EEZ Studio User manual

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

Ver. M15 DRAFT – 06/2023
www.envox.eu
github.com/eez-open

Table of Contents

C1. Legal information

C1.1. 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.

C1.2. 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.

C1.3. 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.

C1.4. 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

C1.5. Revision history

Date	Version	Changes
xxxx-xx-xx	1.0	Initial release

C2. The EEZ Studio overview

C2.1. 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.

C2.2. 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.

C2.3. 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

C2.4. 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.

C3. Installation

C3.1. 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.

C3.2. 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.

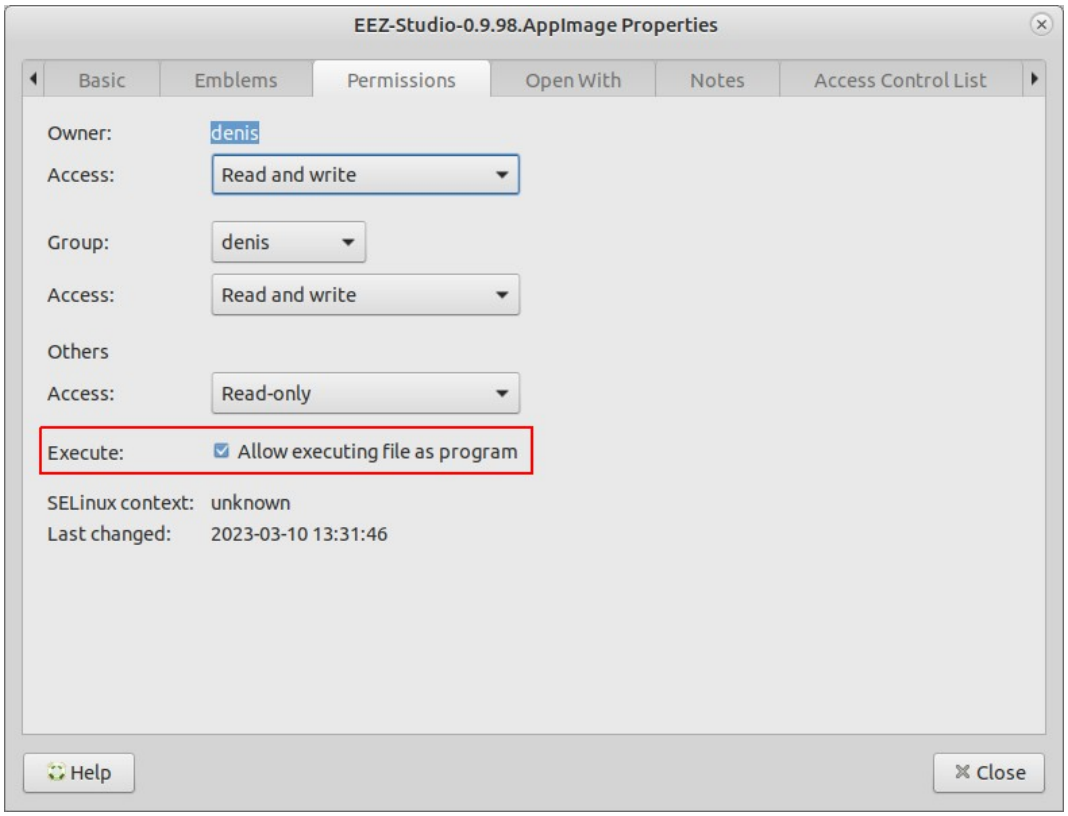


Fig. 1: .AppImage file permission

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`

C3.3. Mac

Required OS version: macOS 10.10 (Yosemite) or newer
Download *eezstudio-mac.zip*, unpack and move *eezstudio.app* to Applications.

C3.4. Windows

Required OS version: Windows 7 (64-bit) or newer
Download and start *EEZ_Studio_setup.exe*.

C3.5. 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.

C3.6. 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

C3.6.1. Linux only

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

C3.6.2. Raspbian only

Install *Node.js 16* and *npm* on Raspberry Pi: https://lindevs.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
```

C3.6.3. 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
```

C3.6.4. Raspbian

```
npm run dist-raspbian
```

C3.6.5. Nix

To build:

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

To start:

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

C3.7. 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.

C3.7.1. Windows

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

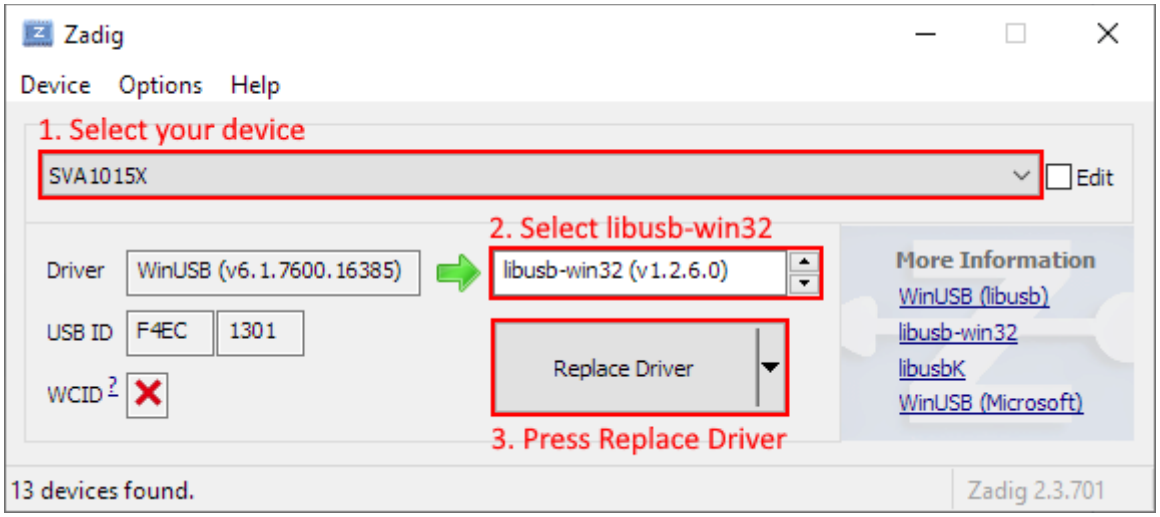


Fig. 2: Zadig driver settings

C3.7.2. 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.

C3.8. 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`

C4. Key features

C4.1. 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

C4.2. 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)

C4.3. 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

C5. Menu options and Settings

C5.1. 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.

