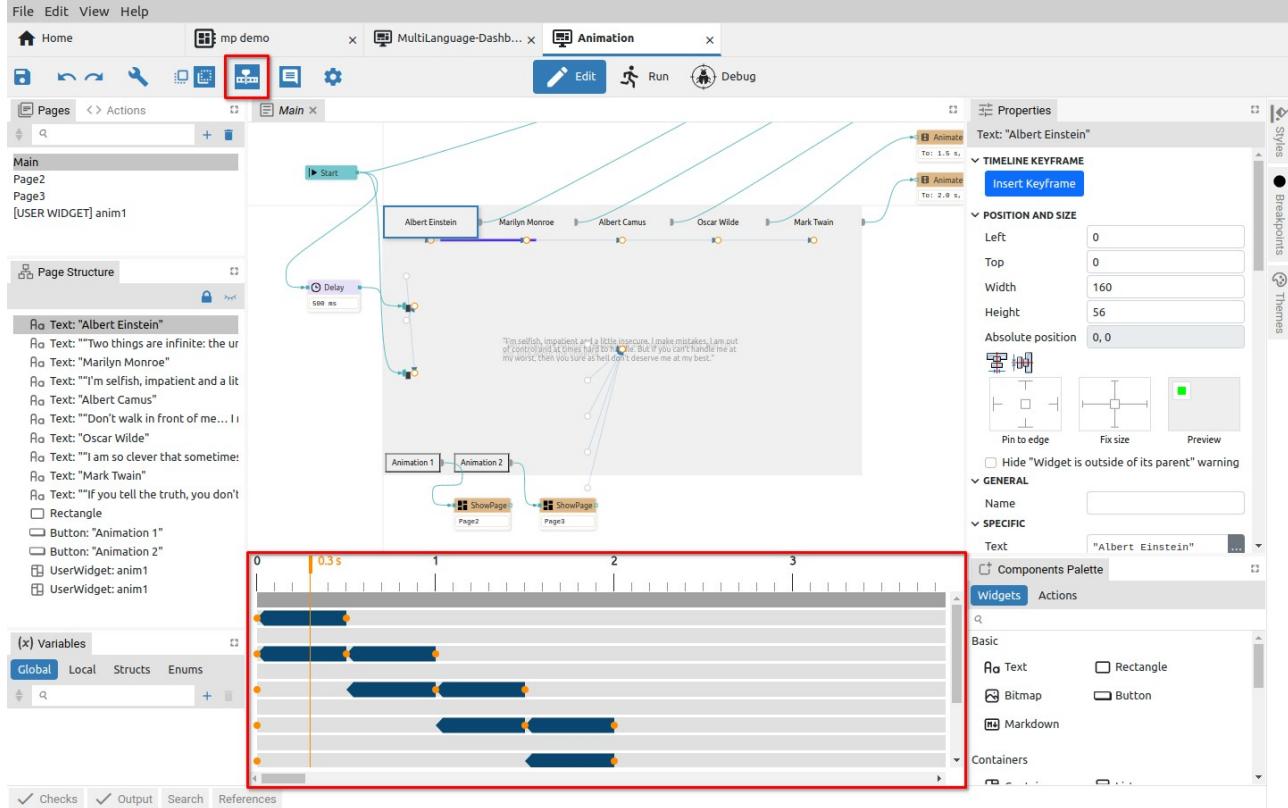


Fig. 82: Project animation timeline editor

Show / Hide component descriptions

Each component has a *Description* property. With this option, we choose whether the description will be seen under the component or not (Fig. 83).

The option is only displayed if the page editor is in focus and *Show back face* is selected.

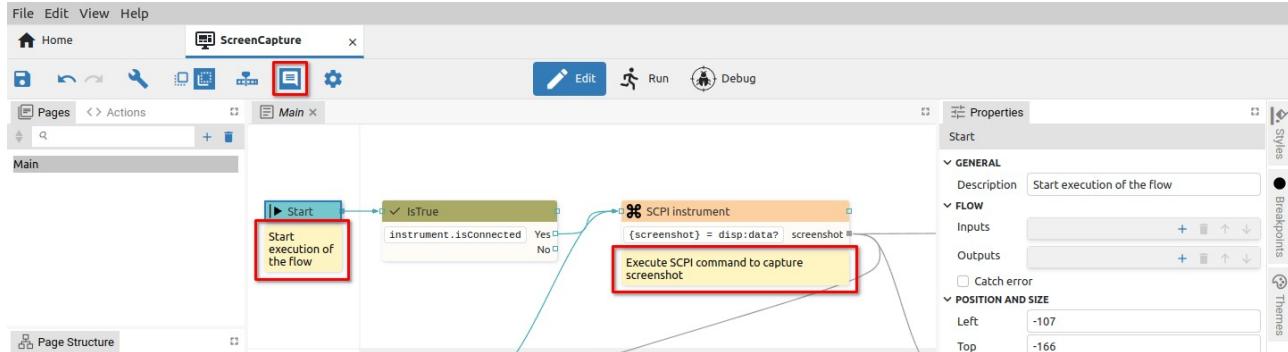


Fig. 83: Component descriptions are visible

Language selector

This option is present in the toolbar if the *Texts* feature is selected in the project general settings and if at least one language is defined. Then the *Texts* tab will appear in the left border tabset (2), whose panel (3) contains definitions of multilingual text strings, used languages and translation statistics.

The texts displayed in the page editor will be displayed in the language selected in the language selector of the toolbar (1). In the example in Fig. 84 French (FR) is selected.

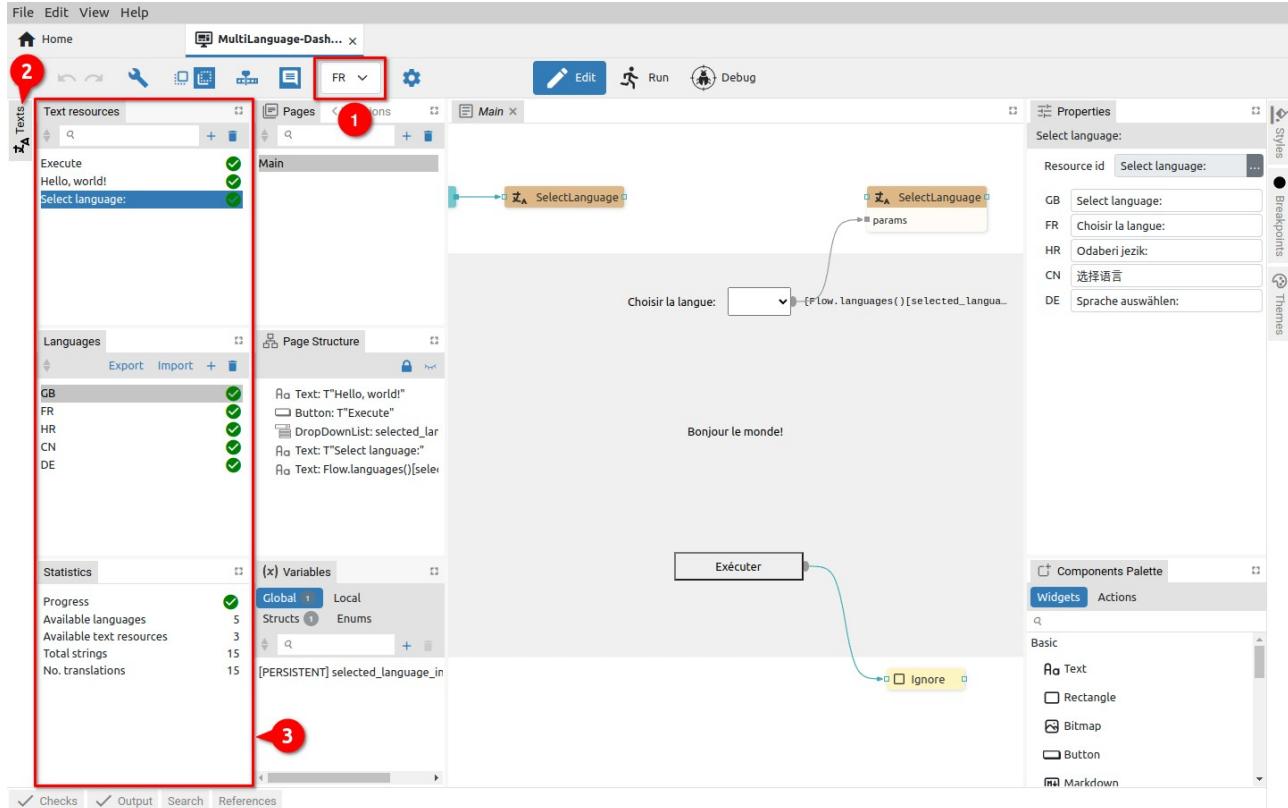


Fig. 84: Project language selector

10.3. Feature buttons

The following project features when selected in the project general settings will add an icon to the toolbar: *Shortcuts*, *MicroPython* and *Readme*. Project *Settings* also has its icon in the toolbar. The mentioned features, when selected, are displayed in the project editor as described in Chapter XX.

11. Project editor panels

11.1. Panel items

Panel items are marked in Fig. 85 and described below. As you can see, different panels can have different number of items.

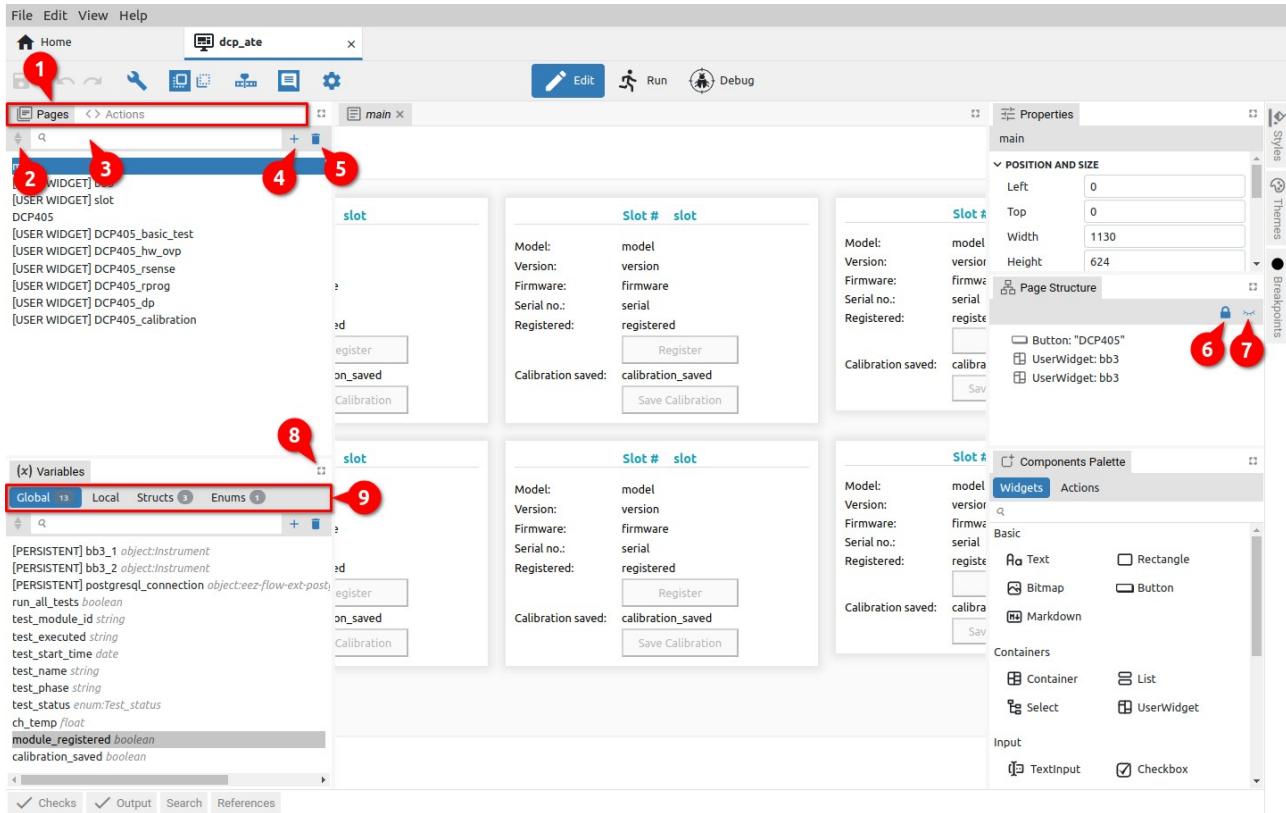


Fig. 85: Panel items

Item

- 1 *Panel tabs*
- 2 *Items sort order*

Description

Selecting panels within a tabset.

Toggle between three sort states: User (both arrows inactive), Ascending (upper arrow active) and Descending (lower arrow active). When the user sort order is selected, which is the default, it is possible to change the position of the item in the list. For that, you need to click and hold on the item you want to move (1), when the appearance of the cursor will change and the background of the item name will change. By moving the cursor, an indication of the new position of the item will be displayed (2) and finally, when the mouse button is released, the item will appear in the new location (3).

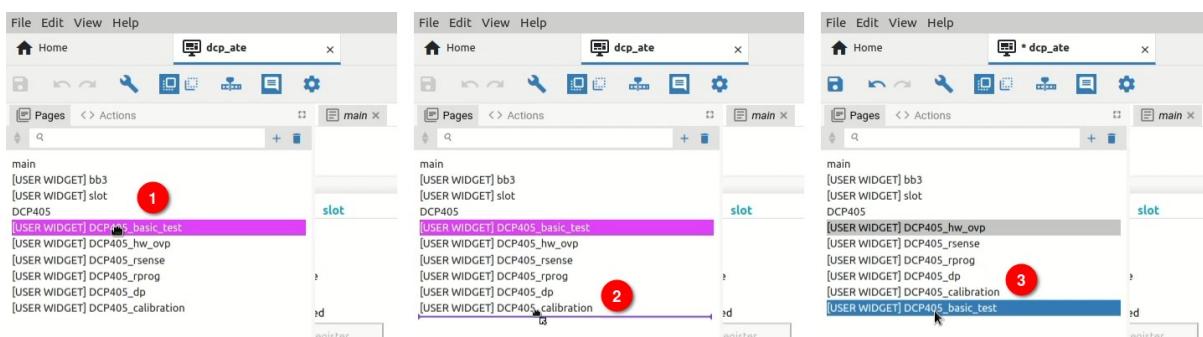


Fig. 86: Changing item position

IMPORTANT: make sure that the first page in the list is the one you want to be attached first when starting the project. The name of the page can be arbitrary (main in the example in Fig. 86).

- | | |
|----------------------|--|
| 3 <i>List filter</i> | Filtering items in the list according to the search term.
If something is entered in the list filter box, then drag & drop in the list of items is disabled when User sorting order is selected. |
| 4 <i>Add item</i> | Adding a new element to the panel. Opens a new dialog box with one or more parameters depending on the type of item. The name of the new item must be unique. Example in Fig. 87 shows the dialog box for adding a new page. |

IMPORTANT: The page name must not contain a dot (.) because when importing, the dot is used as a separator between the name of the external library and the page name.

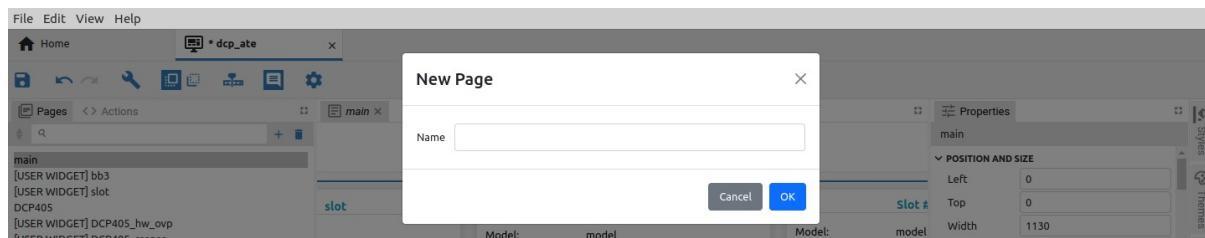


Fig. 87: Adding a new item (Page)

- | | |
|------------------------------------|---|
| 5 <i>Delete selected item</i> | Deleting the selected element from the panel. A deleted item can be restored with <i>Undo</i> option in the Toolbar. |
| 6 <i>Lock All / Unlock All</i> | Lock / Unlock all panel elements. |
| 7 <i>Hide All / Show All</i> | Hide / Show all panel elements. |
| 8 <i>Maximize tabset / Restore</i> | Maximizing tabset display. When maximized, that icon is replaced by <i>Restore</i> . |
| 9 <i>Sub tabs</i> | Certain panels of the Project editor, e.g. <i>Components Palette</i> or <i>Variables</i> use their tabs to organize content. These are displayed as sub-tabs within a tabset. |

11.2. Right-click menu

Right-click opens a context menu that generally contains options as in Fig. 88. The right-click menu for Widgets (Fig. 89) has a few more options and will be described in Chapter xx.

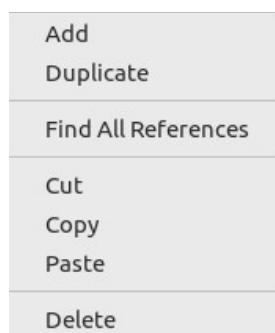


Fig. 88: Right-click menu (common)

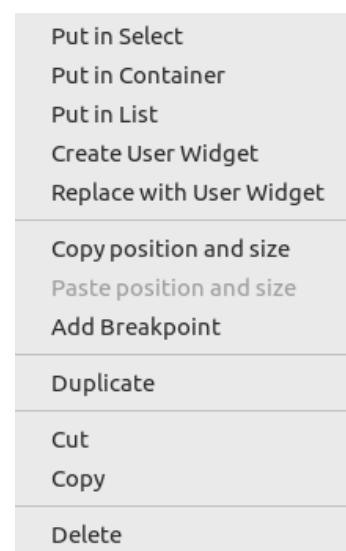


Fig. 89: Right-click menu (Widgets)

Option	Description
Add	Adding a new item. See the description in the previous subsection.
Duplicate	Duplication of items in the list. The name of the duplicated item will be given a numerical suffix: for example, <i>main</i> will be duplicated in <i>main-1</i> .
Find All references	Finding all references to the selected item. The results are displayed in the References panel. Clicking on the reference leads to the place in the project where the item is used.

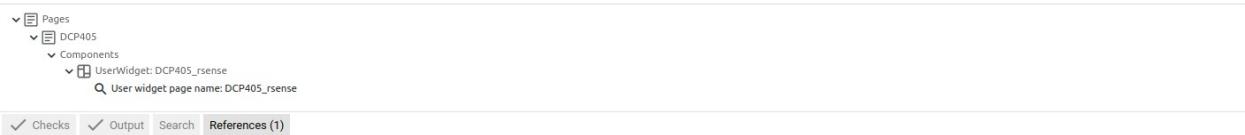


Fig. 90: Displaying the results of the Find all references operation

Cut	Cut (remove) item from list and copy it to clipboard.
Copy	Copy item to clipboard.
Paste	Adding items to the list from the clipboard. This option is hidden if the clipboard is empty.
Delete	Deleting the selected element from the panel. A deleted item can be restored with <i>Undo</i> option in the Toolbar.