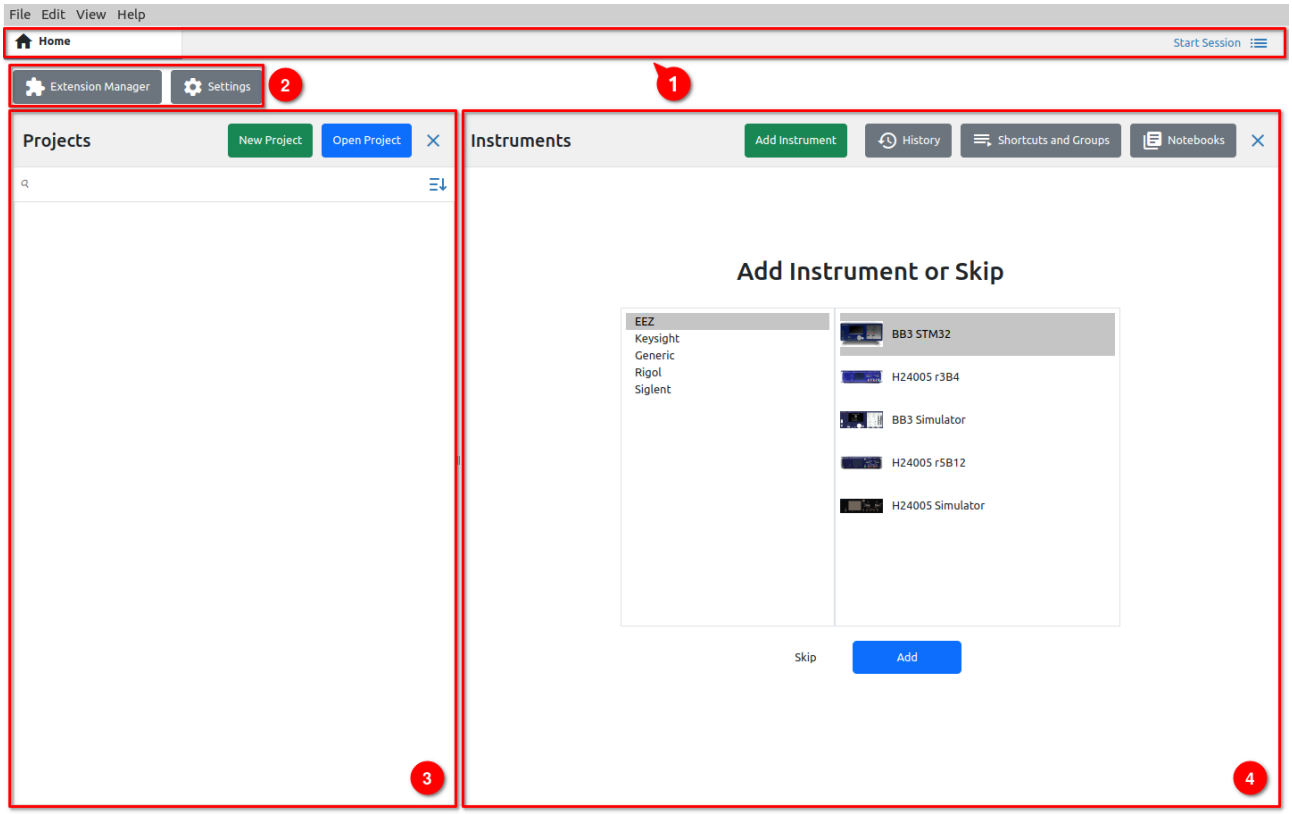


Fig. 3: Home page

C5.2. Menu options

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

C5.2.1. 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.

C5.2.2. Edit

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

C5.2.3. View

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

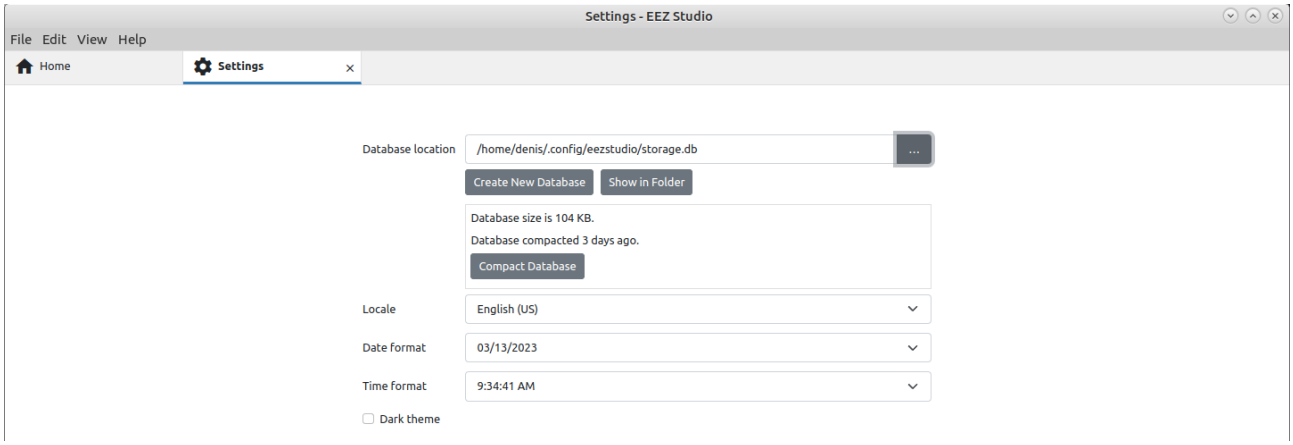


Fig. 4: Settings tab

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).

C5.2.4. Help

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

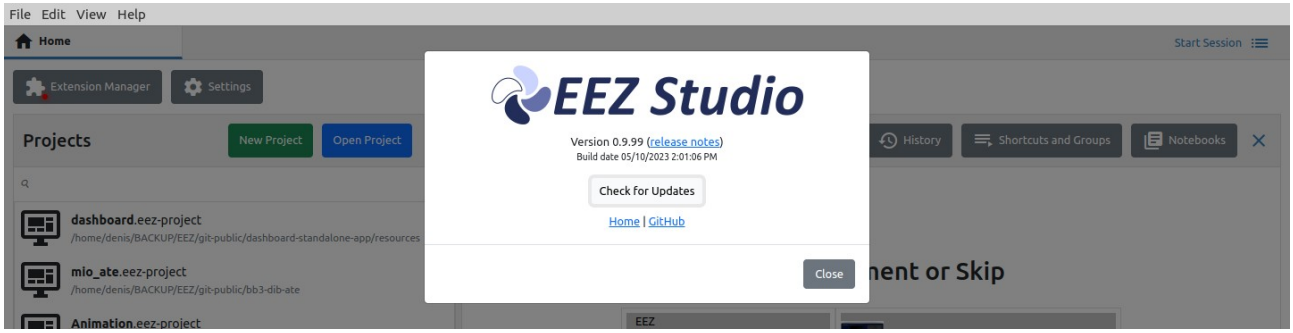


Fig. 5: About page

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 Project

Table of Contents

P1. 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. 1. which displays a searchable Recent Project List (RPL).

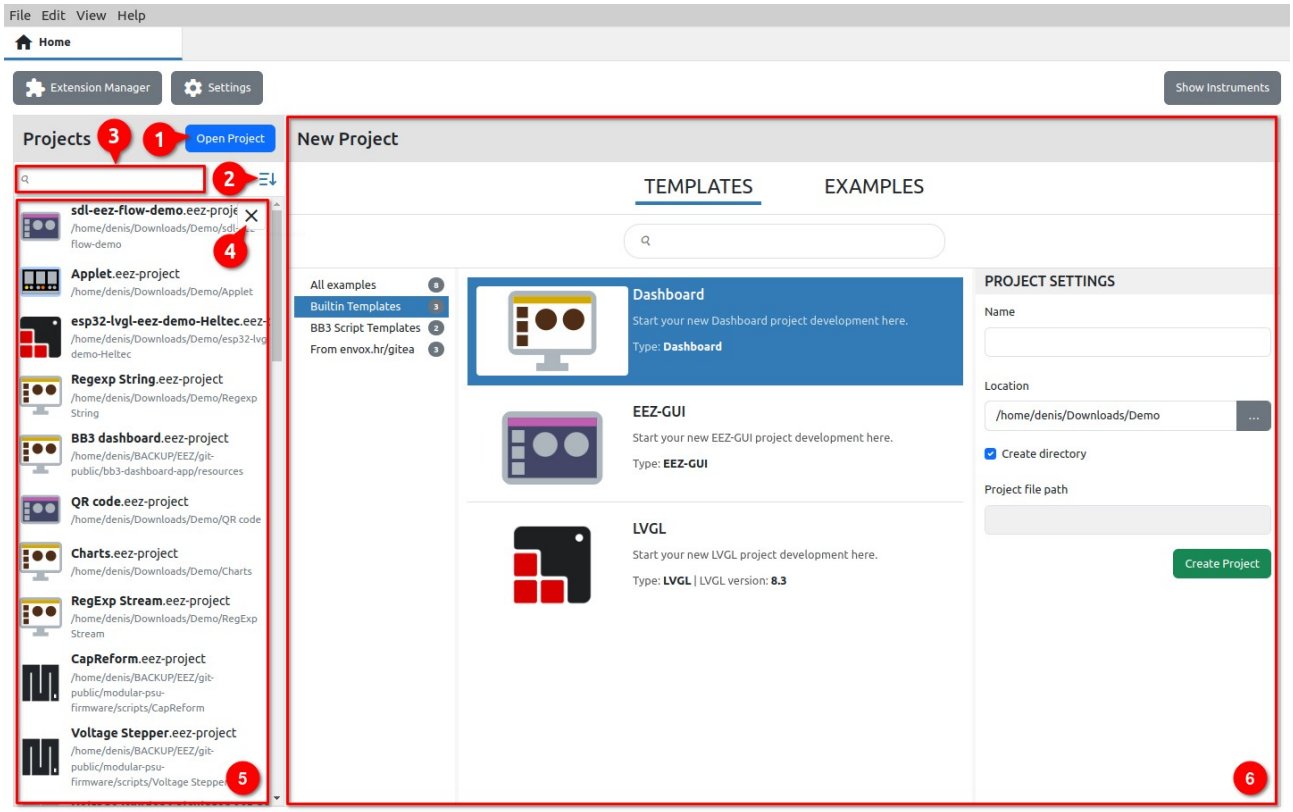


Fig. 1: Home page project options

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

P1.1. 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.

P1.2. Create new project

Creating a new dialog is possible from the New Project section, which is displayed to the right of the *Recent Project List* (Fig. 3). If the Instruments section is also enabled on the home page, *New project* will be displayed as a button and a new dialog box will open for selecting a project from the offered Templates and Examples. (Fig. 2).