11.3. Edit mode panels overview

Panel	Description
Pages	Pages that will be able to be displayed in the GUI. The page at the top of the list will be the first to be displayed at runtime. Pages opened in the tabset editor can be edited.
Actions	Project Actions created in EEZ Flow.
Page structure	List of all Widgets used in the currently selected page in the editor.
Variables	Global and Local variables. Definitions of <i>Structs</i> and <i>Enums</i> types.
Properties	Display and edit properties of the selected item.
Breakpoints	List of all breakpoints where it is possible to enable / disable individual breakpoints.
Components Palette	List of all Widgets and Actions that can be added to a page or Action. The project type determines which Widgets and Actions will appear in the palette.
Styles	All styles for GUI elements.
Themes	Themes are used to easily switch styles and thus change the GUI appearance. When creating a new theme, all styles that currently exist will be added to the new theme.
Bitmaps	List of all imported bitmaps. It will be displayed if the <i>Bitmaps</i> feature is enabled in the project general settings. Bitmaps cannot be edited in the project editor.
Fonts	List of all imported fonts. It will be displayed if the <i>Fonts</i> feature is disabled in the project general settings. The project editor enables basic editing of fonts.
Texts	Localizing texts for multilingual GUI. It will be displayed if the <i>Texts</i> feature is disabled in the project general settings. The localization of the texts is described in Chapter XX.
IEXT (EEZ-GUI only)	Definition of IEXT extension. It will be displayed if the <i>IEXT defs</i> feature is disabled in the project general settings. One project can define multiple IEXT extensions. The IEXT creation procedure is described in Chapter XX.

<i>SCPI (EEZ-GUI only)</i>	List of SCPI commands that will be accessible in IEXT. It will be displayed if the <i>SCPI</i> feature is disabled in the project general settings.
<i>Shortcuts (EEZ-GUI only)</i>	List of Shortcuts that will be accessible in IEXT. It will be displayed if the <i>Shortcuts</i> feature is disabled in the project general settings.
<i>Changes</i>	List of all commits if the project is in a git repository. It will be accessible if the <i>Changes</i> feature is disabled in the project general settings.
<i>Checks</i>	The project editor is constantly looking for errors in the project (e.g. wrong expressions) in the background, and the errors found will be listed in this panel.
<i>Output</i>	It shows the report after the build is complete and the errors found.
<i>Search</i>	Content search and replace. See Section 11.3.1.
<i>References</i>	The following items can be found where they are all used in the project: Variables, Struct, Enum, Page, Action, Style, Font, Bitmap. Right click on the object and "Find all references", the found references will be displayed in this panel (see Fig. 90).

11.3.1. Search and Replace

The *Search* panel allows you to search for a project according to the given criteria with the replace option.

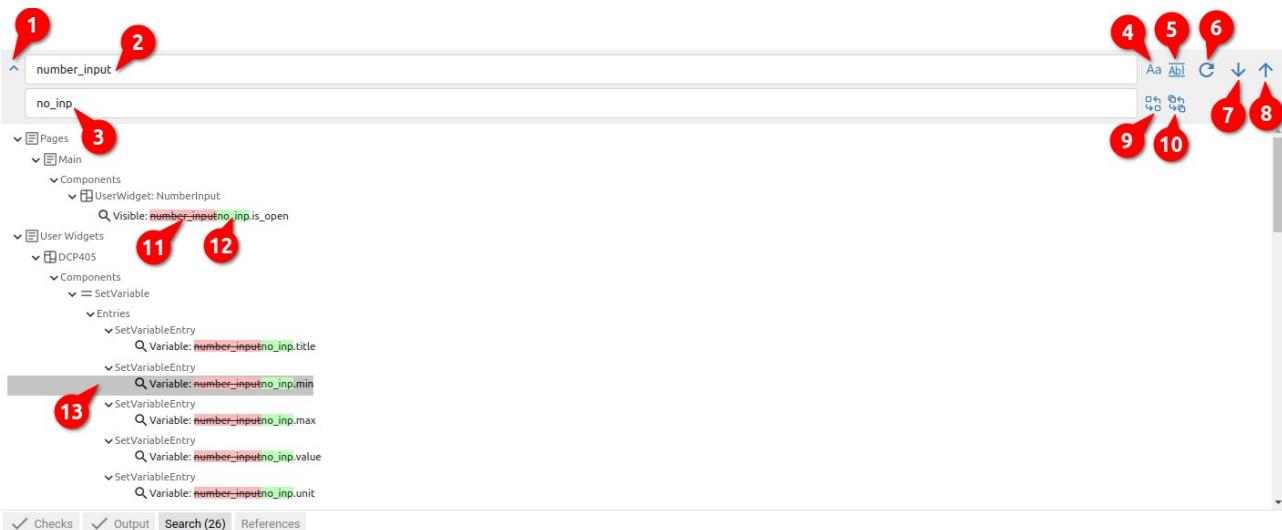


Fig. 91: Search and Replace panel

#	Item	Description
1	<i>Toggle Replace</i>	Shows or hides the Replace field (3).
2	<i>Search</i>	Searched content. Criteria (4) and (5) are taken into account during the search.
3	<i>Replace</i>	New content with which search content is to be replaced.
4	<i>Match Case</i>	Searching for case-correct content.
5	<i>Match Whole Word</i>	Searching for the whole word.
6	<i>Refresh Search Result</i>	Refreshing the results after changing the criteria.
7	<i>Next Result</i>	Move to the next result in the found list.
8	<i>Previous Result</i>	Move to the previous result in the found list.
9	<i>Replace Selected</i>	Content replacement only for the selected item from the found list.
10	<i>Replace All</i>	Content replacement for all items from the found list.
11	<i>Original content</i>	Mark of the original content that will be replaced by the new one.

- 12 *Replaced content* Mark of newly added content.
- 13 *Selected item* The currently selected item that can be replaced using option (10) or from which it can be moved to the next item with option (7) or the previous (8) item in the list.

11.4. Debug mode panels overview

Panel	Description
<i>Pages</i>	Display of all project pages without the possibility of editing.
<i>Actions</i>	Display of all project Actions without the possibility of editing.
<i>Active Flows</i>	List of active Flows.
<i>Watch</i>	Display of all variables and their current values during Flow execution.
<i>Queue</i>	Display all components queued for execution.
<i>Breakpoints</i>	List of breakpoints.
<i>Logs</i>	View logs during execution. Supported log types: <i>Fatal</i> , <i>Error</i> , <i>Warning</i> , <i>Info</i> , <i>Debug</i> and <i>SCPI</i> . It is possible to filter the display according to the given criteria.

12. Project editors/viewers

12.1. Editors

The central part of the workspace represents editors tabssets in which it is possible to edit one or more pages, project features (such as project settings, etc.) or Actions. When the EEZ Flow is enabled in the project, the editors for pages and Actions are displayed in *Edit* mode. In this chapter, you can find an overview of all editors and viewers.

12.1.1. Page editor

The displayed page in editor also has two auxiliary lines that determine the left and top borders, and the starting point of the page ($x = 0$, $y = 0$) is at the top left (Fig. 93).

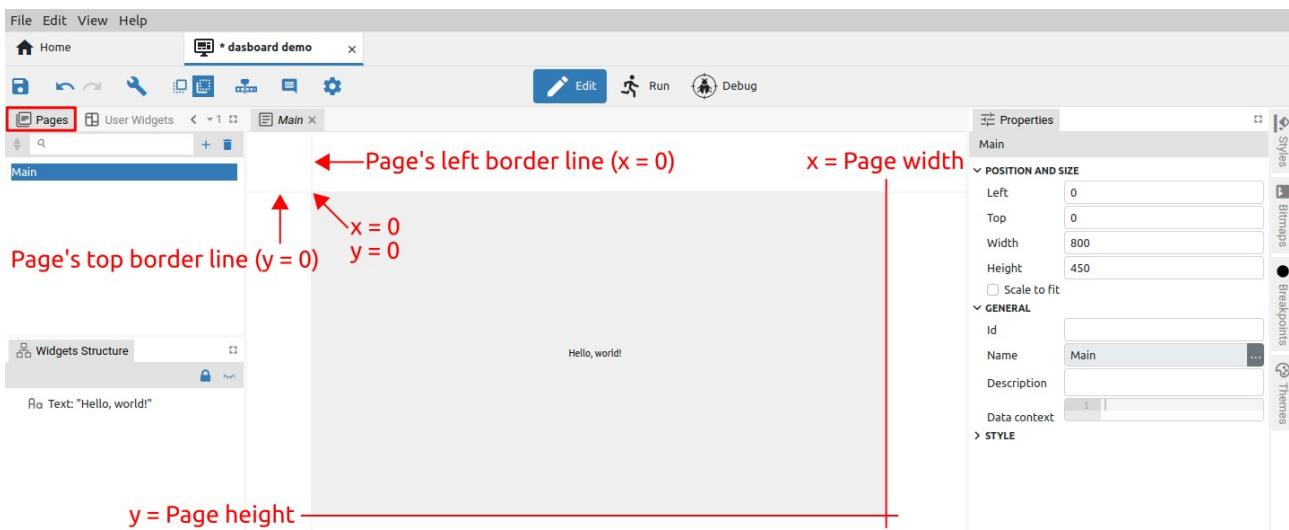


Fig. 92: Display of the page in the editor

12.1.2. User Actions

The User Actions editor allows editing the selected Action from the User Actions panel (Fig. 93).

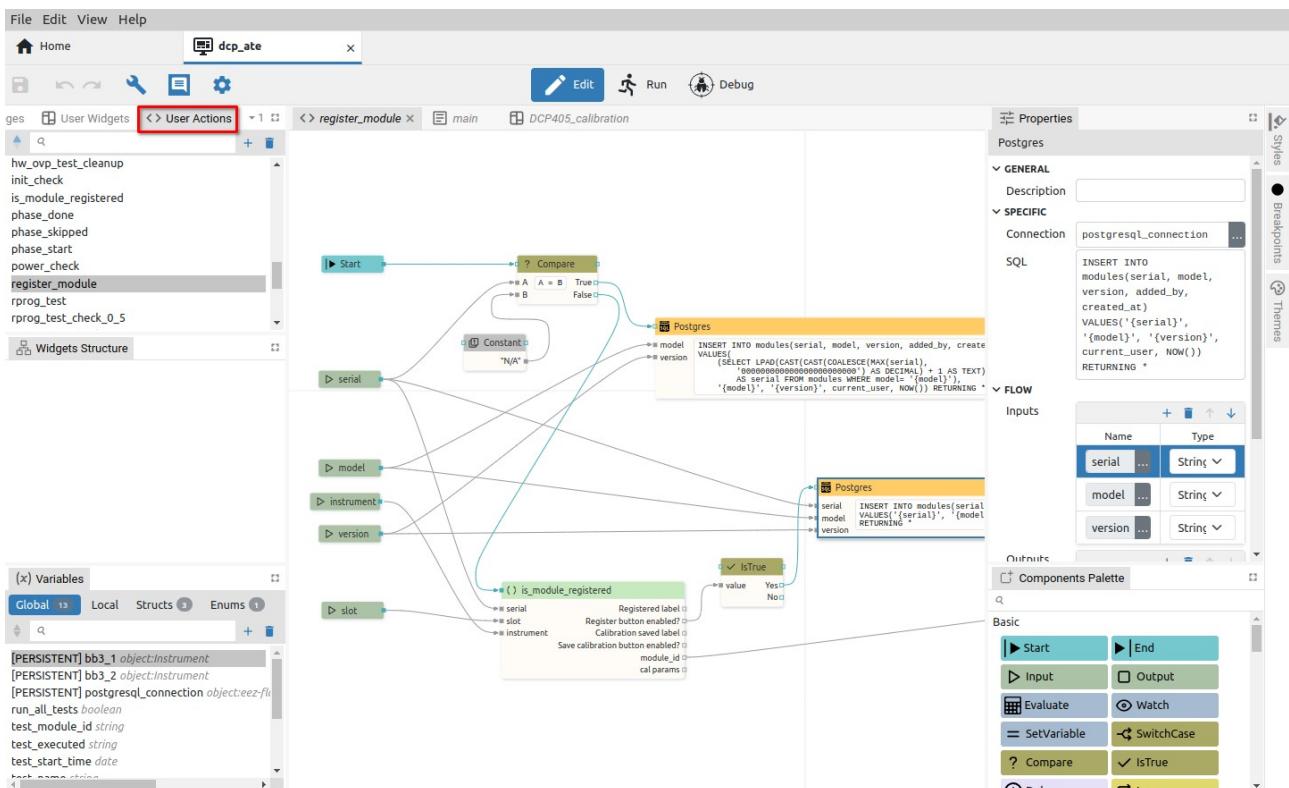


Fig. 93: User Actions editor page

12.1.3. User Widgets

The User Widgets editor allows editing the selected Widget from the User Widgets panel (Fig. 94).

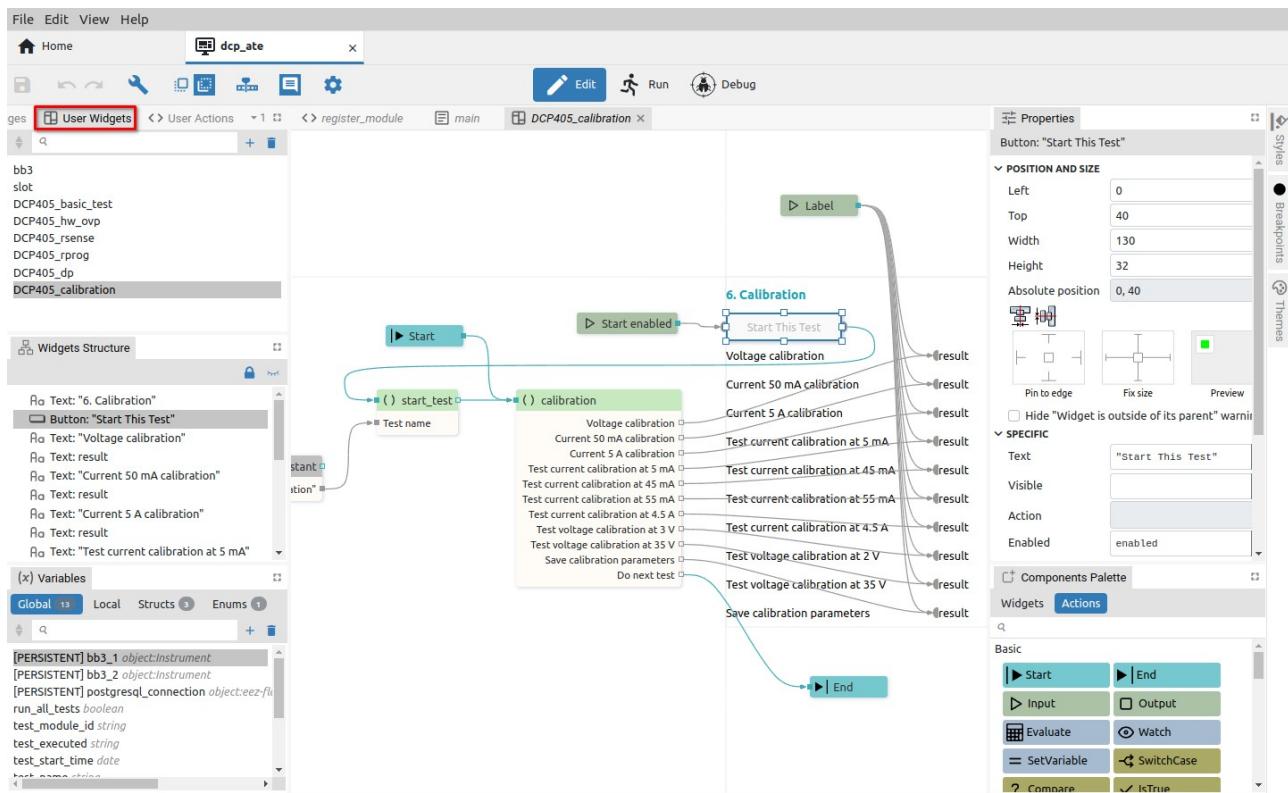


Fig. 94: User Widgets editor page

12.1.4. Font editor

Display of all characters in the font. It also enables the subsequent addition of a new character or the deletion of an existing one. The project will have a Fonts panel if the *Fonts* feature is selected in the project general settings.

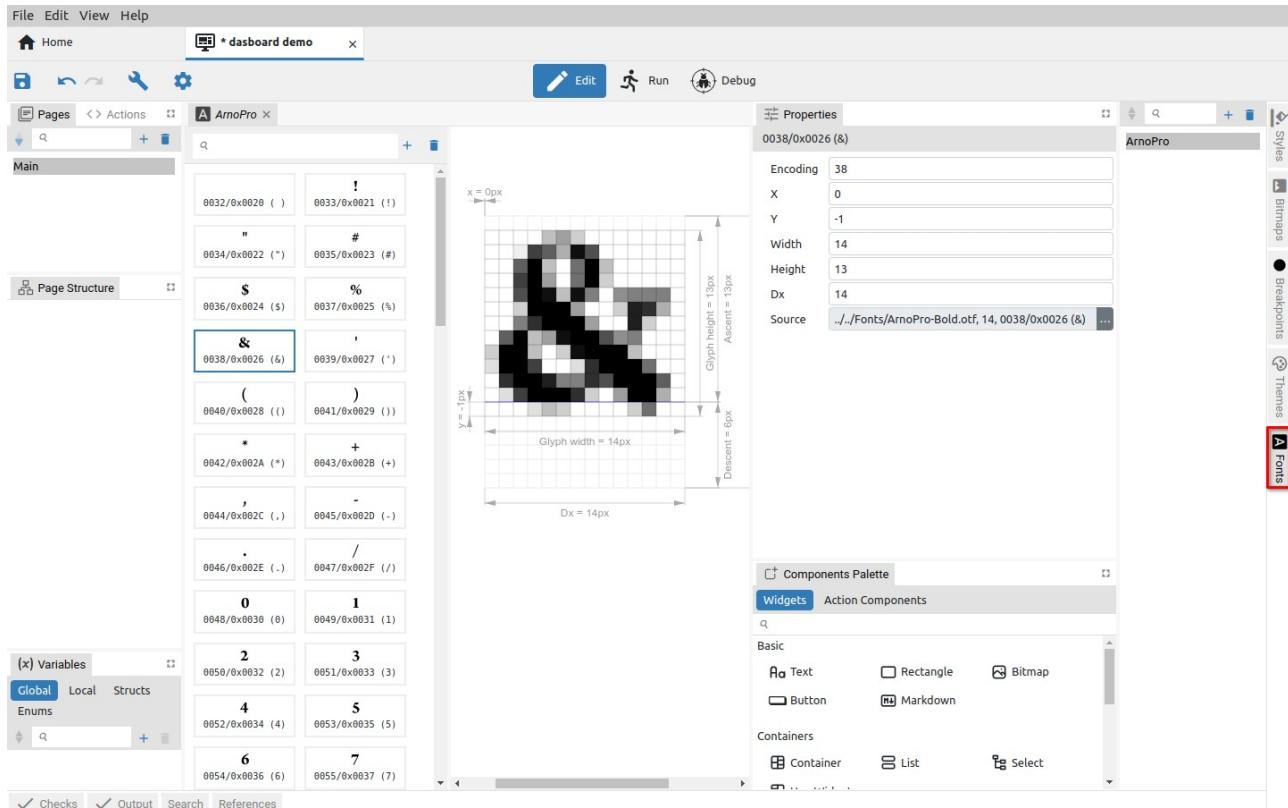


Fig. 95: Font editor page

12.1.5. Shortcuts (EEZ-GUI only)

An *EEZ-GUI* project that includes Instrument Extension definitions (*IEXT defs* feature) can also have the *Shortcuts* feature enabled. In this case, the *Shortcuts* icon appears in the toolbar, which is used to display the page for defining shortcuts in the editor tabset (Fig. 96).

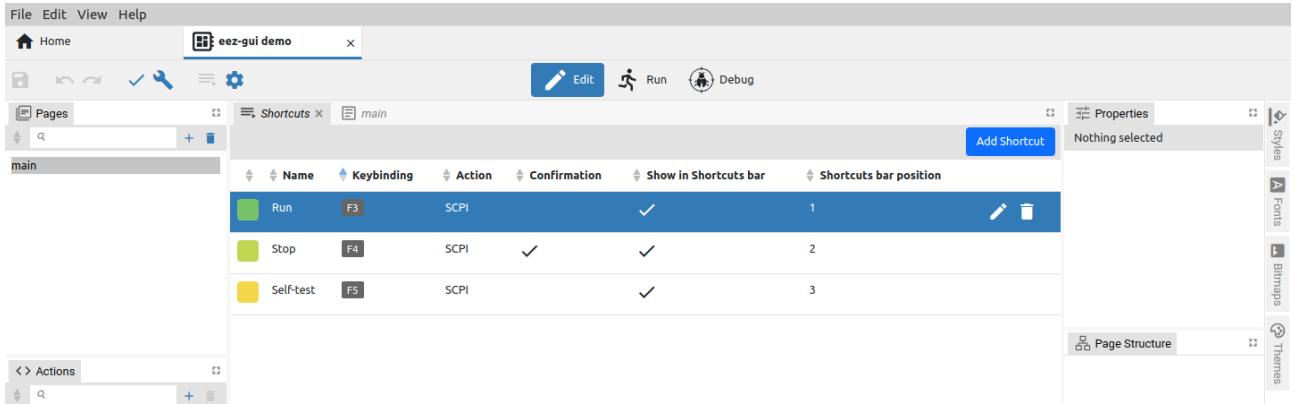


Fig. 96: Project shortcuts page

12.1.6. MicroPython (EEZ BB3 only)

Opens the MicroPython text editor page (Fig. 97).

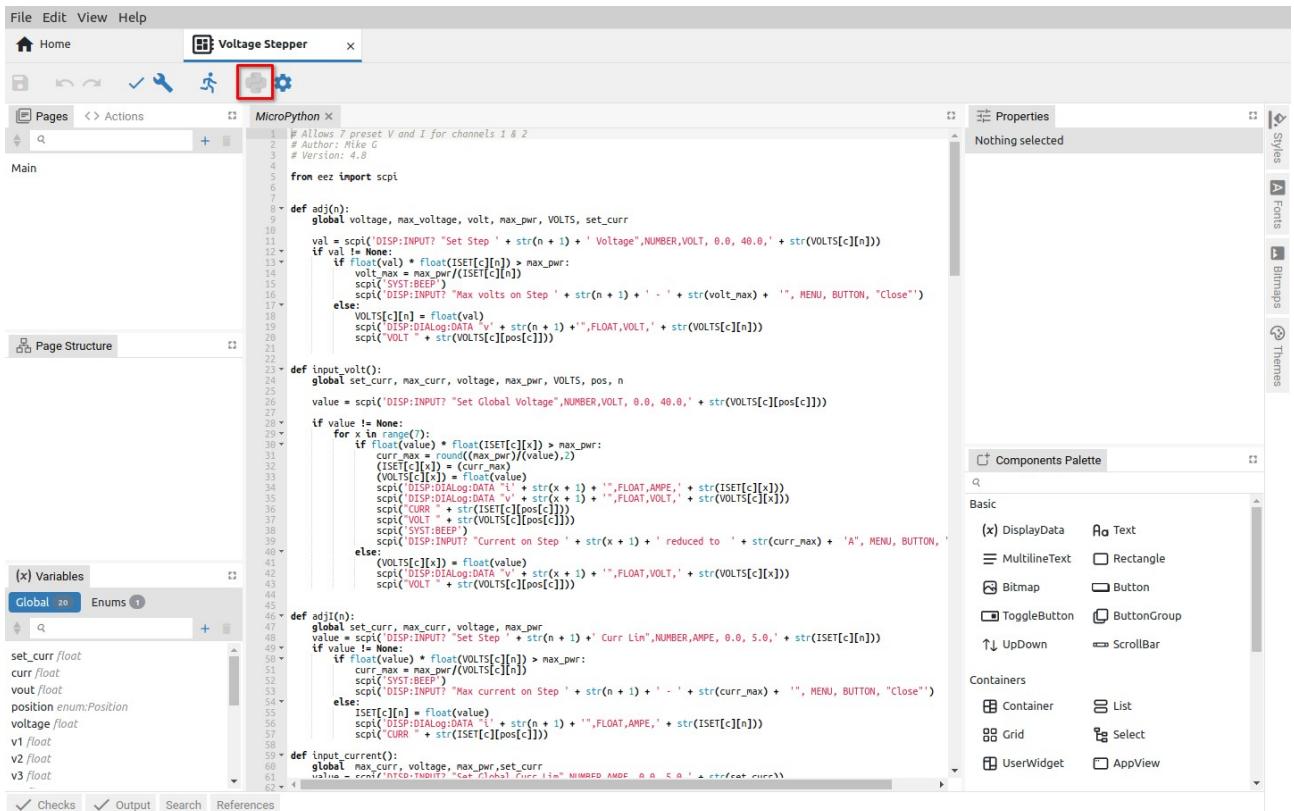


Fig. 97: MicroPython editor page

12.1.7. Readme

The project will have a *Readme* file if the *Readme* feature is selected in the project general settings. It can be used to add clarifications or reminders, e.g. how to build a project for the native platform, information about the target platform, etc.

The *readme* file defined in *Properties* can be displayed but not edited. The *readme* file can be removed (1) or its file path can be selected (2). Text (.txt) and markup (.md) file types are supported.

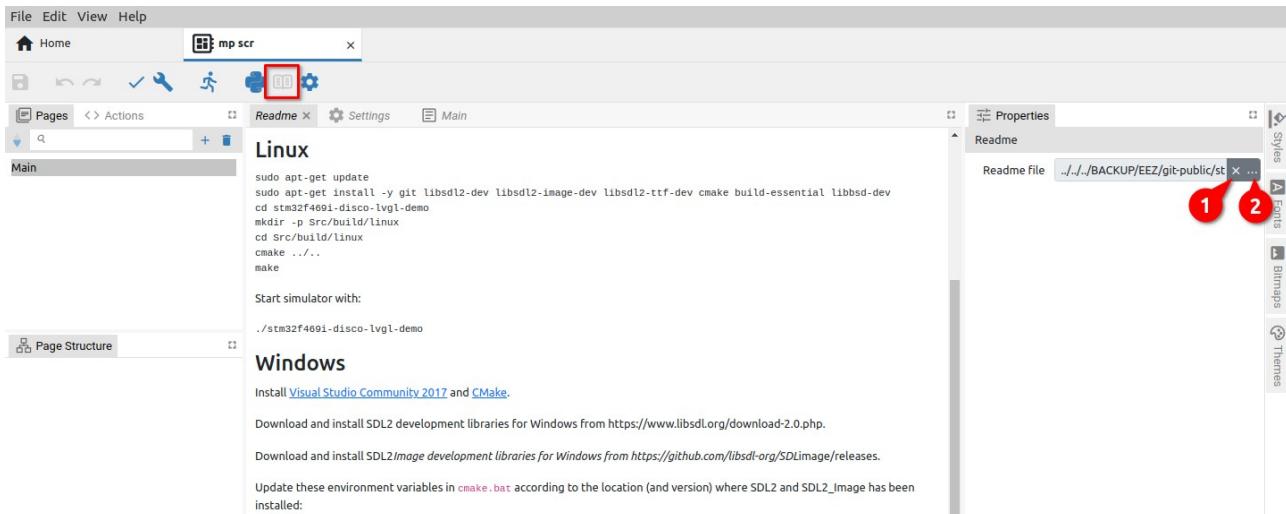


Fig. 98: Project Readme file

12.1.8. Settings

The Settings page is used to edit the global parameters and features of the project (Fig. 99).

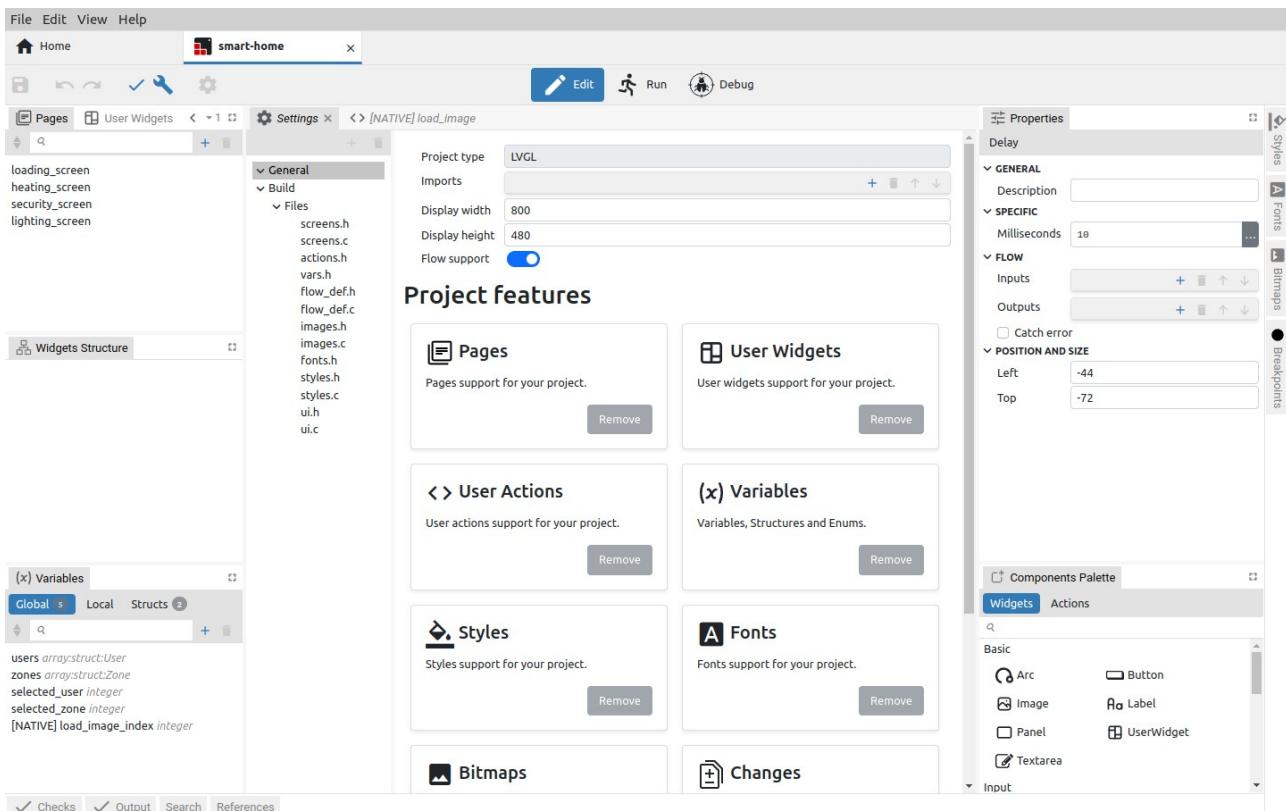


Fig. 99: Project settings page

12.2. Viewers

12.2.1. Page viewer

In *Run* mode, it is possible to see only the viewer of the currently active page (Fig. 100). In *Debug* mode, pages and Actions cannot be edited, so editors effectively become viewers (Fig. 101).

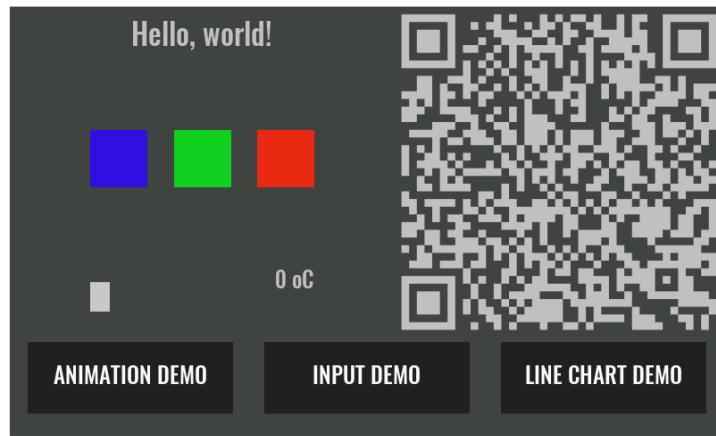


Fig. 100: Page view in Run mode

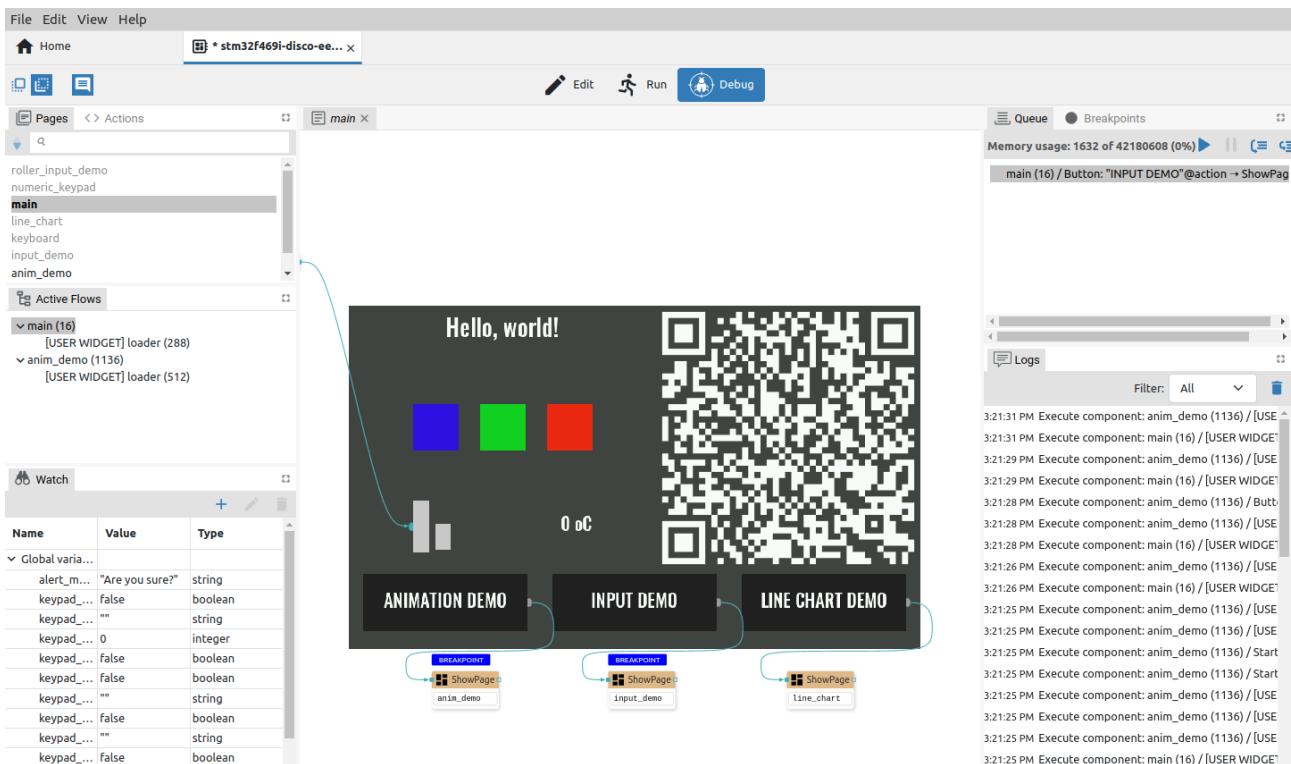


Fig. 101: Page view in Debug mode

12.2.2. Action viewer

In the Action viewer, Actions are displayed without the possibility of editing. It is also possible to see which Action components are currently being executed. If the Flow is paused, you can add breakpoints and see what values the component has on the inputs.

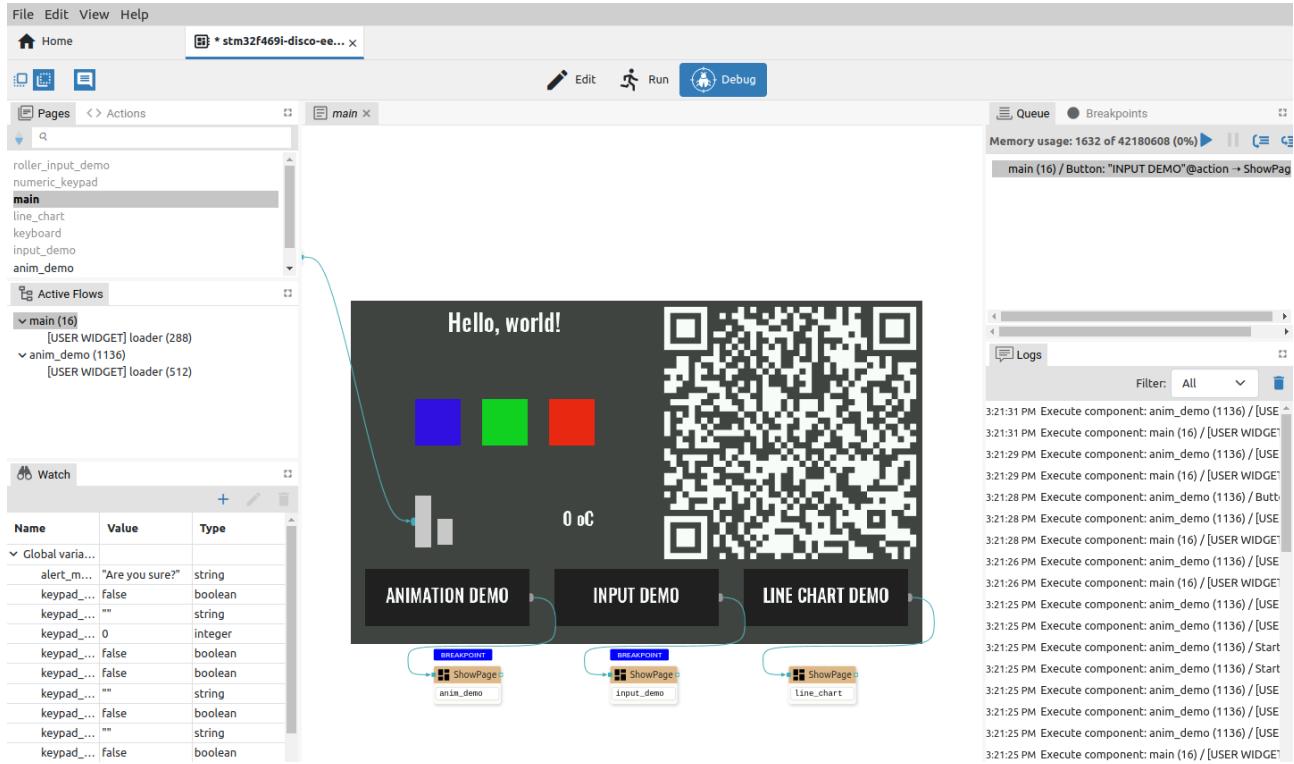


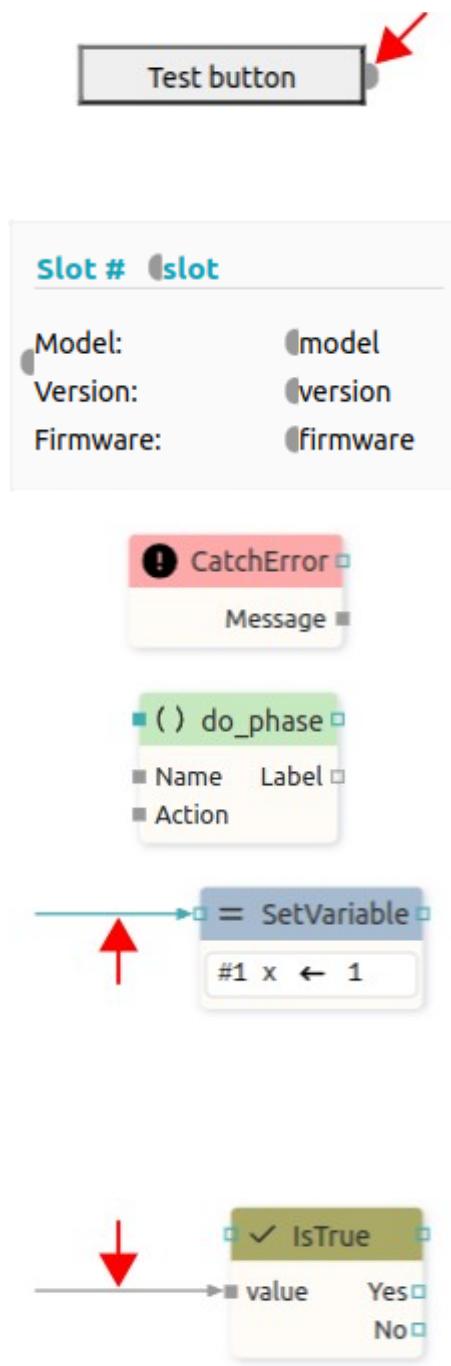
Fig. 102: Action viewer

13. EEZ Flow

13.1. EEZ Flow basic concepts

EEZ Flow is used to add programming logic to a project which enables programming using Flow-charts.