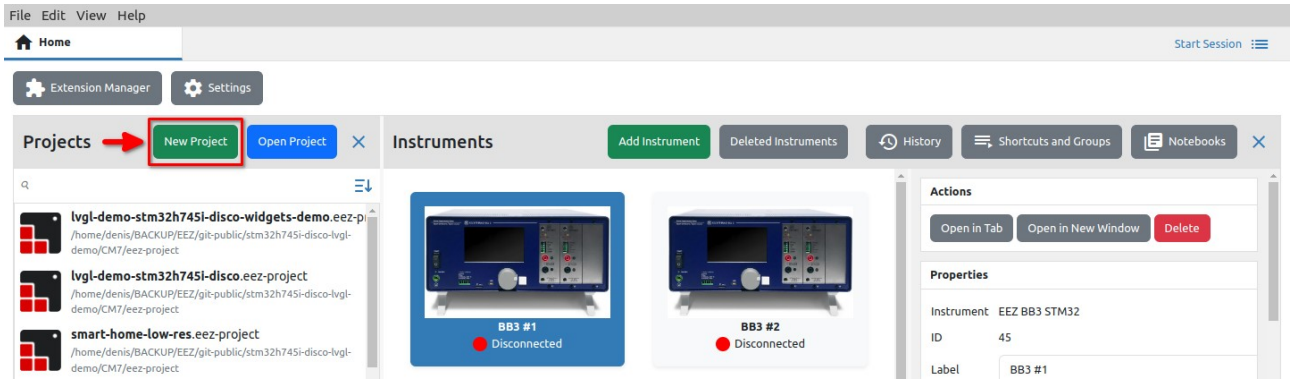


Fig. 2: New project as button when the Instruments section is also enabled

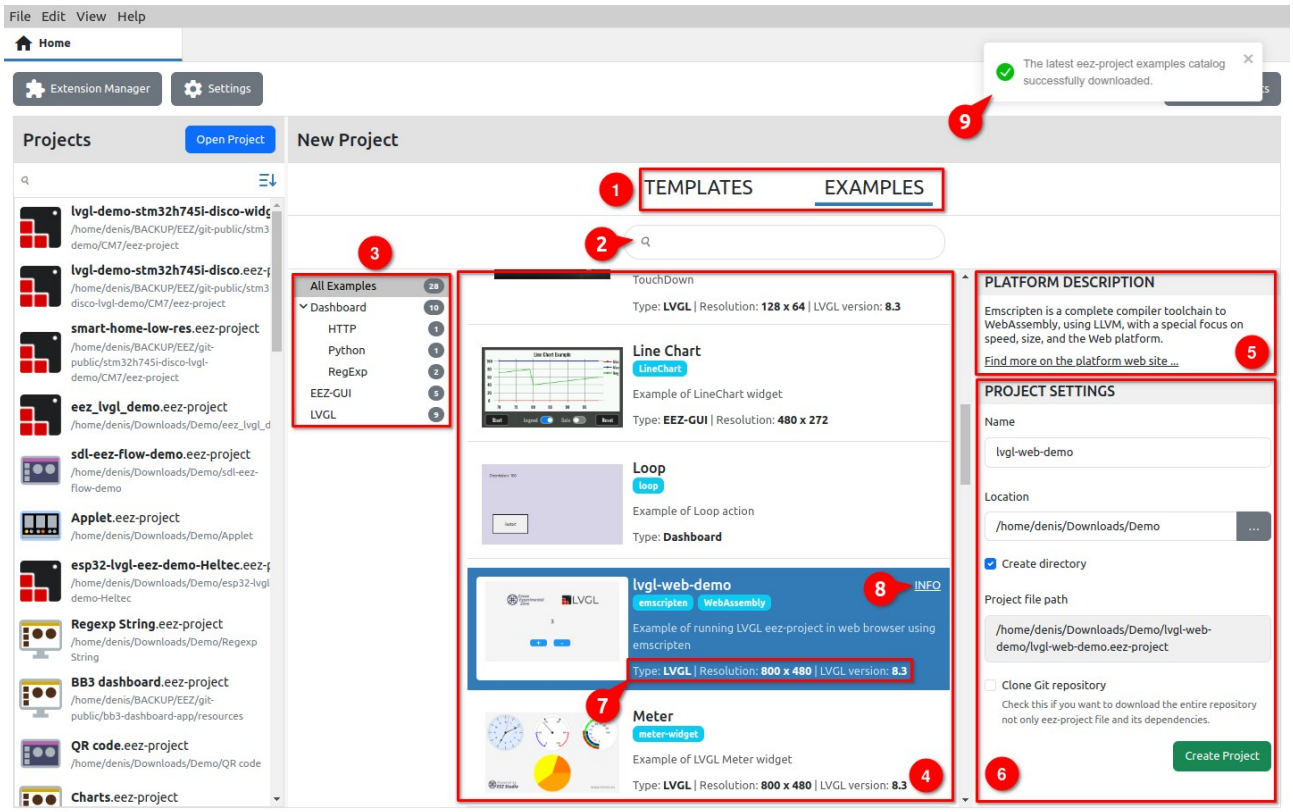


Fig. 3: Create new project

#	Option	Description
1	*Example category selection*	Examples are listed in two categories: project Templates contain the basic configuration for the selected type of project, while project Examples represent a functional application that can contain multiple Pages with User Actions and User Widgets.
2	*Search*	Search by project name.
3	*Project list*	List of all projects within the selected category, grouped into expandable sublists.
4	*Project selector*	Project selector from the currently selected subgroup. By navigating through the list, the Project settings are displayed on the right. Positioning the cursor on the project thumbnail changes the cursor icon, and clicking on it enlarges the image.
5	*Platform description*	When present, it provides a description of the target platform for which the project is intended, as well as a link to an external website with additional information about the platform.
6	*Project settings*	Project basic parameters (see below).
7	*Project details*	Basic information about the project: type, screen size and, in the case of an LVGL project, the version of the library used.
8	*Info*	If the project has a Git repository, this link will appear that takes you to the repository home page.
9	*Examples catalog info*	At the start, EEZ Studio checks whether there are changes in the example repository, and if it finds them, it displays this message after a successful download.

P1.3. Project basic settings

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. This option is not available if the project is taken from a Git repository (in this case a new folder is always created).

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

Clone Git repository
When creating a new project from an Example sourced from a Git repository, the `.eez-project` file is always copied. Check this option if you want all other files from the repository to be copied.

Initialize as Git repository
Specifies whether the newly created project from the selected template will be immediately initialized as a Git repository. The option does not need to be checked if you do not use Git for source control.

P1.4. 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. 4) when creating a new project or set a reference to a local copy of the repository (Fig. 5).

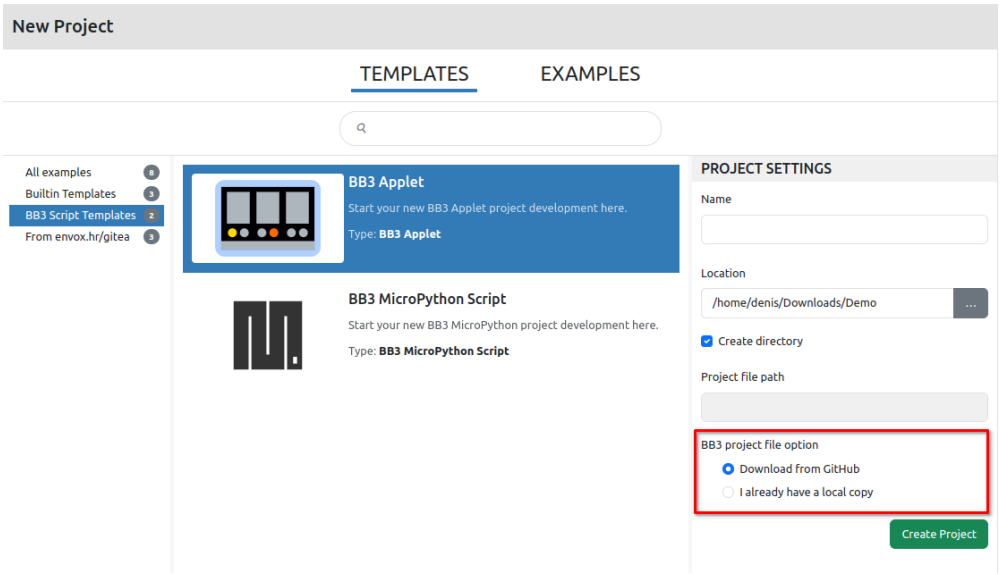


Fig. 4: EEZ BB3 applet new project additional option

When creating a *MicroPython script* project, it will be necessary to define which firmware version is used on the target EEZ BB3 in order to create the corresponding resource file during the build (Fig. 5).