Flow can be an integral part of the page definition because the Widgets can be interactive and thus be an integral part of the Flow (connected to Actions and other Widgets by Flow lines). It is also possible to create a User Action for a Flow that does not have any graphic elements. The basic Flow elements are described below.



Widget

A component that adds a visible graphic element to a page. The Widget can be combined with other Widgets and Actions. For this, one or more inputs and outputs can be defined, which are displayed as semicircles on the left or right side (see arrow).

User Widget

User Widget is a convenient way to group part of a project that contains graphical elements for further reuse. *Input* and *Output* Actions are used to connect the User Widget with the rest of the Flow which are displayed as semicircles on the left or right side. A User Widget can be created from the *User Widgets* panel (*Add Item* option) or by selecting a part of the flow in the page editor and using the *Create User Widget* option in the right-click menu.

Action

A component that has no visible element on the page. An Action usually has at least one input and/or output to connect to other Widgets and Actions.

User Action

User Action is a convenient way to group part of a project for further reuse. User Action can use *Start*, *End*, *Input*, *Output* actions as connection points with the rest of the Flow.

Sequence Flow line

Sequence Flow line is used to define the execution Flow. The Action or Widget will be executed when it receives execution information on the sequence input (there is no data transfer, so it can be said that "null data" has been received). At the end of the execution of the Action or Widget, information ("null data") is sent to the next one or more Actions or Widgets through the sequence output. Sequence Flow line when not selected is shown in verdigris (greenish-blue) color.

Data Flow line

A data Flow line similar to a sequence Flow line can be used to define Flow execution. The data Flow line connects to the data input of the Action or Widget. Likewise, obtaining information after the execution of an Action on the data output is used to pass the execution information to the next one or more Actions or Widgets. In contrast to the sequence Flow line, the actual data is transferred along the data line: integer, string, structure, etc. When the data Flow line is not selected, it is displayed in gray color.

13.2. Flow execution

EEZ Studio allows multiple Flows to be executed in parallel within the same project. Project execution monitoring is possible in Debug mode (Fig. 103).

During execution, the current value of all global variables and the list of active Flows is preserved.

At some point there can be one or more active Flows. Each active Flow stores the current values of all its local variables, the values of all inputs on all components and the internal state of all components belonging to that Flow (namely, some components have internal state, for example, the *Loop* component remembers how many times it has looped).

The execution queue contains a list of all components that are ready for execution. All active Flows share the same execution queue.

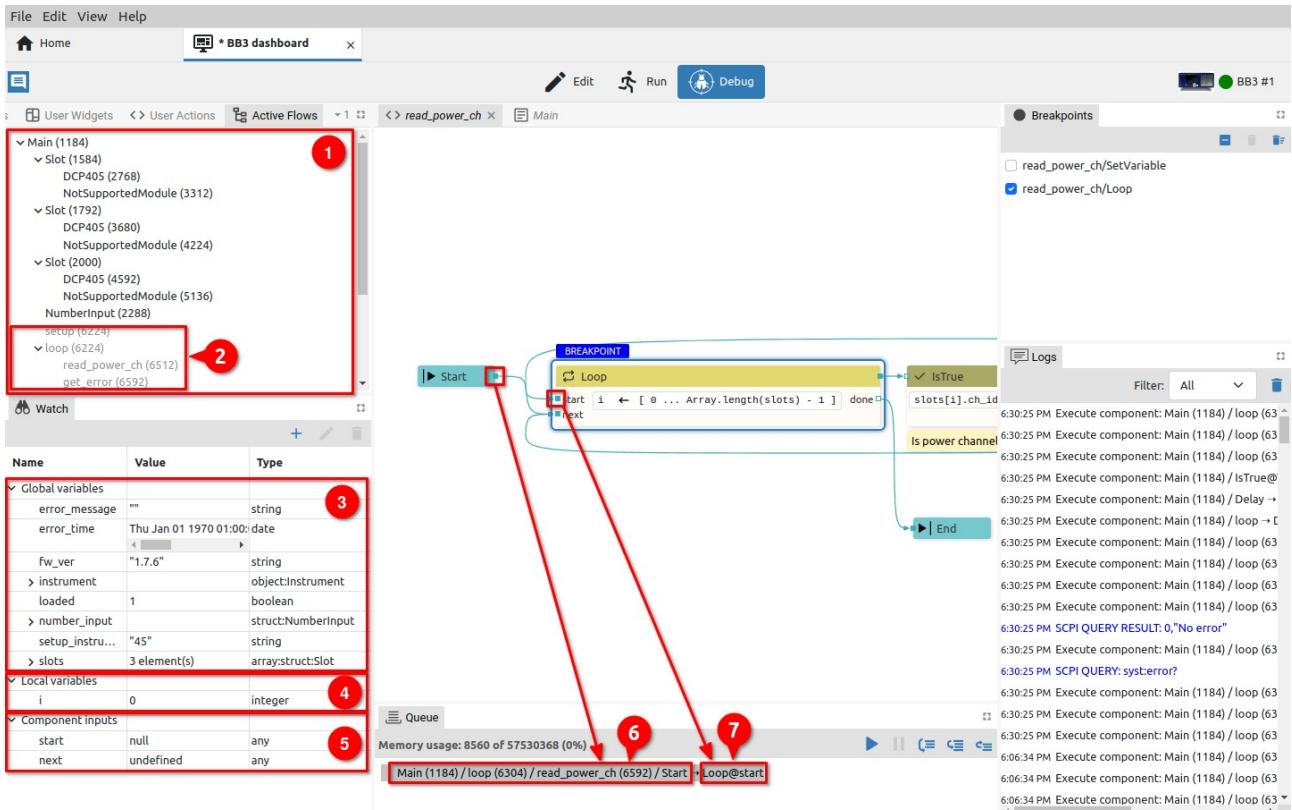


Fig. 103: Flow execution monitoring

When the project is executed in *Debug* mode, we can monitor in the *Active Flows* panel which Flows are active (1) and, as well as those that have finished execution, are no longer active, i.e. they are no longer in the execution queue (2). In the example in Fig. 103 we see that the *Main* page has one active Flow and under it are all the active Flows for the Widgets located in the *Main* page: we have three *Slot* Widgets, and each of them has its own two active Flows under it (*DCP405* and *NotSupportedModule*).

The *Watch* panel allows us to monitor the state of global variables (3). There we also find a list of local variables of the currently active Flow (4) as well as the input state of the component that will be executed next (5).

Numbers in parentheses next to the Flow name are memory addresses where the state of an individual active Flow is stored (e.g. 1184 for *Main* Flow).

One component at a time is taken from the beginning of the execution queue and executed.

During the execution of a component, data can be sent to one of the outputs, which will then be forwarded via Flow lines to the inputs of other components.

In the *Queue* panel, we can see the current activity, for example, that a value was sent from the output of the *Start* component to the *Start* input of the *Loop* component.

At the moment when the component receives data via the Flow line on one of the inputs, it will be placed at the end of the execution queue (i.e. it is ready for execution when it comes to its turn) if by then it has received data on all data inputs and on at least one sequence input (if such exists). If there is no Flow line that ends in an input, then that input is not looked at in this test.

Why only one sequence input? For example, the *Loop* component has two sequence inputs, *Start* and *Next*, and it is enough for one of them to receive data to become ready for execution (once on *Start* and later multiple times on *Next*).

When executing the component, all sequence inputs are deleted (data value is cleared), and data inputs keep the current data (last data value is kept). Which means that if new data comes later on only one sequence input, the component will be executed again because it already has data on all data inputs since they have been saved. Exceptions are possible here when a component can delete the value on one of its data inputs by itself. For such components, it will be specifically stated in its description.

If the component has no inputs (or if there is no Flow line that ends in one of the component's inputs), then it is immediately placed in the execution queue during initialization (i.e. when the Flow is started). For example *Start* is such a component and it is always executed immediately.

The *Catch* error component, although it has no inputs, will not be executed immediately, but only when an error occurs in the Flow in which it is located.

The *OnEvent* component also has no inputs and will not be executed immediately, but only when a page event occurs (examples of page events: open page, close page).

Widgets are executed immediately. Namely, Widgets are also components that participate in the execution of the Flow: they can receive values on their inputs and can send values through their outputs.

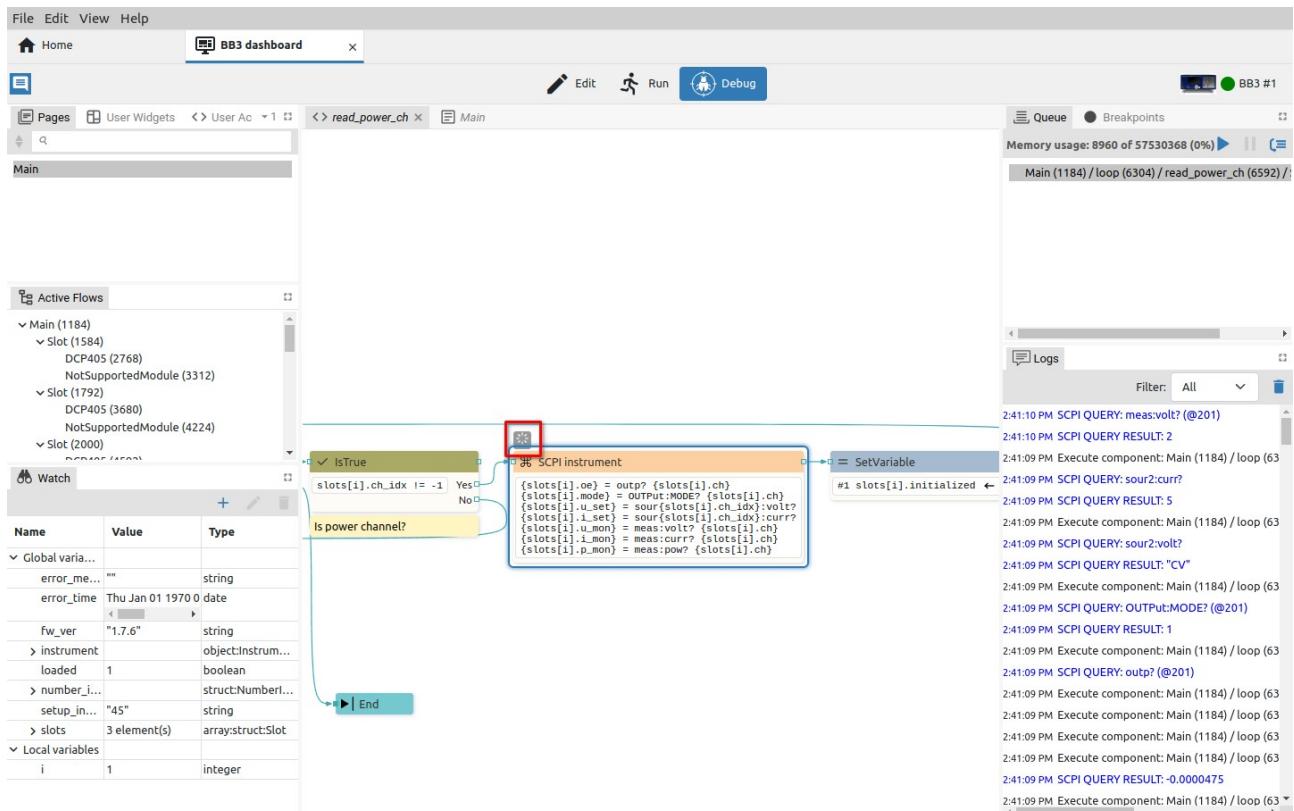


Fig. 104: Indication that the component preserves internal states

Components that preserve internal state, i.e. whose execution takes a long time, are also marked with a special icon in the debugger (Fig. 104). Example of such components: Loop, Delay, SCPI, etc. Such components, when they have done part of their work, can put themselves back in the queue. For

example The SCPI component executes the first command, is placed in the queue again, then executes the second command, is placed in the queue, and so on until the last command - while keeping the information in its internal state about which command it reached. In this way, parallel execution of all Flows was achieved, i.e. there is no waiting for the SCPI component to execute all its commands before some other component can be executed.

13.3. Flow examples

Flow execution will depend on the way components are connected. In Fig. 105 shows four simple Flows that contain User Actions whose inputs are triggered in different ways. Fig. 106 also shows the final execution results when the defined string will appear upon startup (4), with a defined delay (1) (3) or will not be displayed at all due to incorrect connection (2).

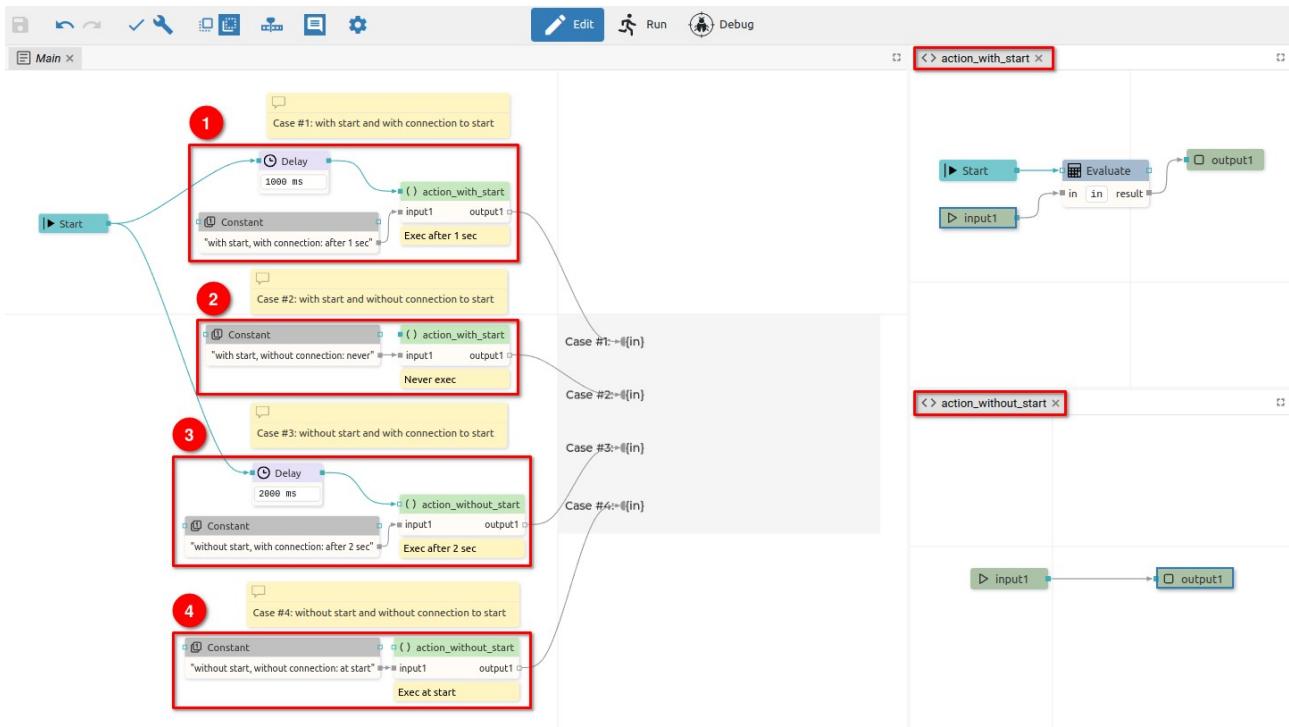


Fig. 105: Flow execution examples

Case #1 contains the User Action *action_with_start* that implements the *Start* action, making the sequence flow input mandatory. Flow will display the result after the 1 second *Delay* action is over. Case #3 will behave in the same way, where even though the sequence flow input is not mandatory (*action_without_start* is used).

In Case #4, it can be seen that if the sequence flow input is not mandatory, the User Action will be executed immediately at the start when the *Constant* will pass the string to be displayed.

In Case #2, we have a mandatory sequence input and nothing is connected to it. The Action will therefore not be executed and an error will be displayed in the editor.

Important: Although case #2 reports an error in the editor, it is allowed to run such a Flow. This is handy in case when not everything is connected, but we still want to test what has been done so far.

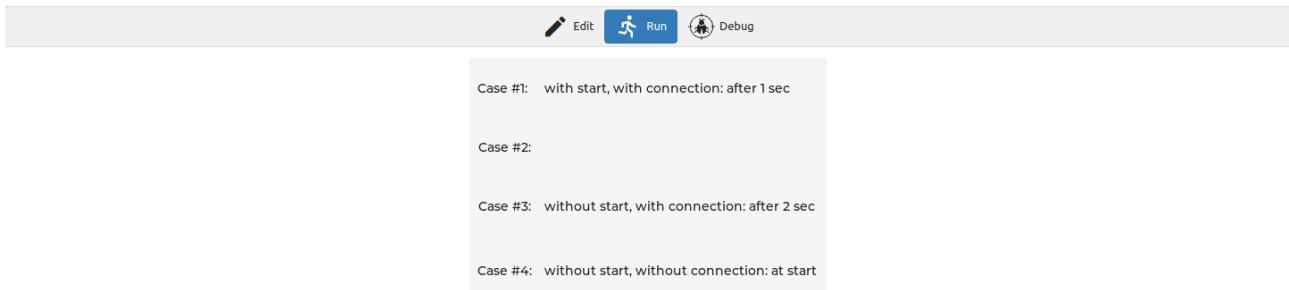


Fig. 106: Execution results

14. Project editing

Designing the graphical layout of the page is possible by simple drag & drop of one or more Widgets from the *Components palette*. The first step is to click on the Widget and hold which will change the cursor (Fig. 107)

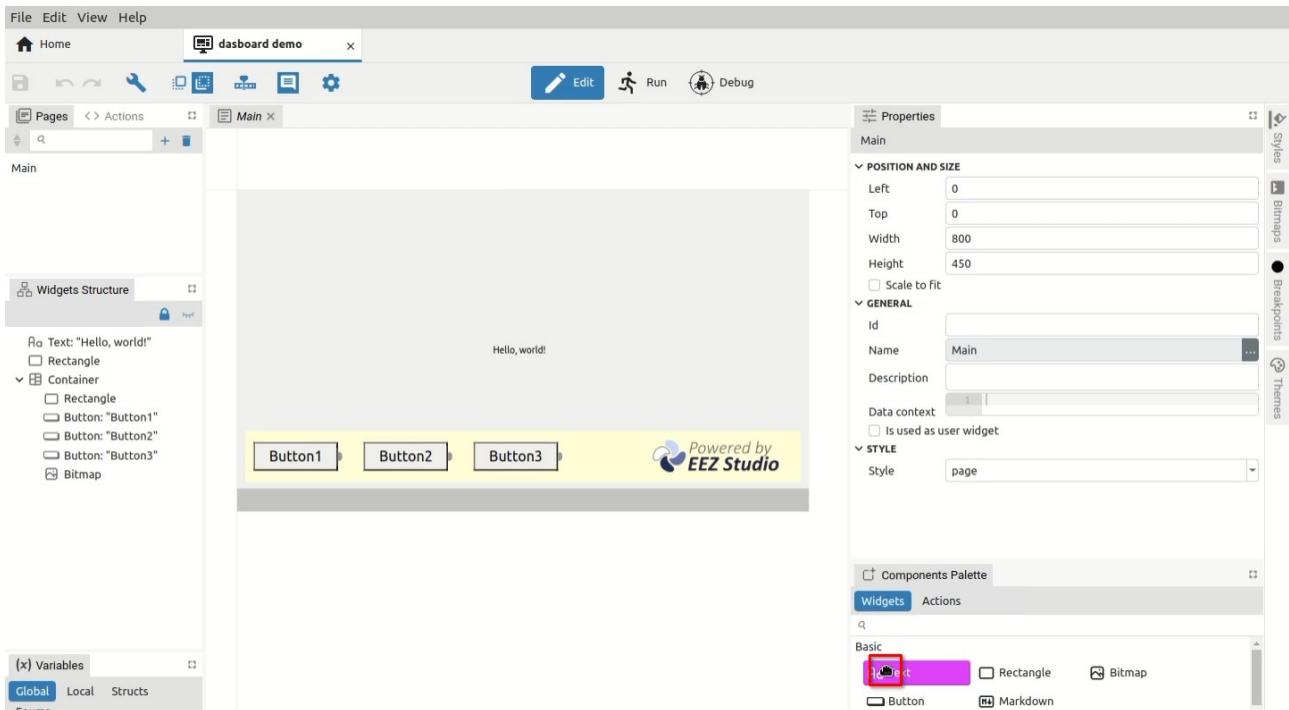


Fig. 107: Selection of Widget to add to the page

When the Widget is dragged into the editor area, auxiliary snap lines will appear immediately, which will make it easier to place the Widget in the desired place. Snap lines are displayed depending on nearby objects. If the position of the new Widget is not close enough to the others, or if it is the first Widget added to the page, the snap will be possible towards the page itself as in the examples in Fig. 108 where snap lines appear for horizontal or vertical centering.

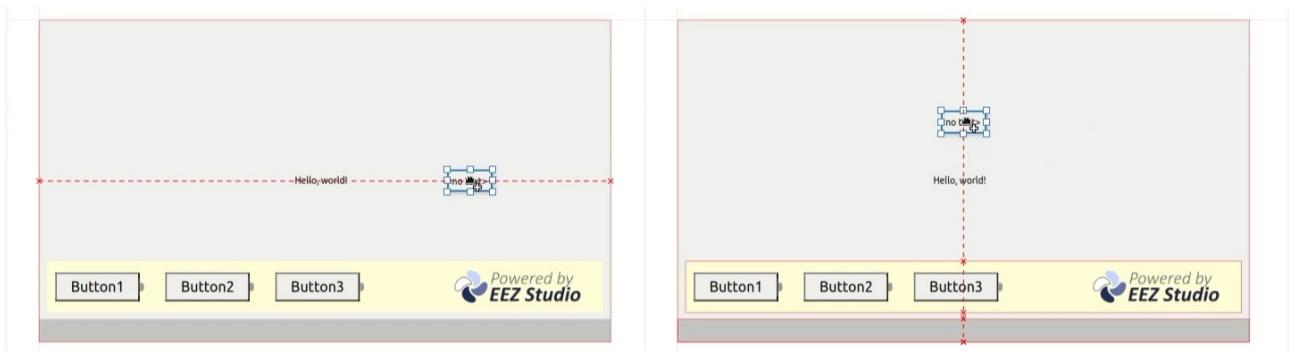


Fig. 108: Snap lines for centering in the page

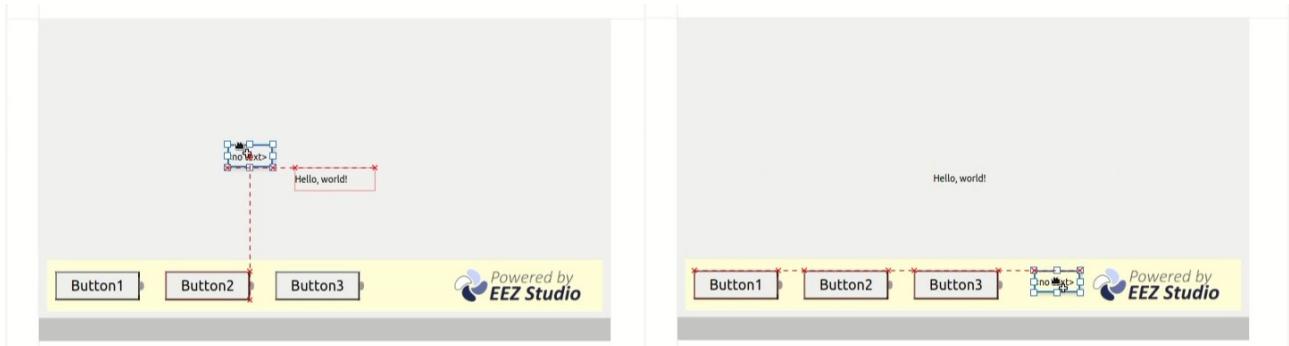


Fig. 109: Snap lines for positioning versus other Widgets

Fig. 109 shows examples of snap lines to the edges of other Widgets on the page.

In the event that snap lines become a nuisance during positioning for any reason, they can be disabled by holding down the SHIFT key while moving.

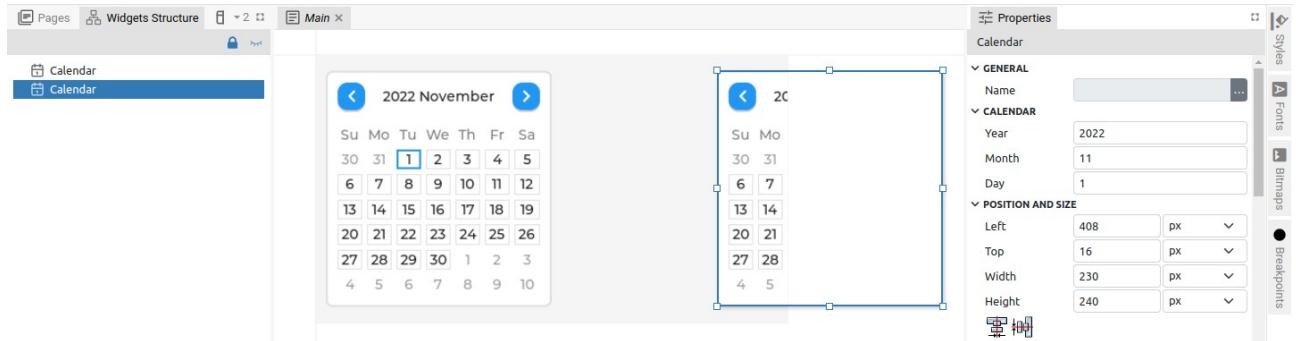


Fig. 110: Widget positioned outside the page

Please note that if the Widget in the *EEZ-GUI* and *LVGL* projects is set to protrude from the page (Fig. 110), the part that is outside the page will not be visible.

Widget positions can be freely changed and this can be done for one or more selected Widgets. It is possible to select multiple Widgets (Fig. 111): both in the page editor (1) or in the *Page Structure* panel (2). In both cases, information will appear in the *Properties* panel that multiple Widgets are selected (3). When selecting in the *Page Structure* panel, it is possible to use the SHIFT key to select a continuous sequence or CTRL to add individual Widgets to the selection.

There are two methods of multiple selection in the editor: selecting Widget by Widget while holding down the SHIFT key, and the second method is the so-called rubber band selection shown in Fig. 112 when selecting the area that will include the Widgets we want to select.

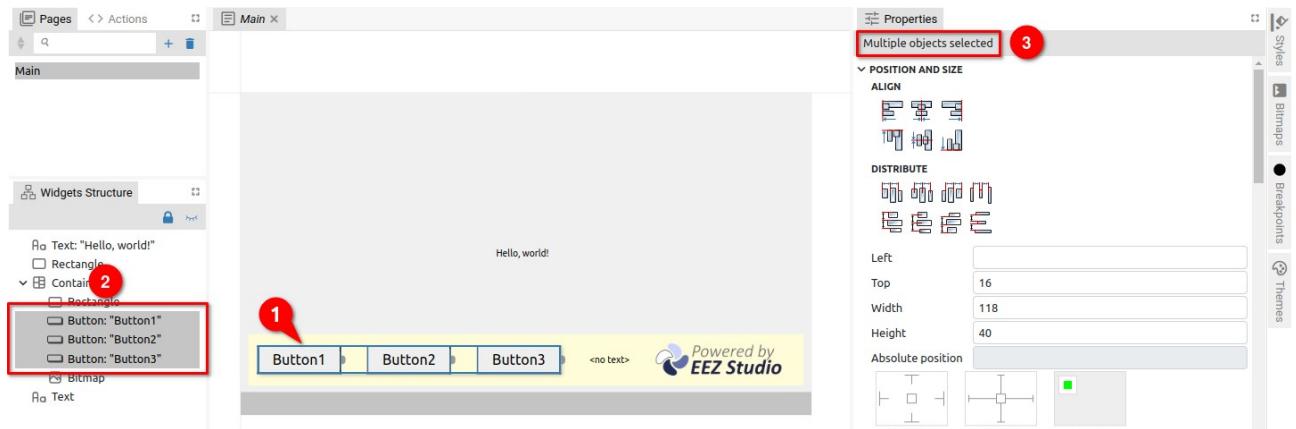


Fig. 111: Multiple Widgets selection

First you need to click on a place outside the Widget, then drag the mouse and release the button (a rectangle is displayed and when the mouse is released all Widgets inside will become selected).



Fig. 112: "Rubber band" selection

As shown in Fig. 111, it is possible to change *Properties* for multiple Widgets. Specific to multiple selection is the *Position and size* section when the *Align* subsection gets more options, and a complete *Distribute* subsection appears.

The *Distribute* subsection will be enabled only if three or more Widgets are selected.

Align

Icon	Title	Description
	Align left edges	Align to the left edge of the leftmost Widget.
	Center on vertical axis	Vertical centering towards the center of the widest Widget.
	Align right edges	Align to the right edge of the rightmost Widget.
	Align top edges	Align to the top edge of the uppermost Widget.
	Center on horizontal axis	Horizontal centering towards the center of the tallest Widget.
	Align bottom edges	Align to the bottom edge of the lowest positioned Widget.

Distribute

Icon	Title	Description
	Distribute left edges equidistantly	Distribution of all Widgets between leftmost and rightmost Widgets for equal distance between left edges.
	Distribute centers equidistantly horizontally	Distribution of all Widgets between leftmost and rightmost Widgets for equal distance between centers.
	Distribute right edges equidistantly	Distribution of all Widgets between leftmost and rightmost Widgets for equal distance between right edges.
	Make horizontal gaps equal	Distribution of all Widgets between leftmost and rightmost Widgets for an equal gap between them.
	Distribute top edges equidistantly	Distribution of all Widgets between the uppermost and the lowest positioned Widget for equal distance between top edges.
	Distribute centers equidistantly vertically	Distribution of all Widgets between the uppermost and the lowest positioned Widget for equal distance between centers.
	Distribute bottom edges equidistantly	Distribution of all Widgets between the uppermost and the lowest positioned Widget for equal distance between bottom edges.
	Make vertical gaps equal	Distribution of all Widgets between the uppermost and the lowest positioned Widget for an equal gap between them.

The page in the editor can be resized or set to the default size (1:1) or scrolled horizontally and vertically within the editor. For these operations, the mouse wheel is used in combination with the SHIFT and CTRL keys, as shown in Fig. 113.

Operation	Description
Page view resize	CTRL + mouse wheel is used to zoom the page.
Horizontal scroll	SHIFT + mouse wheel is used for horizontal page scrolling.
Vertical scroll	The mouse wheel is used for vertical page scrolling.
Move page	The page can be moved with the middle or right mouse button.
View reset	Double-click resets the zoom and centers the page.
Move Widget	Drag and drop is used to move selected Widgets within the page.

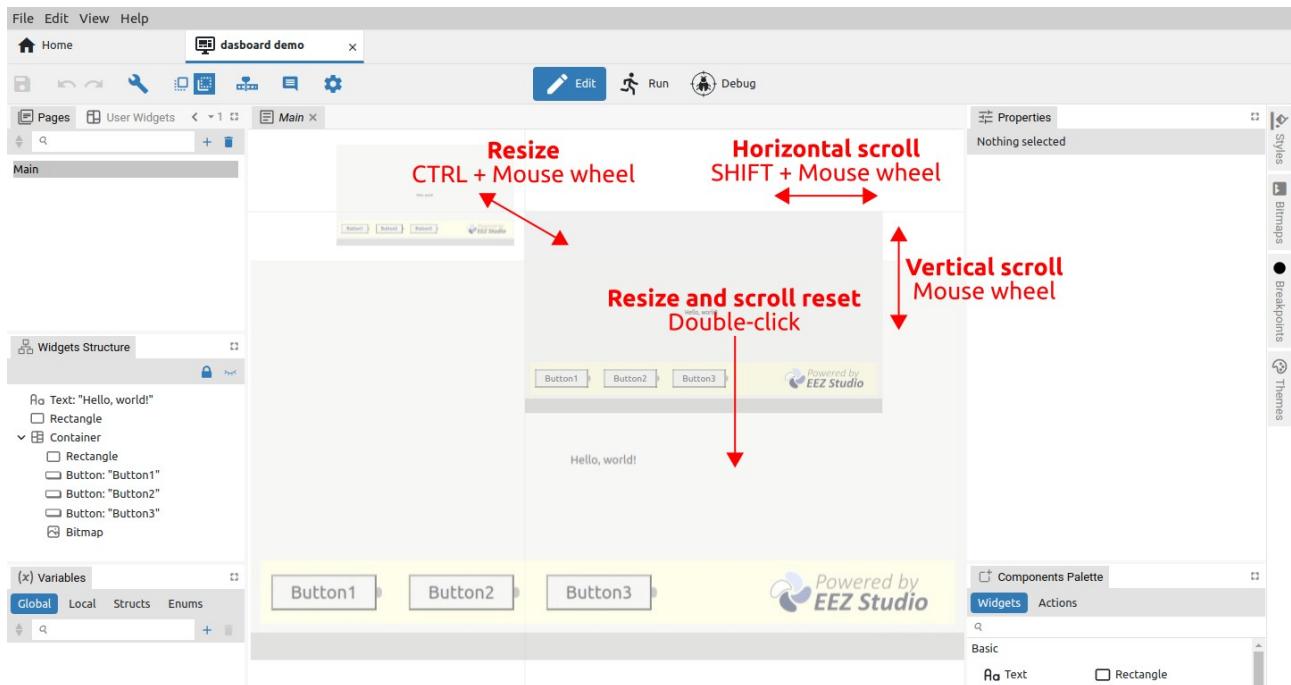


Fig. 113: Page resize and scroll

14.1.1. Connecting Flow components

The connection (Flow line) between Flow components (Widgets and Actions) is used to define the flow of execution. In Fig. 108 shows how the output of one Action is connected to the input of another Action. When we position ourselves on the output (1), the color of its background will change, and if we continue with the mouse drag, the Flow line (2) will appear. When the cursor reaches the input of the second component, the Flow line will change color to green (3). Now we can release the mouse when the connection between the two components will be established (4). In case of moving one of the components, the Flow line will move with it.

To delete the Flow line, it will be necessary to select the Flow line (the color will change to red) and select the Delete option in the right-click menu (or the DEL key).

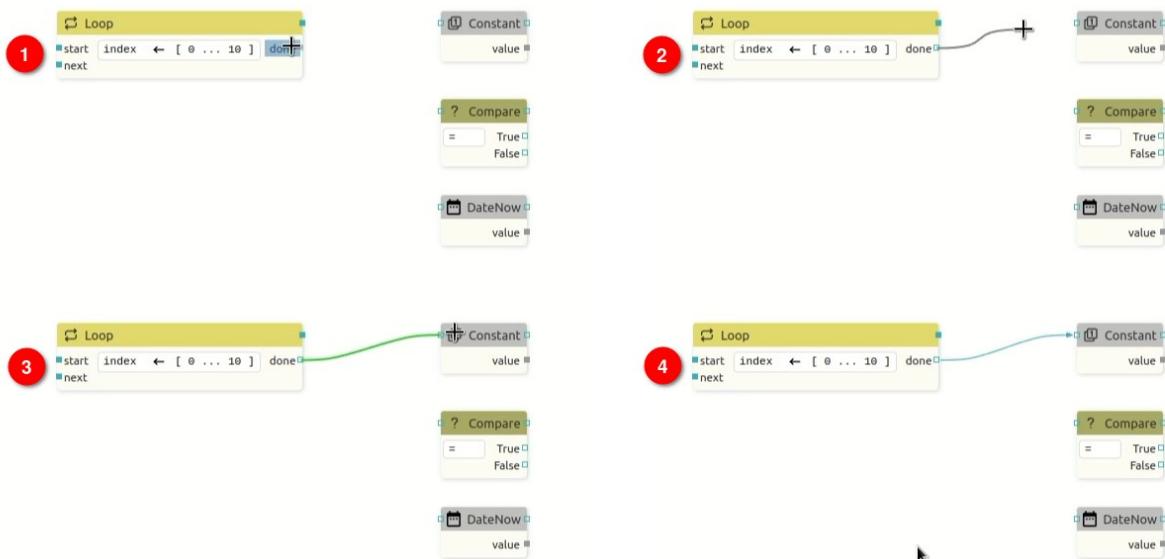


Fig. 114: Connecting the output to the input of the Widget

Adding a Flow line is also possible by starting from the input of one component to the output of another. In Fig. 115 shows how to connect the input of one Action to the output of another.

Note that it is possible to connect more than one Flow line to the single output, which also applies to the connection to the single input.

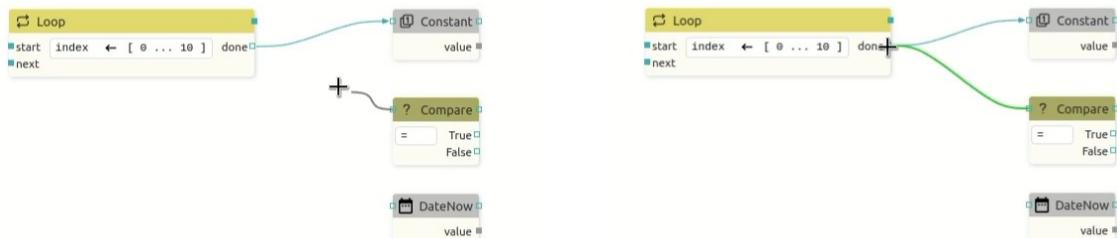


Fig. 115: Connecting the input to the output of the Widget

In case we have multiple Flow lines that end at a single input or output, it is possible to move them all to another input or output if necessary. Example in Fig. 116 shows the multiple Flow line moving from one output to another. First, we need to get to the output when the color of the background and all affected flow lines (1) will change. Then we need to drag the mouse while holding SHIFT, and a copy of the selected lines will appear, and their end can now be moved as desired (2). When we reach a new output, the color of the flow lines changes to green (3) and when the mouse button is released, a new connections will be drawn.