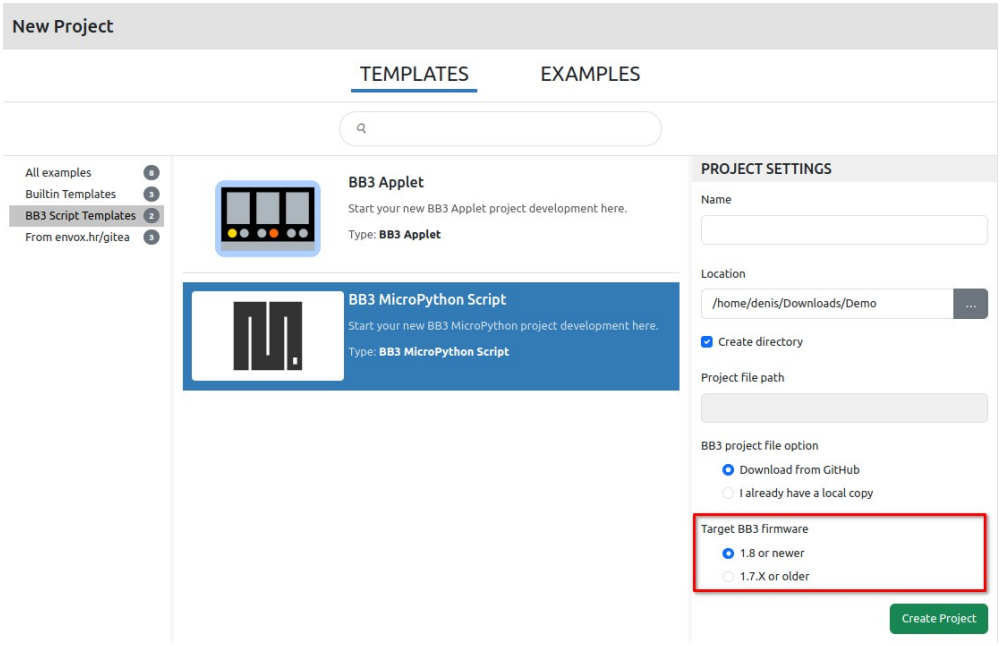


Fig. 5: EEZ BB3 MicroPython new project additional options

After all the basic parameters have been entered, the new project will be displayed in the project editor in *Edit* mode. Fig. 6 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.

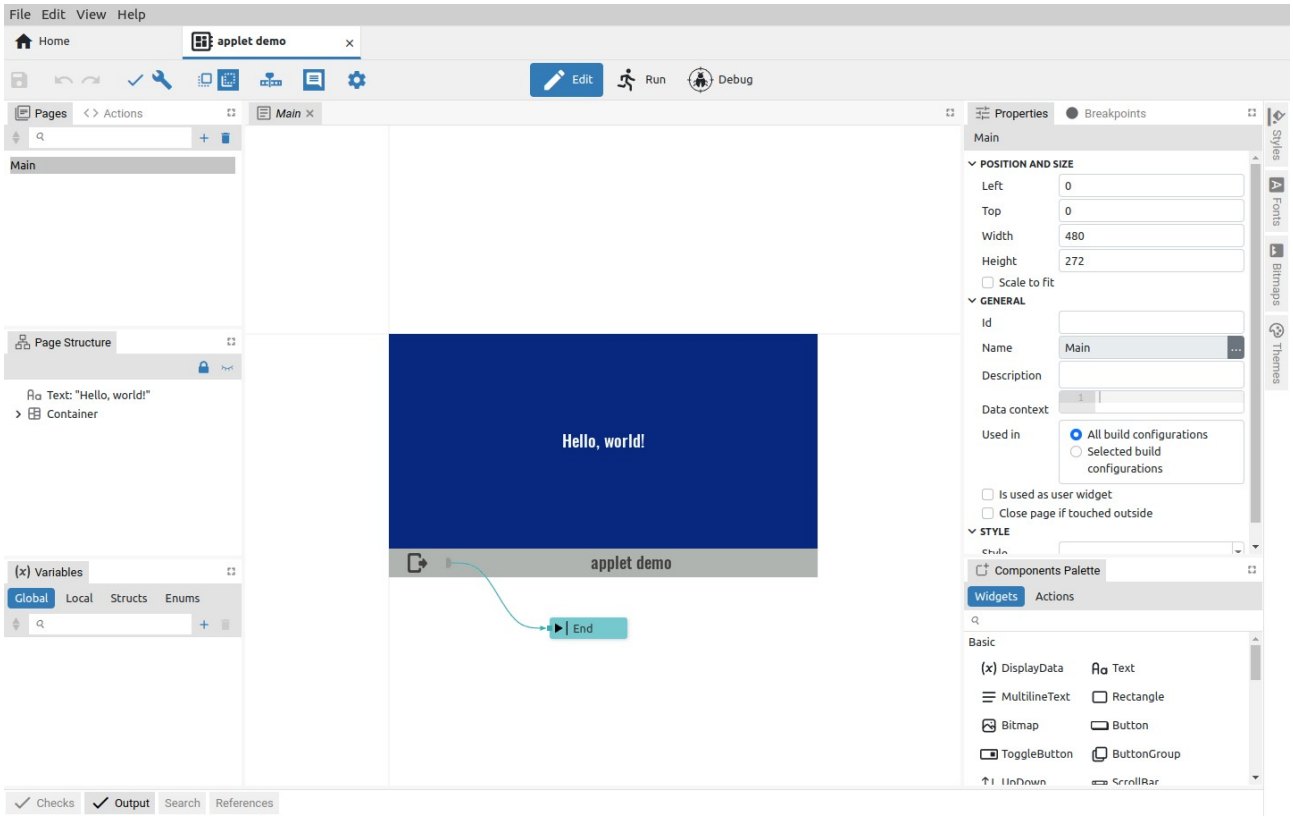


Fig. 6: Newly created project in Edit mode

The basic project settings set by the New project 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. 7.

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).

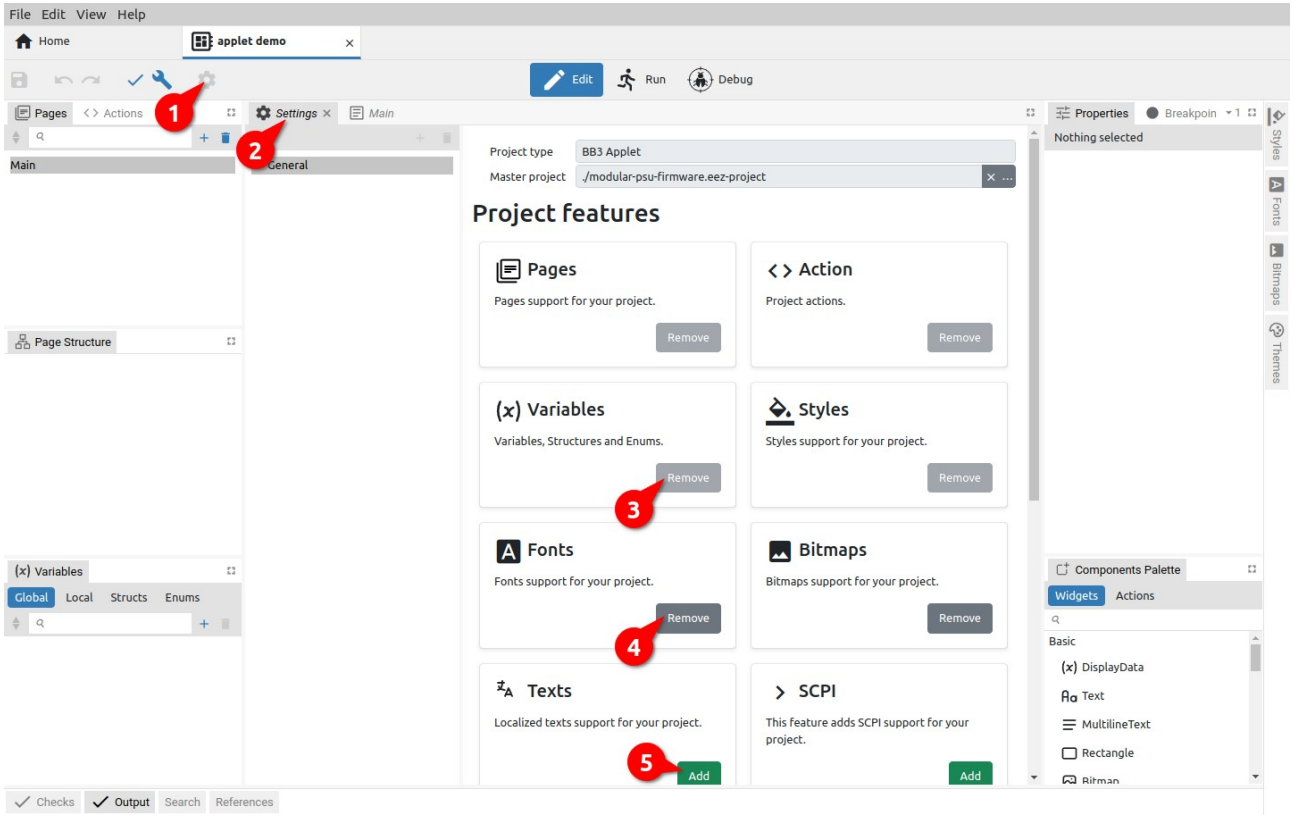


Fig. 7: Newly created project settings

P2. 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.

P2.1. Project editor workspace

Fig. 8 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.

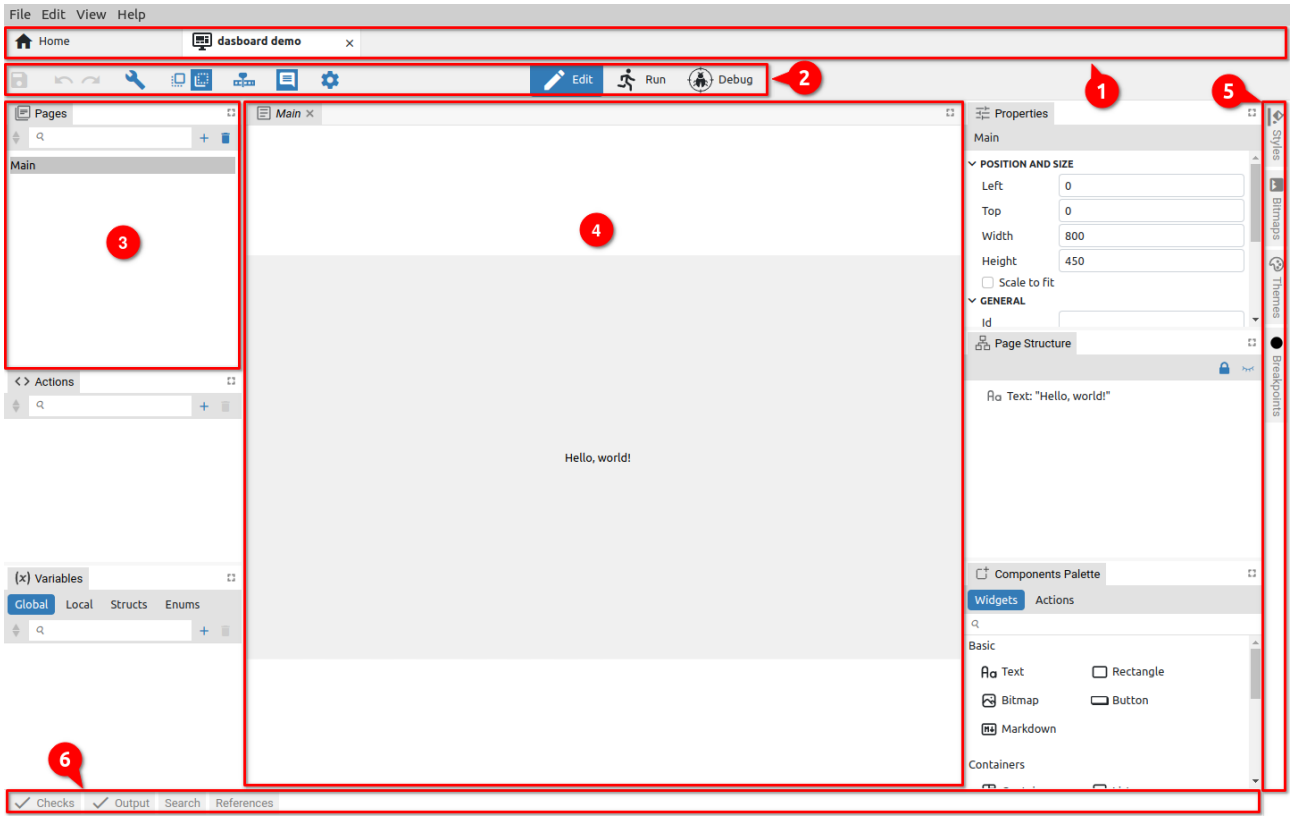


Fig. 8: Project editor sections

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 tabsets. Tabsets 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. 8 are explained below.

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

P2.2. Display of the page in the editor

In Fig. 9 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).

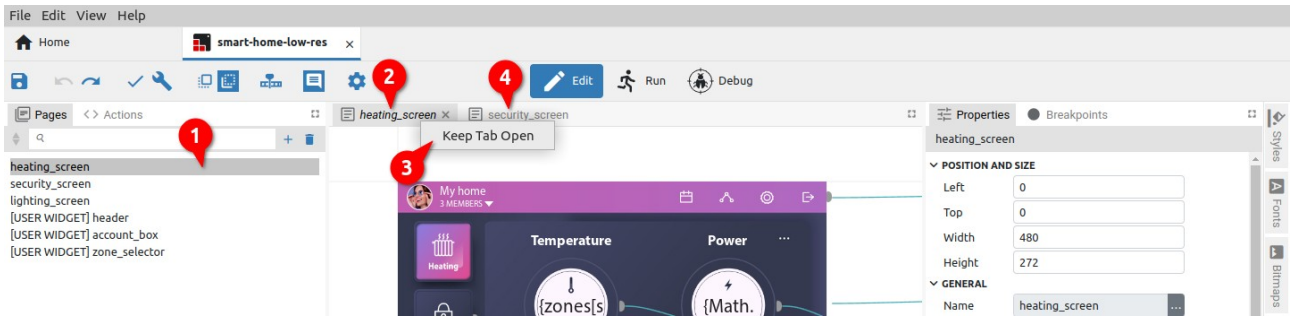


Fig. 9: Page editor tab locking

P2.3. 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. 10).

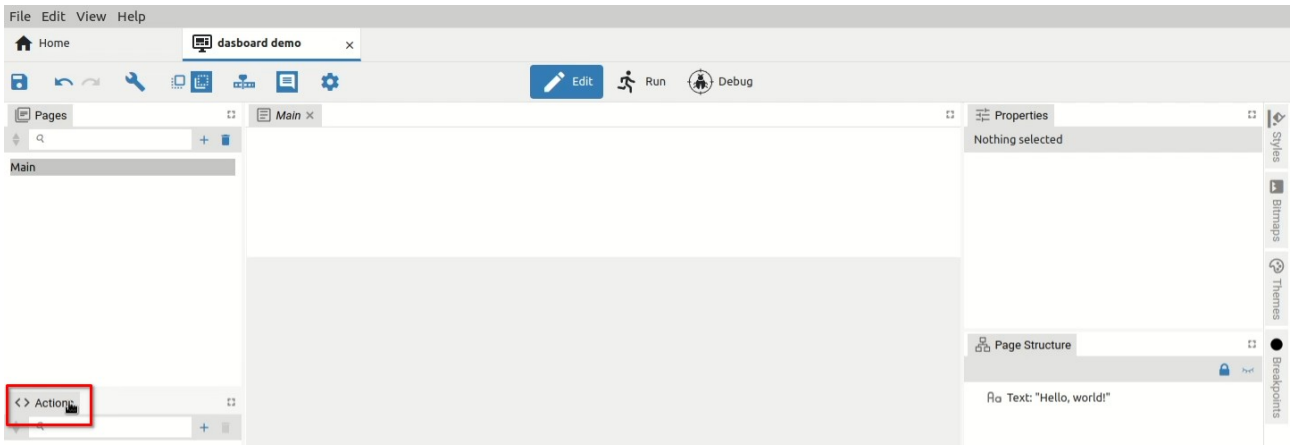


Fig. 10: Panel selection

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 tabsets (Fig. 11).