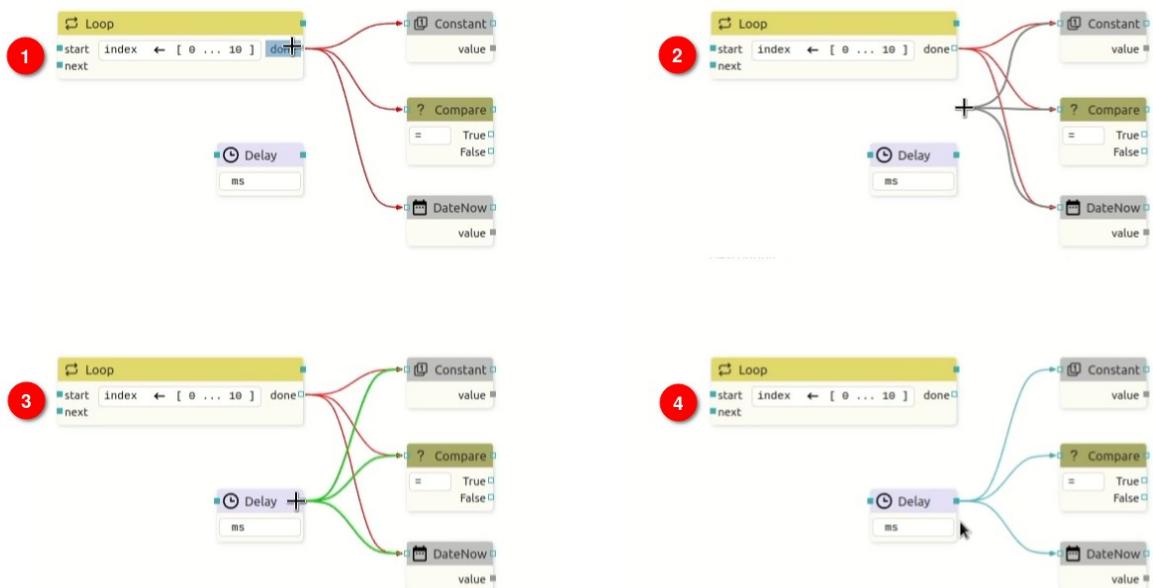


Fig. 116: Moving multiple connections

14.1.2. Copy & Paste between two projects

To copy between two projects, it will be necessary to open two EEZ Studio windows using the *New window* (CTRL + SHIFT + N) option from the *File* menu. In the first project, select the section you want to copy and select the *Copy* option from the right-click menu (or CTRL + C) to copy to the clipboard. From the clipboard, the selected section can now be inserted into another project using the right-click menu *Paste* option (or CTRL + V).

14.2. Working with Widgets

Widget components allow us to quickly add graphics to the page because each one implements a specific element (e.g. button, text, bitmap, QR code, etc.) that can be easily customized as needed. Widgets are located in the *Widgets* subtab of the *Components Palette*, where they are grouped for easier finding.

EEZ Studio supports two types of Widgets that cannot be mixed with each other:

- *EEZ-GUI (Native)* – Widgets created for the purposes of creating the EEZ BB3 embedded GUI for the STM32 family of MCUs that support graphics (Fig. 117)
- *LVGL* – Widgets from the open-source embedded graphics library LVGL. They can only be used in a project of type LVGL. (Fig. 118)

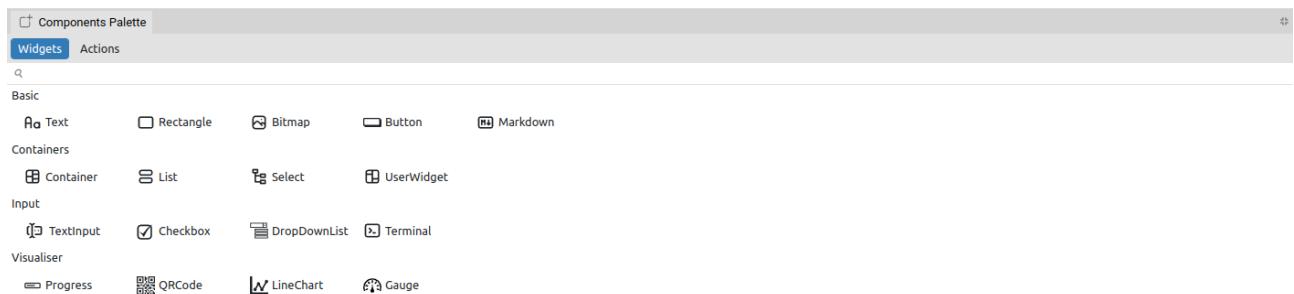


Fig. 117: Widgets palette for the EEZ-GUI project

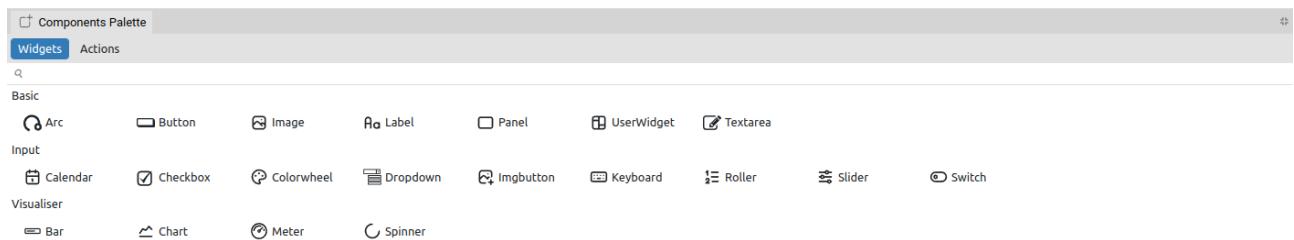


Fig. 118: Widgets palette for the LVGL project

14.2.1. Widget component's items

Widget component items are shown in Fig. 119 and described below.

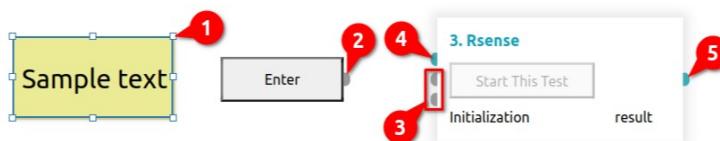


Fig. 119: Widget components items

#	Item	Description
1	<i>Selection handlers</i>	They appear when the Widget is selected and allow it to be resized in all directions.
2	<i>Sequence Output</i>	The mandatory sequence output must be connected, otherwise it will generate an error as the Widget will not be able to perform correctly.
3	<i>Sequence Input</i>	The mandatory sequence input must be connected, otherwise it will generate an error since the Widget will not be able to perform correctly.
4	<i>Data Input</i>	The mandatory data input must be connected, otherwise it will generate an error as the Widget will not be able to perform correctly.

5 Data Output

The mandatory data output must be connected, otherwise it will generate an error as the Widget will not be able to perform correctly.

Table 2 shows all types of I/O pins used as Flow line connection points for both Widgets and User Widgets.

Pin	Description
	Sequence input pin (Flow line connection point).
	Sequence output pin.
	Data input pin.
	Data output pin.

Table 2: User Widget's pin types

14.2.2. Creating a User Widget

The use of User Widgets contributes to modularity, which simplifies maintenance if the same layout elements appear in multiple places on the same or multiple pages. Each change will not need to be implemented in several places, but only in the User Widget.

A project can have an arbitrary number of User Widgets. User Widgets are displayed in the User Widgets panel, where new ones can be added and existing ones can be deleted.

A new User Widget can be created in two ways: using the *User Widgets* panel or the Right-click menu.

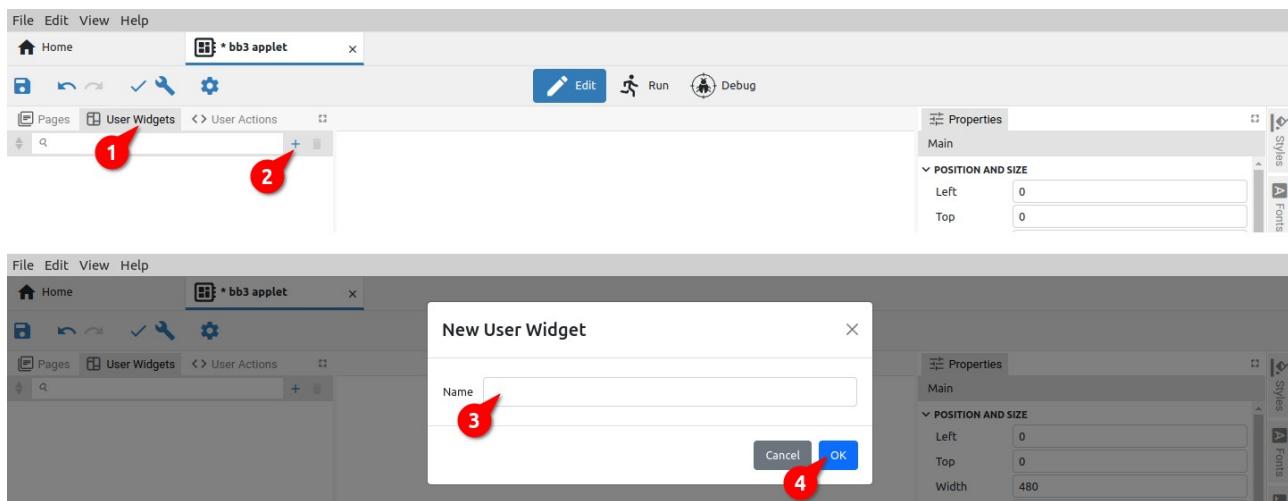


Fig. 120: Creating a new User Widget from the User Widgets panel

In the first case, select the *User Widgets* panel (1) and then the Add option (2), when a dialog box for entering the name of the User Widget will appear (Fig. 120).

After confirmation (4), the newly added User Widget will appear in the *User Widgets* list, where when selected, the editor opens where you can continue editing by adding Widgets and Actions. A User Widget can also contain multiple User Widgets and User Actions.

User Widget added in this way will by default contain a page with dimensions defined in the general Settings of the project. In the example in Fig. that's 480 x 272 pixels. It will also inherit the default style (hence the background is dark blue). The page will be positioned at the starting point (x = 0, y = 0).

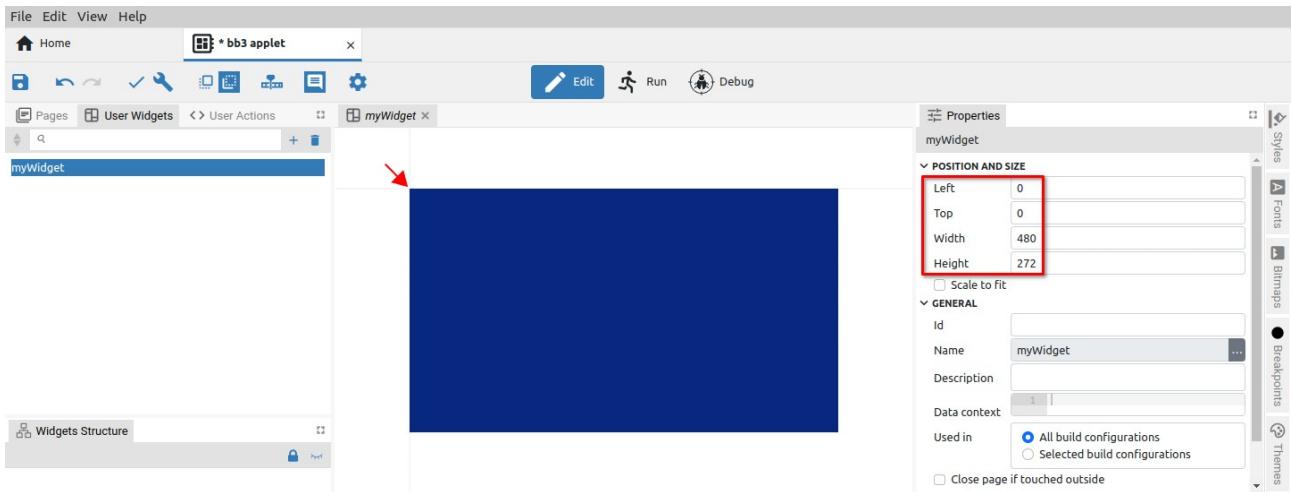


Fig. 121: Page editor of the newly created User Widget

User Widget can also be created by selecting one or more components on the currently displayed page (1) as shown on Fig. 122. By selecting the right-click menu option *Create User Widget*, a dialog box will appear as in the previous case (Fig. 120).

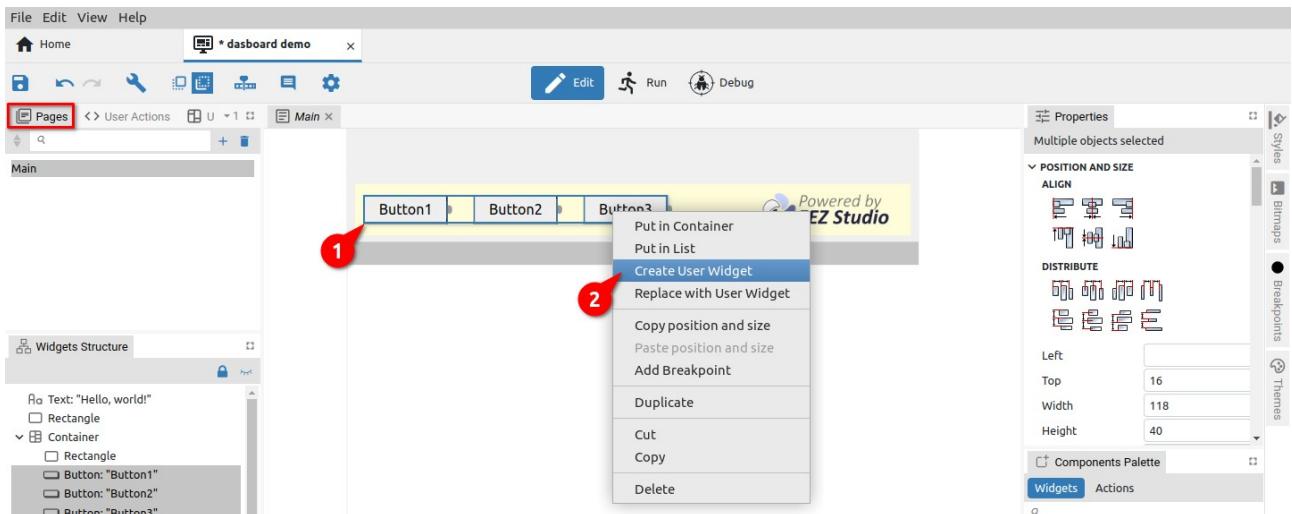


Fig. 122: Creating a new User Widget using the right-menu option

After successfully adding a new User Widget, it can be edited in the page editor (Fig. 123). Unlike the previous case, this Widget will have the dimensions of the original selection and the first component will be positioned at the starting point.

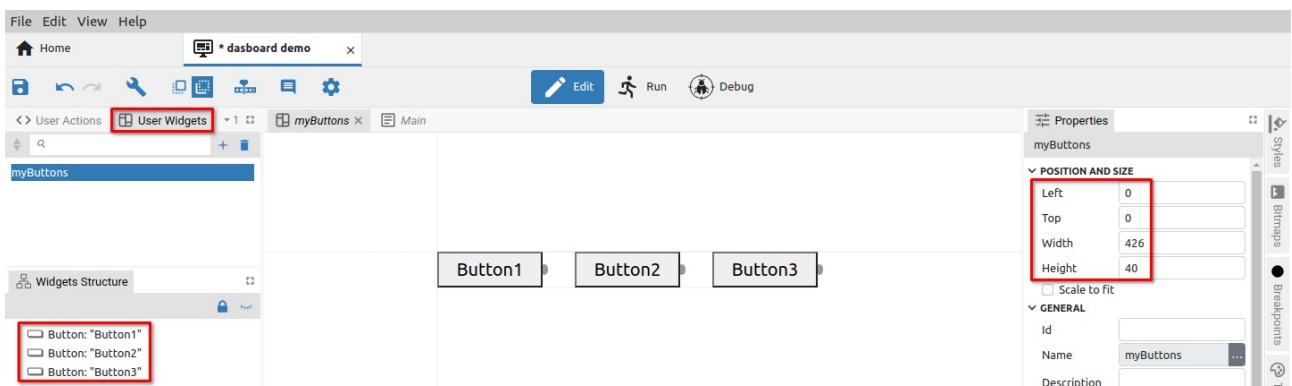


Fig. 123: The newly created User Widget in the editor

14.3. Working with Actions

Action unlike Widget does not have a graphical representation. It only performs some function/operation when executed.

The Actions that come with EEZ Studio (i.e. built-in Actions) are located in the *Actions* subtab of *Components Palette* and are added to the editor with drag & drop and grouped for easy finding. The number of implemented Actions depends on the type of project. In Fig. 124 shows the Actions for the Dashboard project, and Fig. 125 for the LVGL project.

The Action can also be implemented in the EEZ Studio extension. An example of such an Action is *Postgres*, which is shown in the *eez-Flow-ext-postgres* group (Fig. 124).

EEZ Studio also allows defining User Actions. To edit them, we use the User Actions editor. All User Actions are also listed at the bottom of the Actions subtab (Fig. 125), from where they can be added to the project with drag and drop as any Action or Widget.

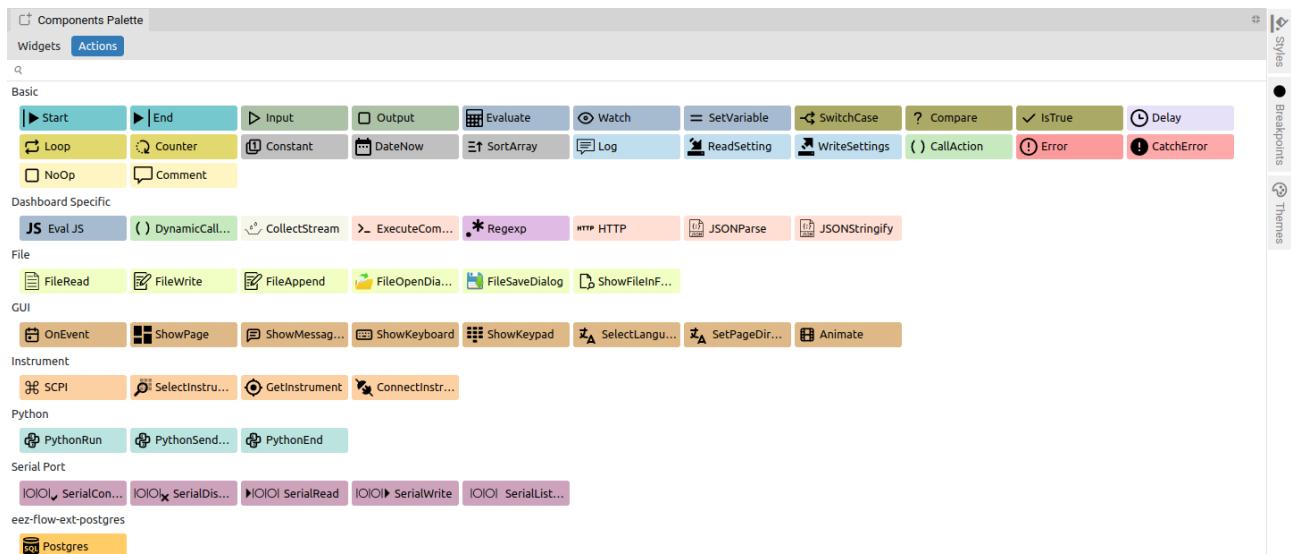


Fig. 124: Actions palette for the Dashboard project

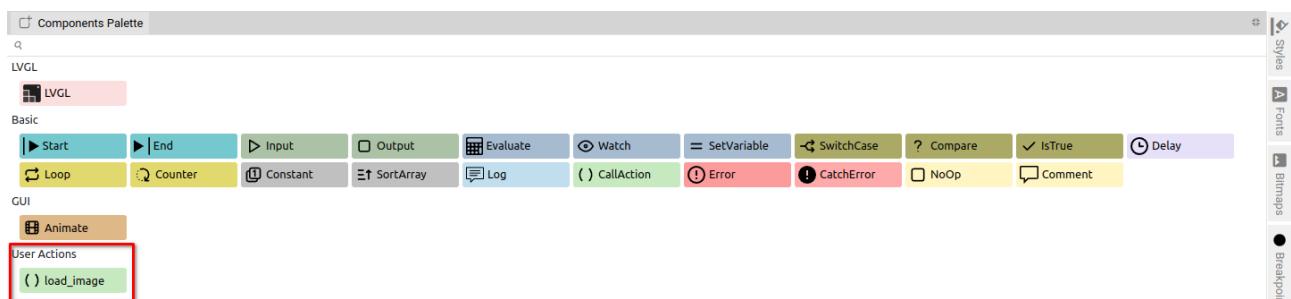


Fig. 125: Actions palette for the LVGL project

14.3.1. Action component's items

Action component items are shown in Fig. 126 and described below.

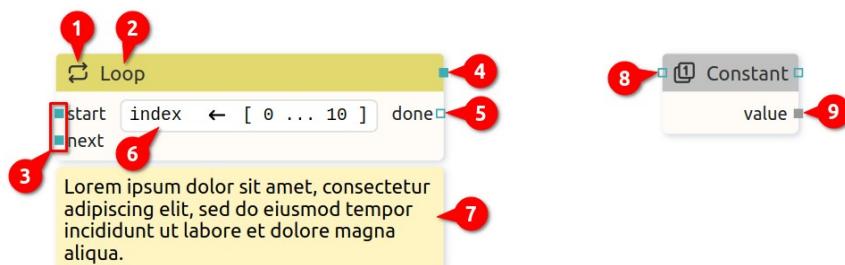


Fig. 126: Action components items

#	Item	Description
1	<i>Icon</i>	Component icon (cannot be changed).
2	<i>Name</i>	Component name (cannot be changed).
3	<i>Mandatory sequence inputs</i>	The mandatory sequence input must be connected, otherwise it will generate an error since the component will not be able to perform correctly.
4	<i>Mandatory sequence output</i>	The mandatory sequence input must be connected, otherwise it will generate an error as the Action will not be able to perform correctly.
5	<i>Optional sequence output</i>	Sequence output that does not necessarily need to be connected for the Action to execute regularly.
6	<i>Additional information</i>	Optional display of additional Action component information.
7	<i>Description</i>	Component description as defined in Properties.
8	<i>Optional sequence input</i>	Sequence input that does not necessarily need to be connected for the Action to execute regularly.
9	<i>Mandatory data output</i>	The mandatory data input must be connected, otherwise it will generate an error as the Action will not be able to perform correctly.

Table 3 shows all types of I/O pins used as Flow line connection points for both Actions and User Actions.

Pin	Description
	Mandatory sequence input or output pin (Flow line connection point).
	Optional sequence input or output pin.
	Mandatory data input or output pin.
	Optional data input or output pin.

Table 3: Action's pin types

14.3.2. Creating a User Action

Using User Actions contributes to Flow's readability and modularity, which makes it easier to maintain if the same functionality appears in multiple places. Thus, each change will not need to be implemented in multiple places, but only in the User Action.

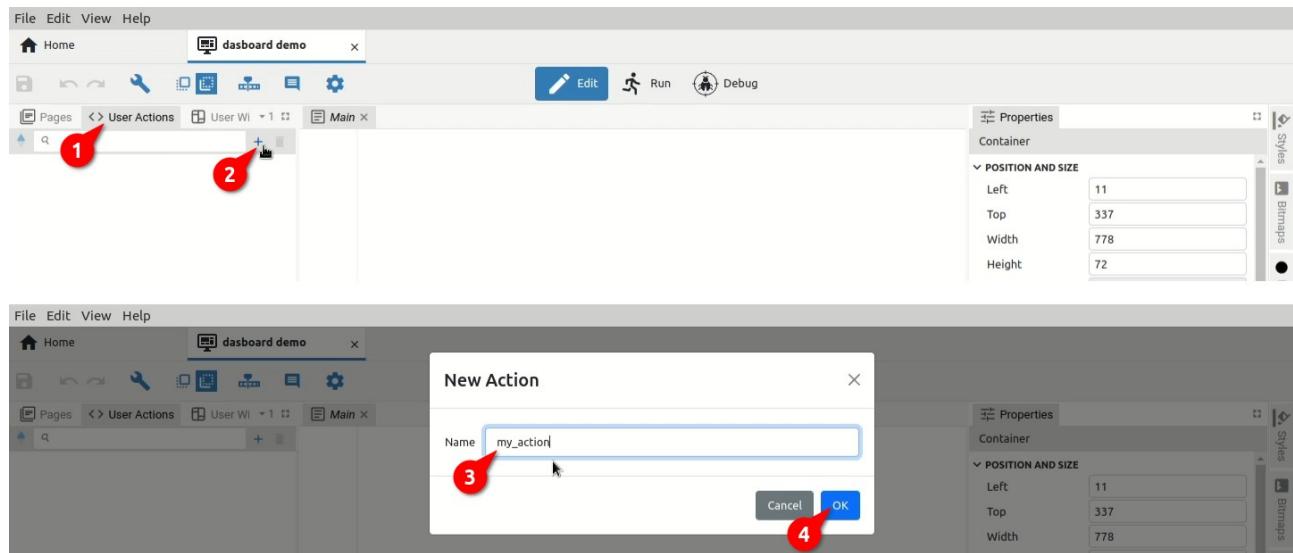


Fig. 127: Creating a new User Action

Please note that adding a User Action to itself is also allowed. However, care should be taken that it is connected in such a way that it does not create an infinite loop during execution.

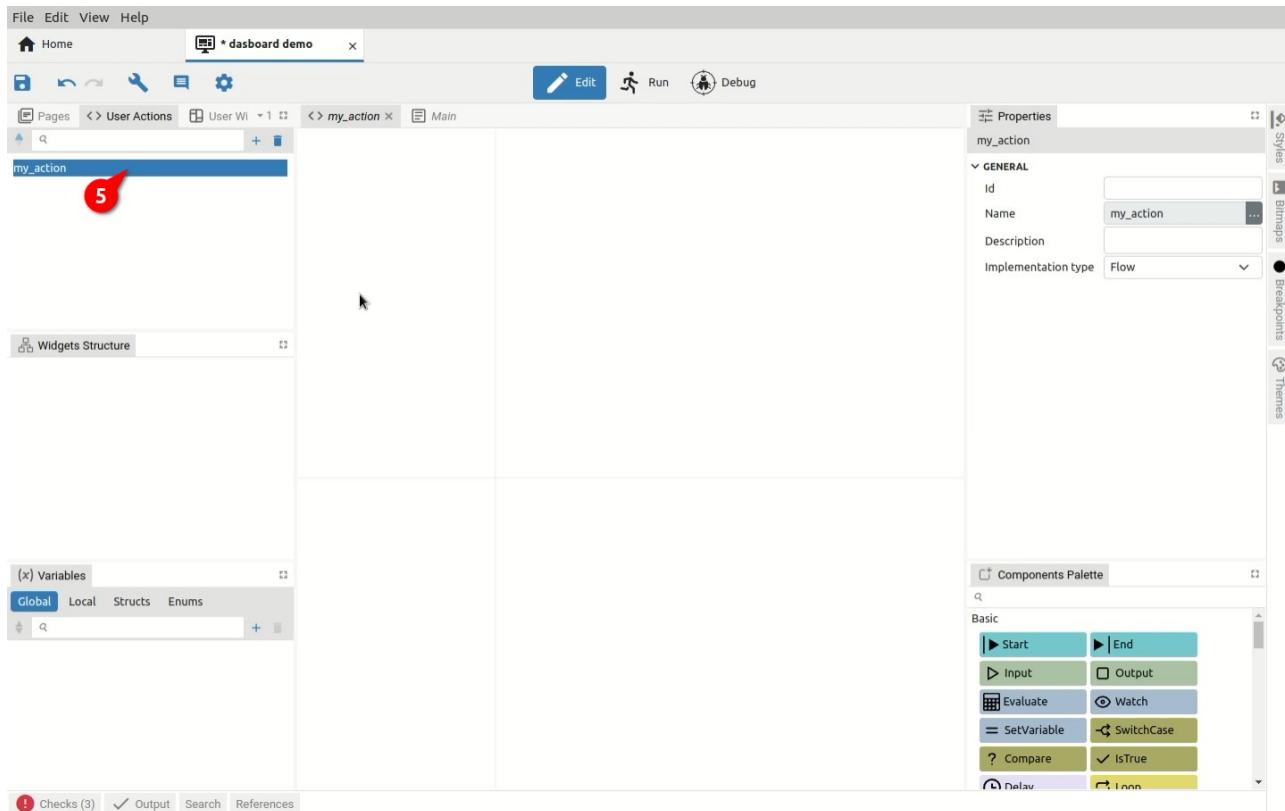


Fig. 128: Flow editor of the newly created User Action

Fig. 129 shows several examples of User Actions and how the use of sequence and data flow lines affects the appearance of the components that will be displayed in the Action palette.

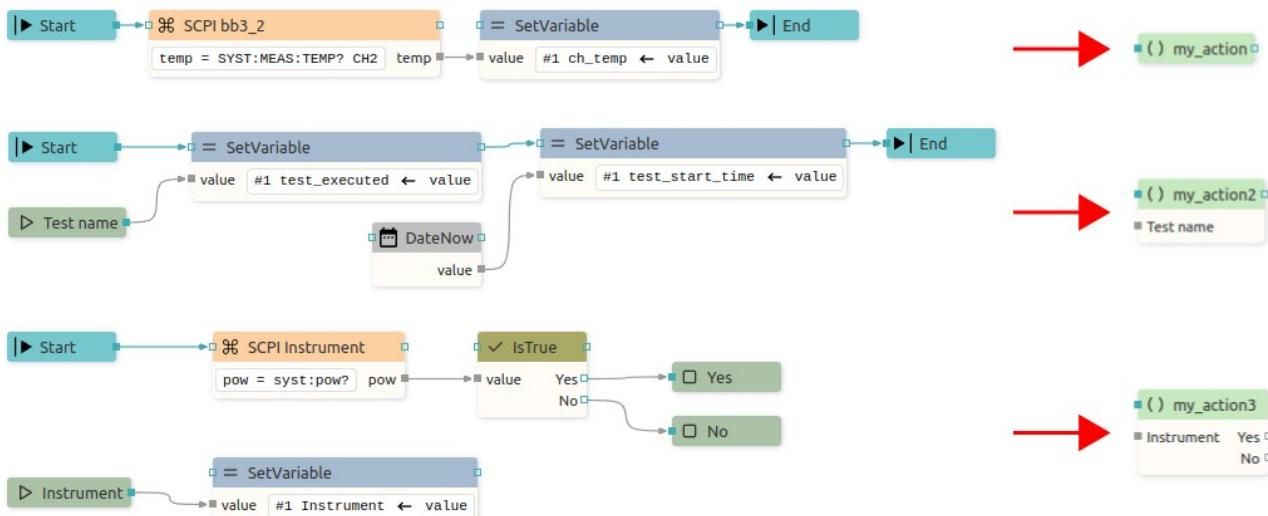


Fig. 129: User Action examples

15. Variables

A variable stores data that can be changed later on. Project with Flow support can have both global and local variables. Global variables are visible from all Flows, local variables are visible only inside the Flow in which it is defined.

Project without Flow support can have only global variables and those variables must be Native i.e. variables managed by the native code (written in C++).

To add variables, use the Variables panel (Fig. 130), when a dialog box will open for defining the basic parameters of the variable (*Name*, *Type* and *Default value*).

To edit the parameters of the variable selected in the *Variables* panel, use the *Properties* panel. In Fig. 131, Fig. 132 and Fig. 133 shows *Properties* panels for different types of variables from different types of projects.

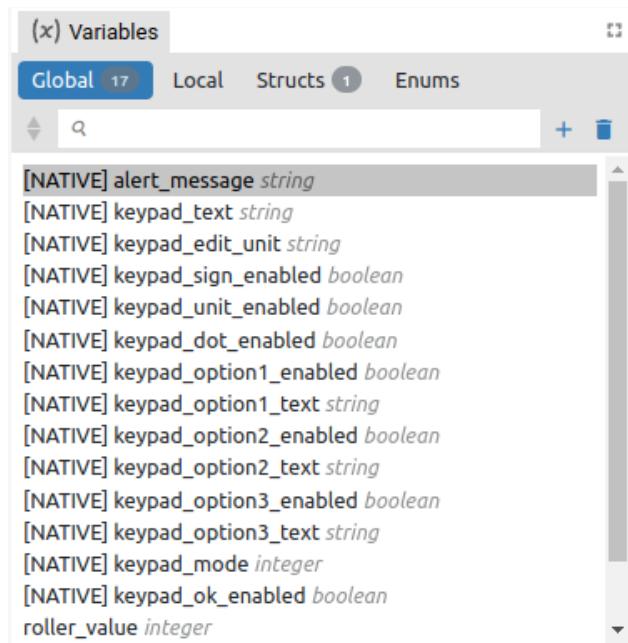


Fig. 130: Variables panel

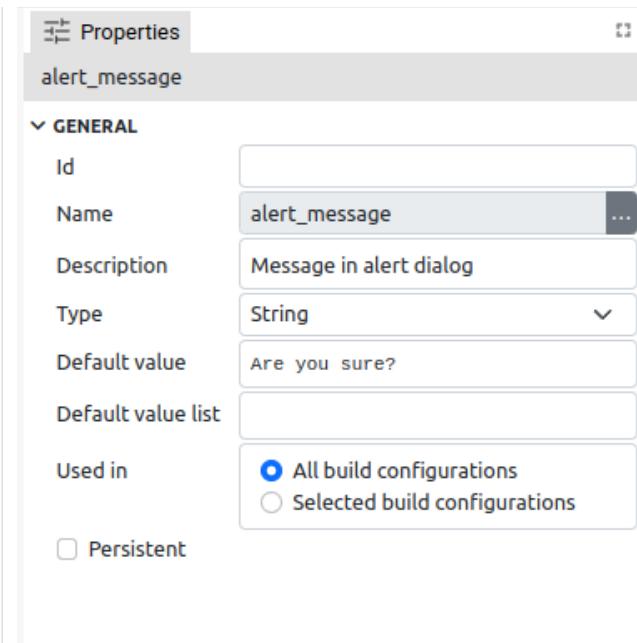


Fig. 131: Variable Properties panel (EEZ-GUI)

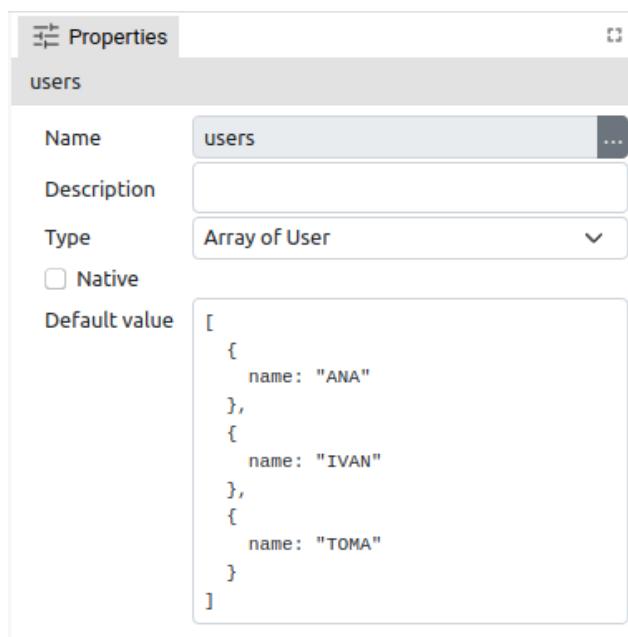


Fig. 132: Variable Properties panel (LVGL)

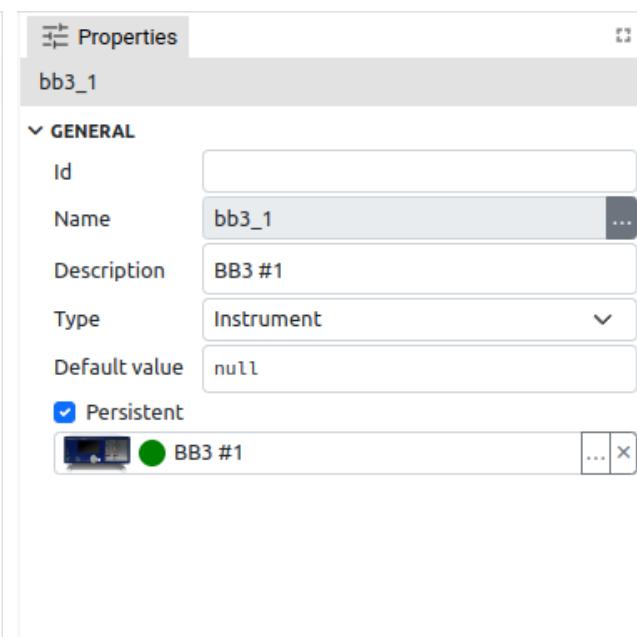


Fig. 133: Variable Properties panel (Dashboard)

Item	Description
<i>Id</i>	ID is <i>EEZ-GUI</i> project specific and is used in <i>Page</i> , <i>Action</i> , <i>Global Variable</i> , <i>Style</i> , <i>Font</i> , <i>Bitmap</i> and <i>Colors</i> . These are all resources that are referenced by name in the project editor. When that project is built, names are no longer used, but numerical IDs. This field is optional, i.e. an ID does not have to be specified, in which case an ID will be assigned during the build. However, if we want an object to always get the same ID, then it needs to be defined. Why would we want to always have the same ID? This is necessary when there is a master project such as <i>modular-psu-firmware.eez-project</i> from EEZ BB3 and that master project is used by BB3 Applets and BB3 MicroPython scripts and they can use resources from the master project that have that ID defined.
	<i>Important:</i> once the ID is set, it should not be changed, otherwise all BB3 scripts that depend on it should be rebuilt.
<i>Name</i>	Variable is referenced in other parts of project by its name. Rules for naming variables: Starts with a letter or an underscore (_), followed by zero or more letters, digits, or underscores. Spaces are not allowed.
<i>Description</i>	Optional field, contains a description of the variable.
<i>Type</i>	The type of data stored in variable. When adding a new variable, its suggested default value will depend on the selected type (0 for <i>Integer</i> , False for <i>Boolean</i> , etc.).
<i>Native</i>	The variable is managed by the native code (written in C++). A Dashboard project cannot have Native variables, and working with them is explained in Chapter XX.
<i>Default value</i>	Default value is the initial value of the variable when Flow starts. Given in JSON notation (https://www.json.org/json-en.html). For example: 123, "Hello", true If types is struct: { "member1": 42, "member2": "Hello" } If type is array: [1, 2, 3] or ["string1", "string2", "string3"]
<i>Default value list</i>	Only supported in <i>EEZ-GUI</i> project that does not have EEZ Flow enabled.
<i>Used in</i>	See <i>Configurations</i> in project <i>Settings</i> .
<i>Persistent (Global variables only)</i>	Stores the last value of the variable in the <i>.eez-runtime-settings</i> file, so that next time projects will have this value, instead of the default value. Supported only in Dashboard projects.

15.1. Variables usage

Variables are used in the project with EEZ Flow enabled. Data stored in a variable can be accessed using expressions. The following Action components are used to work with variables:

- *Evaluate* – evaluates expression, which can use variables, and sends the result through "result" data line.
- *Watch* – monitors the change in the value of the variable. At Flow start, it always sends the current value via the Changed data flow line, and later every value change is sent.
- *SetVariable* – sets a new variable value. Multiple entries are allowed. Each entry contains a variable and an expression field. During Flow execution, the evaluated expression will be stored in a variable.
- *SwitchCase*, *Compare*, *IsTrue* – Actions used for branching in the Flow depending on the value of the variable

Variables are also used in Widget components. Certain Widget properties can be defined as an expression. In this case, the value of that property will change during Flow execution as the expression changes. For example, *Label* widget can show the content of some variable, and it will updated every time this variable has been modified.