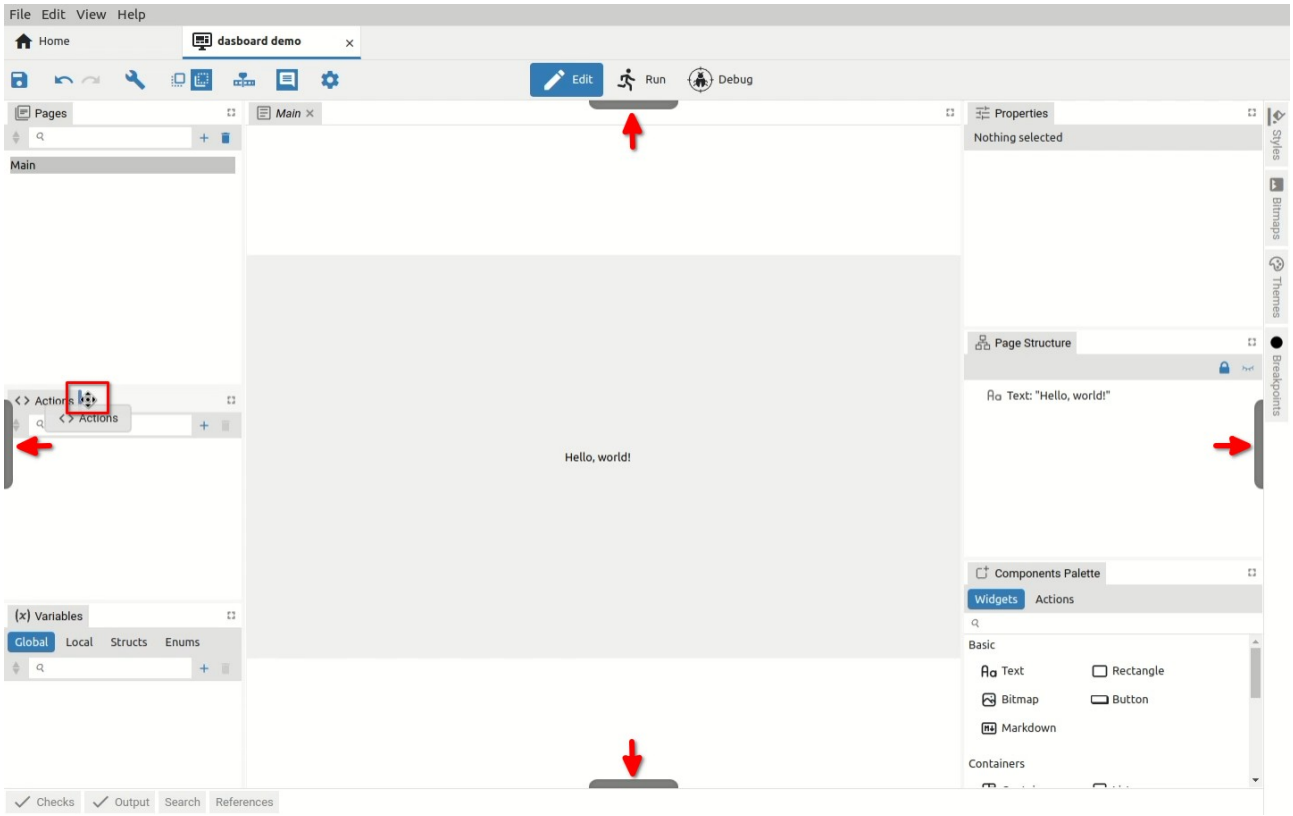


Fig. 11: Docking indicators for border tabs

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. 12.
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. 13.

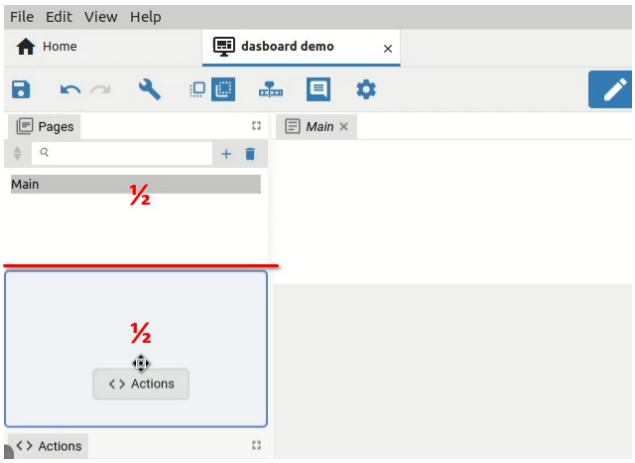


Fig. 12: Panel horizontal positioning

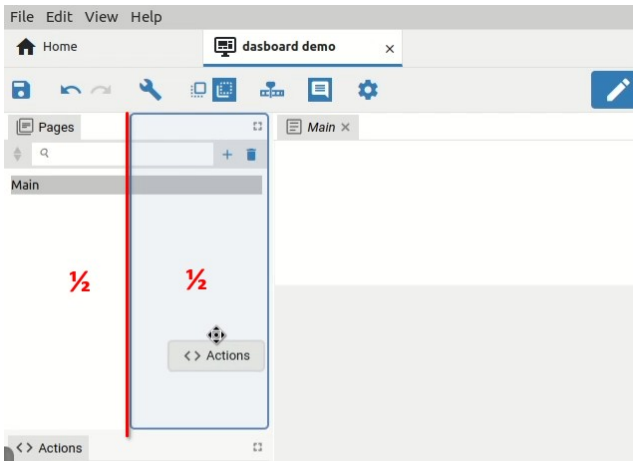


Fig. 13: Panel vertical positioning

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. 14 or to place the cursor approximately in the middle of the existing tab so that a rectangle appears as in Fig. 15.

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. 12 and Fig. 13.

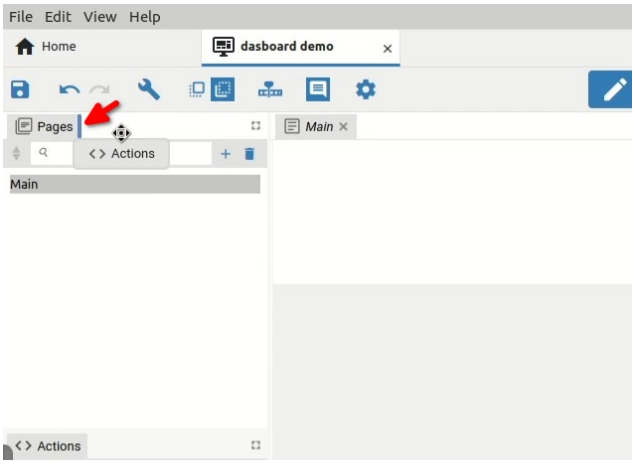


Fig. 14: Positioning in another tabset (1st method)

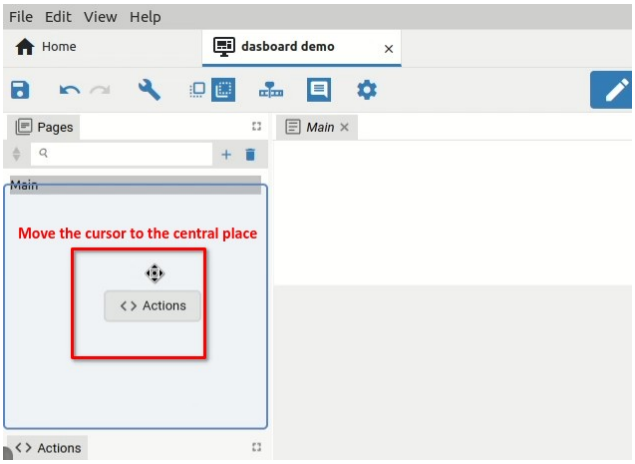


Fig. 15: Positioning in another tabset (2nd method)

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 set with *Pages* (Fig. 16).

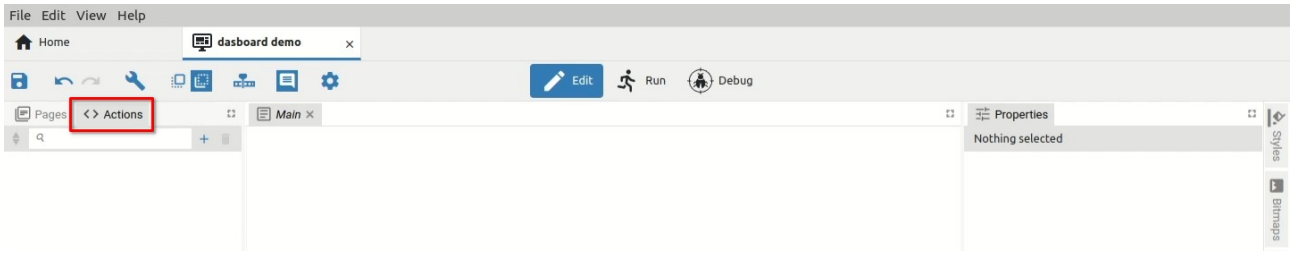


Fig. 16: Panel docking completed

P2.4. Border tabsets

Panels from border tabsets, unlike panels in the workspace, are displayed by clicking the tab (Fig. 17) and closed by clicking the tab again. Only one panel within a border tabset can be open at any time (Fig. 18).

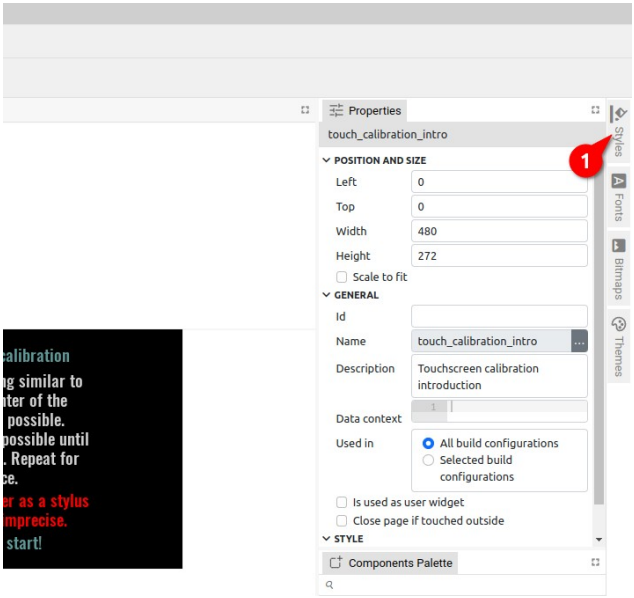


Fig. 17: Border tabs are closed

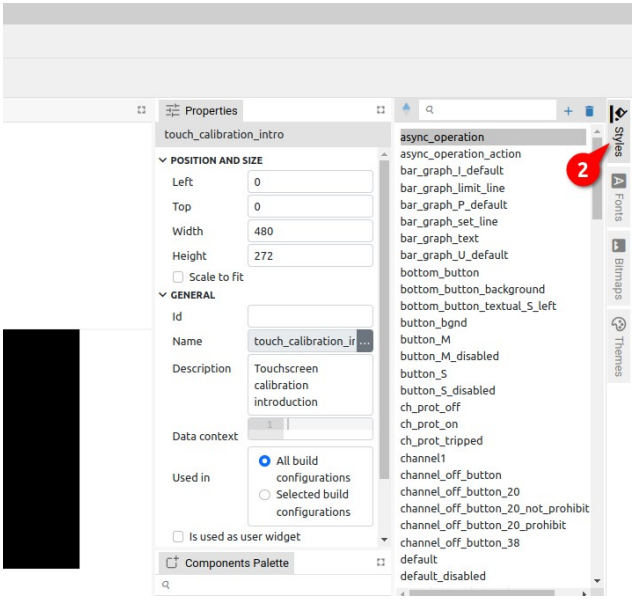


Fig. 18: The border tab is selected and opened

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

P3. 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. 19).

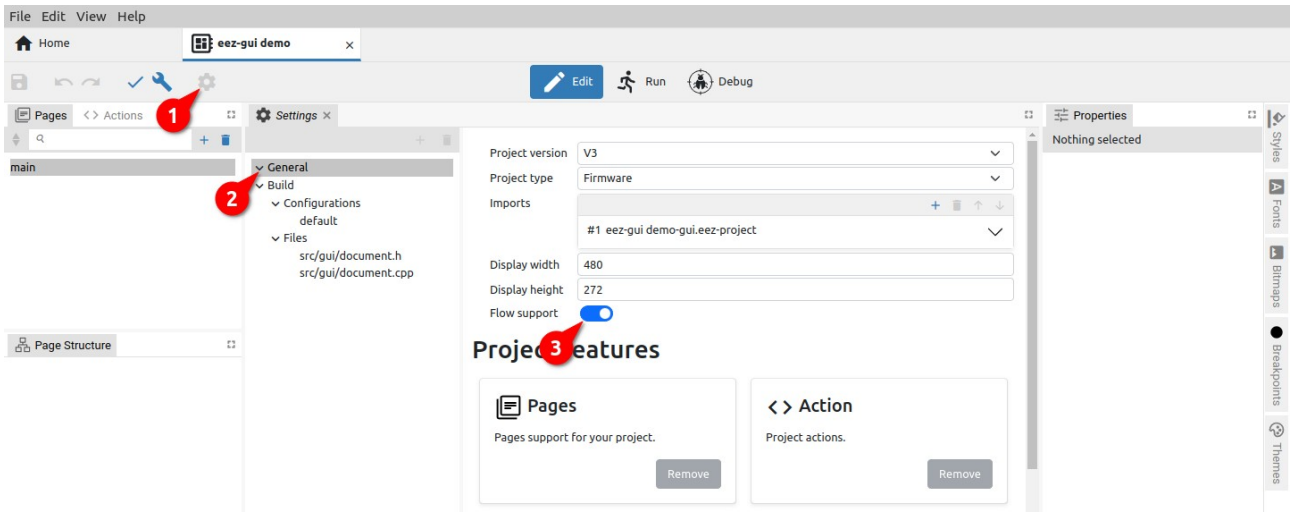


Fig. 19: Enabling EEZ Flow in project Settings

P3.1. 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. 20 shows the toolbar with all options displayed. Their availability in each mode is shown in Table 1.

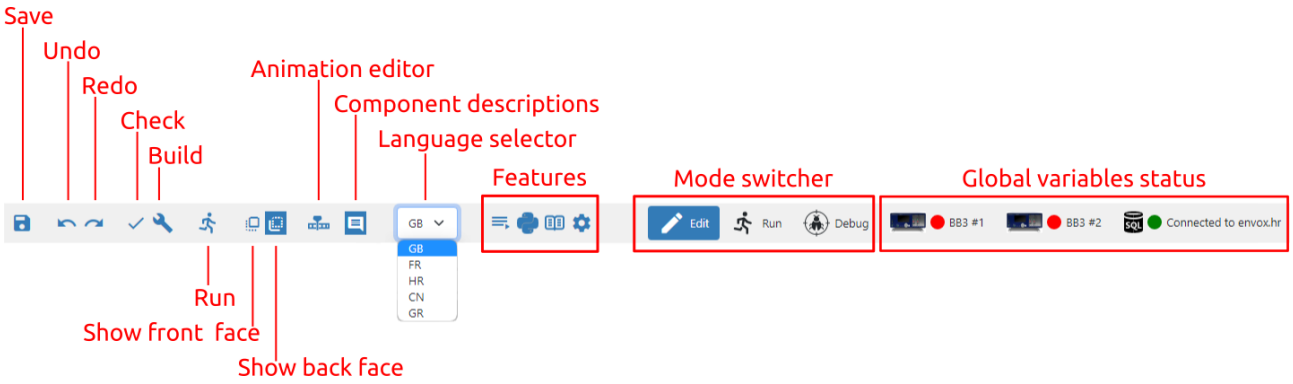


Fig. 20: Project toolbar (all options)

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

P3.2. 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. 21).

Fig. 21: Indication of unsaved changes

Check
All project elements are checked without building the executable code. The Results are displayed in the *Output* panel (Fig. 22).

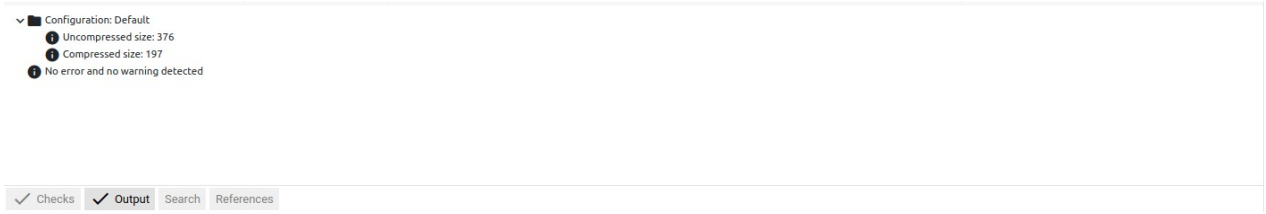


Fig. 22: Results of project checking

Build
Build the executable code after checking all the elements of the project. The results are displayed in the *Output* panel (Fig. 23).

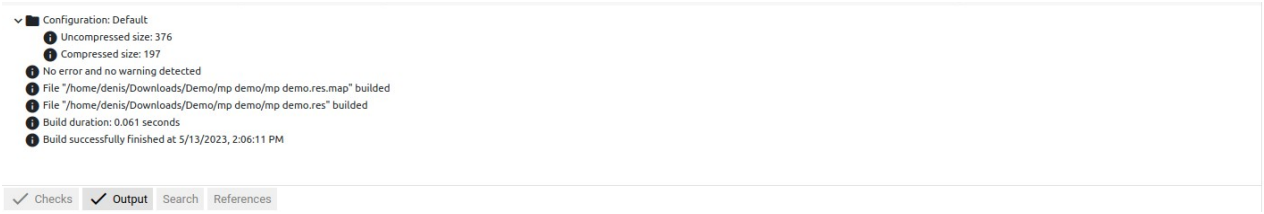


Fig. 23: Project build results

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.