15.2. Variable types

15.2.1. Basic/Primitive types

Item	Description
<i>Integer</i>	Signed 32-bit integer.
<i>Float</i>	IEEE 4-byte floating-point.
<i>Double</i>	IEEE 8-byte floating-point.
<i>Boolean</i>	Can hold true or false value.
<i>String</i>	Sequence of characters.
<i>Date</i>	Unix timestamp.
<i>Blob</i>	Binary large object (Dashboard projects only).
<i>Stream</i>	Stream of data (Dashboard projects only).
<i>Any</i>	Can hold any data type.

15.2.2. Structures

Structure types are defined in *Variables* panel in *Structs* section. Struct type variable stores multiple data values each accessed by its member name. Each member is defined by its name and type.

Structures can only be used in projects that have EEZ Flow enabled.

15.2.3. Enums

Enums types are defined in *Variables* panel in *Enums* section. Enum type variable stores integer data value, but can contain only restricted set of values. Each enum member is defined by its name and integer value.

15.2.4. Objects

Object variables, similar to structs, can hold multiple values, each accessed by member names. The member names depends of the type of Object variable. Example of object variables: Instrument connection or PostgreSQL connection. Object variables are described in more detail in Chapter XX.

Object variables can only be used in Dashboard projects.

15.2.5. Arrays

Array variable stores multiple data values.

15.2.6. Expressions

An expression contains instructions on how to evaluate a data value during Flow execution. An expression is defined in code similar to JavaScript or other C-like languages.

Expression element Description / Example

<i>Literal value</i>	Example: 42, "Hello", true
<i>Variable names</i>	Example: my_var
<i>Input names</i>	Retrieves the data stored in data input using the name of that input. Example: input_name
<i>Binary operator</i>	Example: my_integer_var + 1
<i>Logical operator</i>	Example: my_integer_var < 10
<i>Unary operator</i>	Example: -my_integer_var

<i>Ternary operator</i>	Example: my_integer_var == 1 ? true : false (evaluates to True if my_integer_var is 1, otherwise evaluates to False)
<i>Function calls</i>	Example: String.length(my_string_var)
<i>Parentheses "()"</i>	Specifying the order of the evaluation Example: "Counter: " + (a + 1)
<i>Accessor "."</i>	Structure type member accessor by using ". Example: my_struct_var.member1
<i>Accessor "[]"</i>	Array element accessor by using "[]" Example: my_array_var[3], my_array_var[index]
<i>Enum value</i>	Example: MyEnumTypeName.Member1

Expression examples:

`var[i].member1`
`var` is array which contains structs, which has member `member1`
`i` is integer variable
evaluates to `member1` value in the i-th element

`var == State.START || var == State.EMPTY`
var is of type enum:State, and State enum has two members: START and EMPTY
evaluates to True if var contains data that is either State.START or State.EMPTY

15.2.7. Literals

Type	Description / Example
<i>Integer</i>	`42`
<i>Float or double</i>	`3.14`
<i>String</i>	`"Hello world!"`
<i>Translated string</i>	T"text_resource_id" (prefix T is mandatory)
<i>Boolean</i>	`true` or `false`
<i>JSON</i>	See https://www.json.org/json-en.html

15.2.8. Binary Operators

Each binary operators requires two arguments. Binary operator is written between arguments, for example: <arg1> + <arg2>

+ Addition

Rules:

- If any of the arguments is a string then result is a string. For example, `voltage +"V"` will evaluate to `"1.5 V"` if data stored in voltage variable is `1.5`
- If any of the arguments is a double then result is a double.
- If one argument is a float and the other is a float or an integer the result will be a float.
- If both arguments are integers then result is an integer.

arg1\arg2	integer	float	double	string	boolean	other_type
integer	integer	float	double	string	integer	err
double	double	double	double	string	double	err
float	float	float	double	string	float	err
string	string	string	string	string	string	err

<i>boolean</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>string</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>

Subtraction -

arg1\arg2	integer	float	double	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>err</i>
<i>float</i>	<i>float</i>	<i>float</i>	<i>double</i>	<i>float</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>

Multiplication *

arg1\arg2	integer	float	double	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>err</i>
<i>float</i>	<i>float</i>	<i>float</i>	<i>double</i>	<i>float</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>

Division /

arg1\arg2	integer	float	double	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>err</i>
<i>float</i>	<i>float</i>	<i>float</i>	<i>double</i>	<i>float</i>	<i>err</i>
<i>boolean</i>	<i>double</i>	<i>float</i>	<i>double</i>	<i>double</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>

Remainder %

arg1\arg2	integer	float	double	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>double</i>	<i>err</i>
<i>float</i>	<i>float</i>	<i>float</i>	<i>double</i>	<i>float</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>float</i>	<i>double</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>	<i>err</i>

Left shift <<

arg1\arg2	integer	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>

Right shift >>

arg1\arg2	integer	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>

Binary AND &

arg1\arg2	integer	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>

Binary OR |

arg1\arg2	integer	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>

Binary XOR ^

arg1\arg2	integer	boolean	other_type
<i>integer</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>boolean</i>	<i>integer</i>	<i>integer</i>	<i>err</i>
<i>other_type</i>	<i>err</i>	<i>err</i>	<i>err</i>

15.2.9. Logical operators

Logical operators are also binary operators that result in Boolean values.

Type Description

```
==  
!=  
<  
>  
<=  
>=  
&&  
||
```

15.2.10. Unary operators**Type Description**

```
+  
-  
~  
!
```

15.2.11. Conditional (ternary) operator

The conditional (ternary) operator is the only operator that takes three operands: a condition followed by a question mark (?), an expression to be executed if the condition is true followed by a colon (:), and finally an expression to be executed if the condition is false.

15.3. Functions

15.3.1. System

System.getTick

Description: Retrieves the number of milliseconds that have elapsed since the flow execution was started.
Parameters: None
Return value: Type: Integer
Description: Value in milliseconds.

15.3.2. Flow

Flow.index

Description: Index of current element in the List and Grid widget. Check the description of these two widget for the more information.
Parameters: Name: index
Type: Integer
Description: ???
Return value: Type: Integer
Description: Element index.

Flow.isPageActive

Description: If this function is executed inside the page it will return true if that page is currently active page, otherwise it will return false.
Parameters: None
Return value: Type: Boolean
Description: True if page is active, False if page is not active.

Flow.pageTimelinePosition

Description: If this function is executed inside the page or custom widget it will return the current position at the animation timeline for that page or custom widget.
Parameters: None
Return value: Type: Boolean
Description: Timeline position.

Flow.makeValue

Description: Creates a new value of type *Struct*.
Parameters: Name: structName Type: String Description: Structure name.
Name: value Type: JSON Description: Structure value.
Return value: Type: Struct
Description: Created struct value.

Flow.makeArrayValue

Description: Creates a new value of type array.
Parameters: Name: value
Type: JSON
Description: Array value.
Return value: Type: Array
Description: Created array value.

Flow.languages

Description: Retrieves a list of languages defined in multi-language project as array of strings.
Parameters: None
Return value: Type: Array:string

Description: Array of languages.

Flow.translate

Description: Translate text resource ID, same as T"textResourceID".
Parameters: Name: textResourceID
Type: String
Description: Text resource ID.
Return value: Type: String
Description: Translated string.

Flow.parseInteger

Description: Parse integer value given as string.
Parameters: Name: str
Type: String
Description: Input string.
Return value: Type: Integer
Description: Parsed integer value.

Flow.parseFloat

Description: Parse float value given as string.
Parameters: Name: str
Type: String
Description: Input string.
Return value: Type: Float
Description: Parsed float value.

Flow.parseDouble

Description: Parse double value give as string.
Parameters: Name: str
Type: String
Description: Input string.
Return value: Type: Double
Description: Parsed double value.

15.3.3. Date

Date.now

Description: Returns current date.
Parameters: None
Return value: Type: Now
Description: Current datetime.

Date.toString

Description: Converts given date to string.
Parameters: str date
Type: Date
Description: Input date.
Return value: Type: String
Description: Date string.

Date.toLocaleString

Description: Converts given date to locale string.
Parameters: str date
Type: Date
Description: Input date.
Return value: Type: String
Description: Date string.

Date.fromString

Description: Converts string to date.
Parameters: Name: dateStr

Type: String
 Description: Input string.

Return value: Type: Date
 Description: Date

Date.getYear

Description: Get year from date.
 Parameters: Name: date
 Type: Date
 Description: Input date.
 Return value: Type: Integer
 Description: Year

Date.getMonth

Description: Get month from date (1 to 12).
 Parameters: Name: date
 Type: Date
 Description: Input date.
 Return value: Type: Integer
 Description: Year

Date.getDay

Description: Get day of the month from date (1 to 31).
 Parameters: Name: date
 Type: Date
 Description: Input date.
 Return value: Type: Integer
 Description: Day

Date.getHours

Description: Get hours from date (0 to 23).
 Parameters: Name: date
 Type: Date
 Description: Input date.
 Return value: Type: Integer
 Description: Hours

Date.getMinutes

Description: Get minutes from date (0 to 59).
 Parameters: Name: date
 Type: Date
 Description: Input date.
 Return value: Type: Integer
 Description: Minutes

Date.getSeconds

Description: Get seconds from date (0 to 59).
 Parameters: Name: date
 Type: Date
 Description: Input date.
 Return value: Type: Integer
 Description: Seconds

Date.getMilliseconds

Description: Get milliseconds from date (0 to 999).
 Parameters: Name: date
 Type: date
 Description: Input date.
 Return value: Type: Integer
 Description: Milliseconds

Date.make

Description: Make a date from arguments.

Parameters:

Name: year	Type: Integer	Description: Year
Name: month	Type: Integer	Description: Month
Name: day	Type: Integer	Description: Day
Name: hours	Type: Integer	Description: Hours
Name: minutes	Type: Integer	Description: Minutes
Name: seconds	Type: Integer	Description: Seconds
Name: milliseconds	Type: Integer	Description: Milliseconds

Return value:

Type: Date	Description: Constructed date.
------------	--------------------------------

15.3.4. Math**Math.sin**

Description: Returns the sine of a number in radians.

Parameters:

Name: x	Type: Integer Float Double
---------	----------------------------

Description: A number representing an angle in radians.

Return value:

Type: Float Double	Description: The sine of x, between -1 and 1, inclusive.
--------------------	--

Math.cos

Description: Returns the cosine of a number in radians.

Parameters:

Name: x	Type: Integer Float Double
---------	----------------------------

Description: A number representing an angle in radians.

Return value:

Type: Float Double	Description: The cosine of x, between -1 and 1, inclusive.
--------------------	--

Math.pow

Description: Returns the value of a base raised to a power.

Parameters:

Name: base	Type: Integer Float Double	Description: The base number.
Name: exponent	Type: Integer Float Double	Description: The exponent number.

Return value:

Type: Float Double	Description: A number representing base taken to the power of exponent.
--------------------	---

Math.log

Description: Returns the natural logarithm (base e) of a number.

Parameters:

Name: x	Type: Integer Float Double
---------	----------------------------

Description: A number greater than or equal to 0.

Return value:

Type: Float Double	Description: The natural logarithm (base e) of x.
--------------------	---

Math.log10

Description: Returns the base 10 logarithm of a number.

Parameters:

Name: x	Type: Integer Float Double
---------	----------------------------

Description: A number greater than or equal to 0.

Return value:

Type: Float Double	Description: The base 10 logarithm of x.
--------------------	--

Math.abs

Description: Returns the absolute value of a number.

Parameters:

Name: x	Type: Integer Float Double
---------	----------------------------

Description: A number.

Return value:

Type: Integer Float Double	Description: The absolute value of x. If x is negative (including -0), returns -x. Otherwise, returns x. The result is therefore always a positive number or 0.
----------------------------	---

Math.floor

Description: Always rounds down and returns the largest integer less than or equal to a given number.

Parameters: Name: x
Type: Integer|Float|Double
Description: A number.

Return value: Type: Integer|Float|Double
Description: The largest integer smaller than or equal to x.
It's the same value as $-Math.ceil(-x)$.

Math.ceil

Description: Always rounds up and returns the smaller integer greater than or equal to a given number.

Parameters: Name: x
Type: Integer|Float|Double
Description: A number.

Return value: Type: Integer|Float|Double
Description: The smallest integer greater than or equal to x.
It's the same value as $-Math.floor(-x)$.

Math.round

Description: Returns the value of a number rounded to the nearest integer.

Parameters: Name: value
Type: Integer|Float|Double
Description: A number.

Return value: Type: Integer|Float|Double
Description: The value of x rounded to the nearest integer.

Math.min

Description: Returns the smallest of the numbers given as input parameters.

Parameters: Name: value1, ..., valueN
Type: Integer|Float|Double
Description: Zero or more numbers among which the lowest value will be selected and returned.

Return value: Type: Integer|Float|Double
Description: The smallest of the given numbers.

Math.max

Description: Returns the largest of the numbers given as input parameters.

Parameters: Name: value1, ..., valueN
Type: Integer|Float|Double
Description: Zero or more numbers among which the largest value will be selected and returned.

Return value: Type: Integer|Float|Double
Description: The largest of the given numbers.

15.3.5. String**String.length**

Description: Returns the length of the string.

Parameters: Name: string
Type: String
Description: A string.

Return value: Type: Integer
Description: The length of a string.

String.substring

Description: Returns the part of the string from the start index up to and excluding the end index, or to the end of the string if no end index is supplied.

Parameters: Name: string Type: String Description: A string.
 Name: start Type: String Description: The index of the first character to include in the returned substring.
 Name: end Optional: Yes Type: String Description: The index of the first character to exclude from the returned substring.

Return value: Type: String
 Description: A new string containing the specified part of the given string.

String.find

Description: Searches a string and returns the index of the first occurrence of the specified substring.

Parameters: Name: string Type: String Description: A string.
 Name: substring Type: String Description: Substring to search for.

Return value: Type: String
 Description: The index of the first occurrence of substring found, or -1 if not found.

String.padStart

Description: Pads the current string with another string (multiple times, if needed) until the resulting string reaches the given length.

Parameters: Name: string Type: String Description: A string.
 Name: targetLength Type: Integer Description: The length of the resulting string once the current str has been padded. If the value is less than or equal to str.length, then str is returned as-is.
 Name: padString Type: String Description: The string to pad the current str with. If padString is too long to stay within the targetLength, it will be truncated from the end.

Return value: Type: String
 Description: A String of the specified targetLength with padString applied from the start.

String.split

Description: Takes a separator parameter and divides a String into an ordered list of substrings by searching for the separator pattern, puts these substrings into an array, and returns the array.

Parameters: Name: string Type: String Description: A string.
 Name: separator Type: Integer Description: The pattern describing where each split should occur.

Return value: Type: array:string
 Description: An Array of strings, split at each point where the separator occurs in the given string.

15.3.6. Array**Array.length**

Description: The number of elements in given array.

Parameters: Name: array Type: Array Description: An array.

Return value: Type: Integer
 Description: The length of an array.

Array.slice

Description: Returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.

Parameters: Name: array Type: Array Description: An array.
 Name: start Type: Integer Description: Zero-based index at which to

start extraction.

Name: end Optional: Yes Type: Integer Description: Zero-based index at which to end extraction.

Return value: Type: Array
Description: A new array containing the extracted elements.

Array.allocate

Description: Creates a new array of given size.

Parameters: Name: size
Type: Array
Description: A size number.

Return value: Type: Array
Description: A new array.

Array.append

Description: To an existing array appends an element to the end and returns a new array. The original array will not be modified.

Parameters: Name: array Type: Array Description: An array.
Name: value Type: any Description: Element value to be appended.

Return value: Type: Array
Description: A new array with appended element.

Array.insert

Description: To an existing array inserts an element at given position and returns a new array. The original array will not be modified.

Parameters: Name: array Type: Array Description: An array.
Name: position Type: Integer Description: Zero-based index at which new element will be inserted.
Name: value Type: any Description: Element value to be inserted.

Return value: Type: Array
Description: A new array with inserted element.

Array.remove

Description: From an existing array removes an element at given position and returns a new array. The original array will not be modified.

Parameters: Name: array Type: Array Description: An array.
Name: position Type: Integer Description: Zero-based index from which existing element will be inserted.

Return value: Type: Array
Description: A new array with element removed.

Array.clone

Description: Deep clone of the array.

Parameters: Name: array
Type: Array
Description: An array.

Return value: Type: Array
Description: A new array.

15.3.7. LVGL

LVGL.MeterTickIndex

Description: See the *LVGL Meter* Widget description (Chapter XX) for the purpose of this function.

Parameters: None

Return value: Type: Integer
Description: Index number.

15.4. Expression Builder

Expressions are supported in both Action and Widget components. Each property of component that can be evaluated from the expression has "..." icon which opens *Expression Builder* (Fig. 134). Expressions can be entered manually or using the *Expression Builder*.

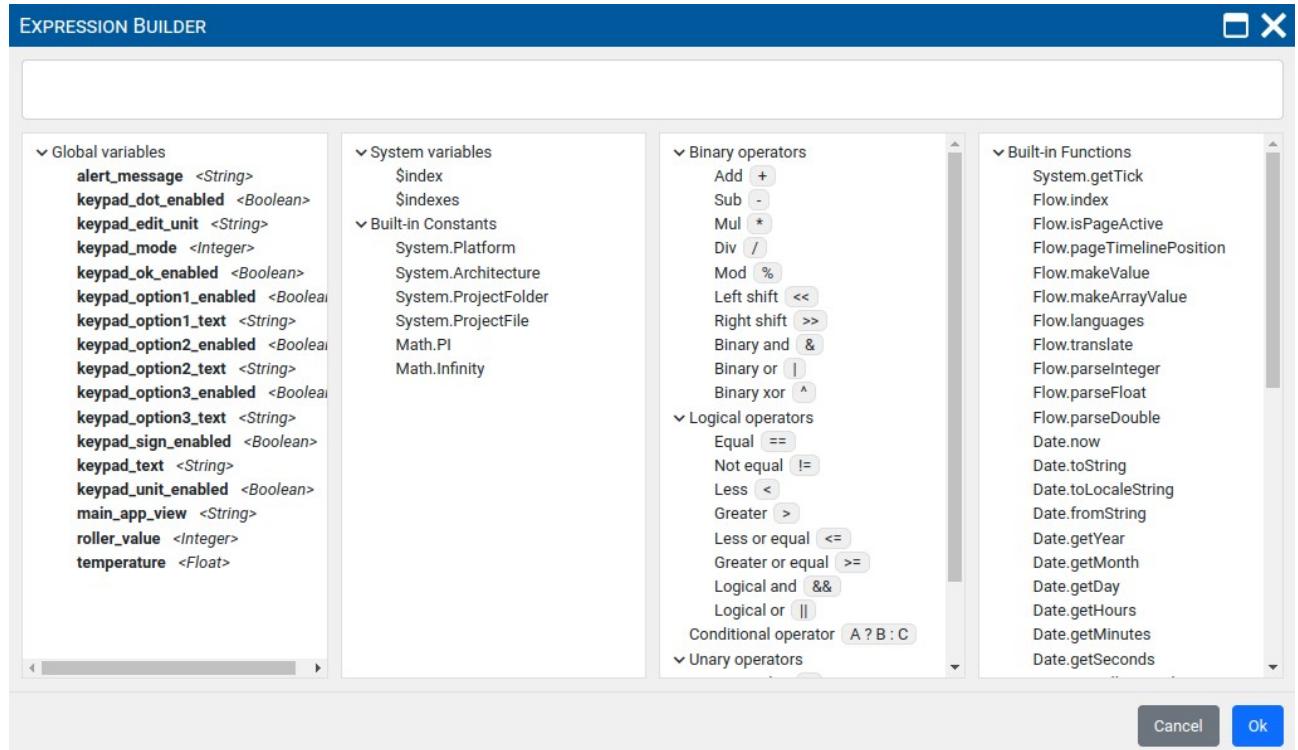


Fig. 134: Expression builder

For more info visit: www.envox.eu
File repository: <https://github.com/eez-open>

Version: M14 DRAFT
Date: 2023-05-23