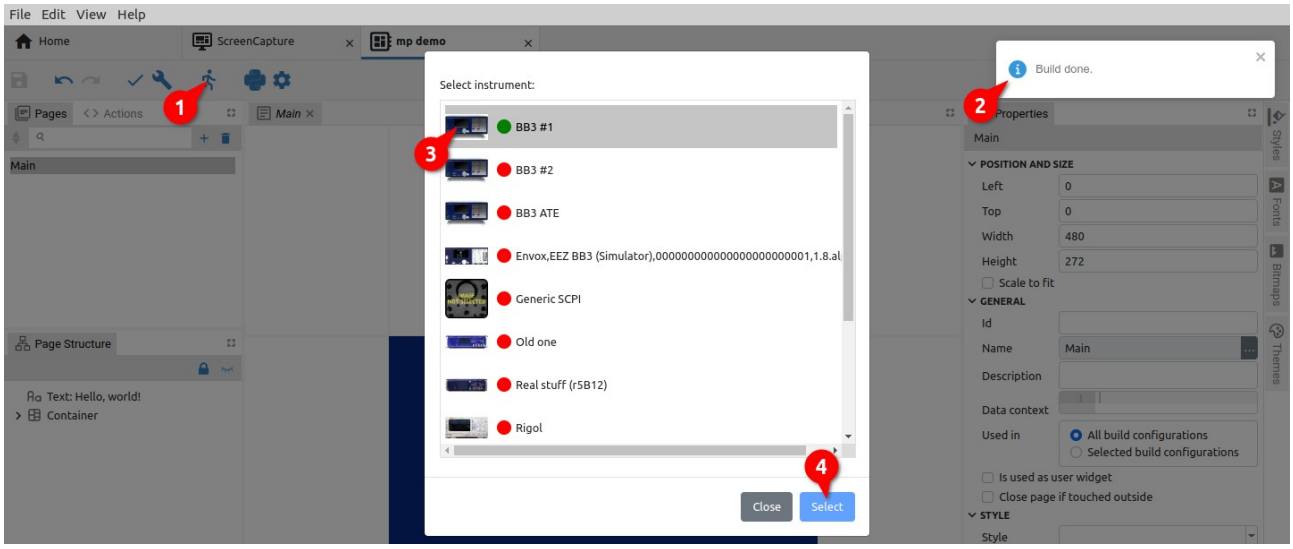


Fig. 24: Selection of target EEZ BB3 for project execution

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. 25 the serial interface is selected.

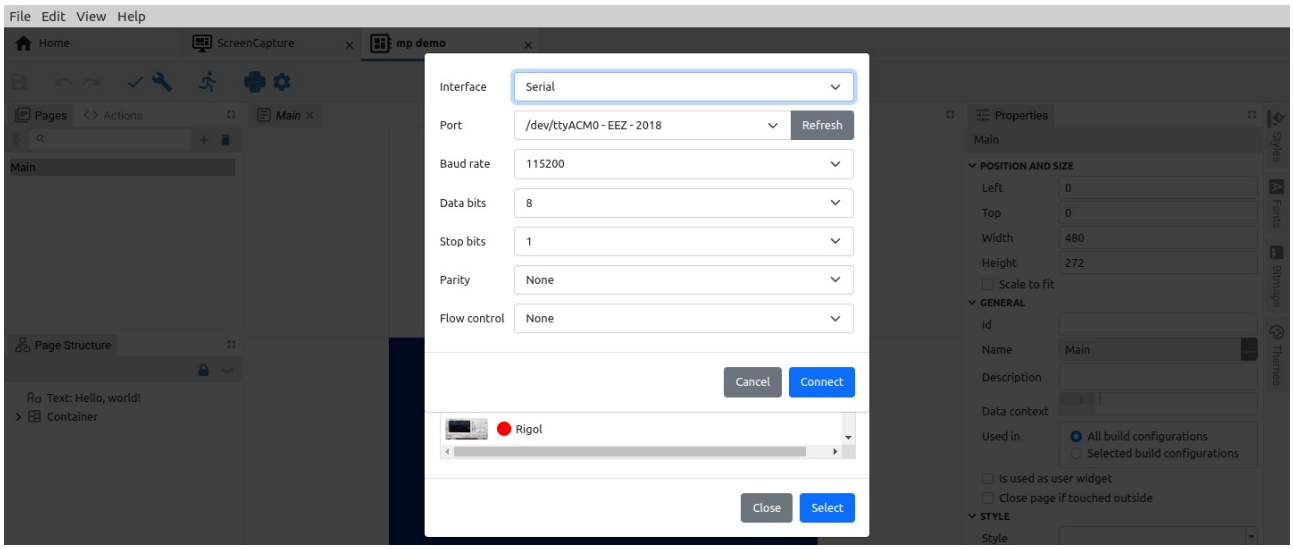


Fig. 25: Selection of target EEZ BB3 for project execution

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. 26).

Fig. 26: 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. 27).

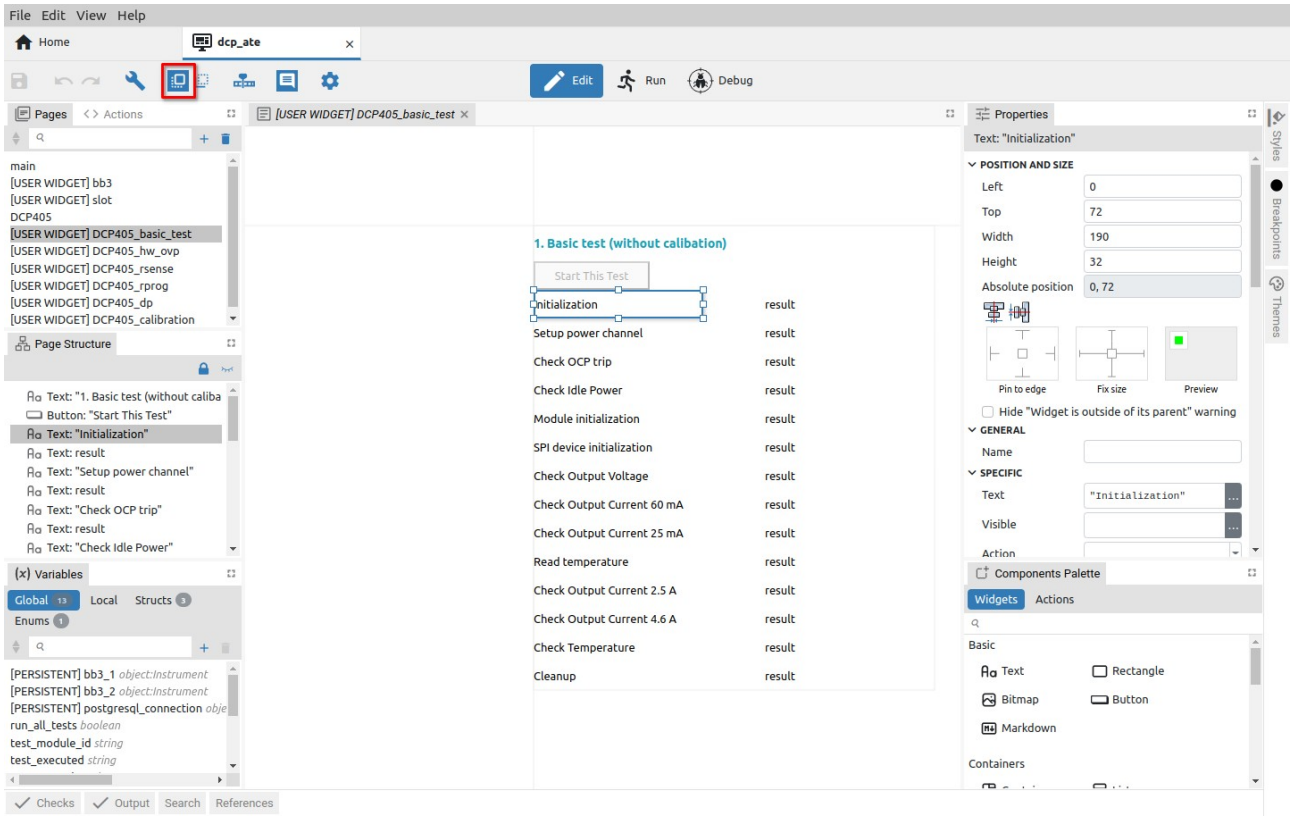


Fig. 27: Project page in front face view

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.28).

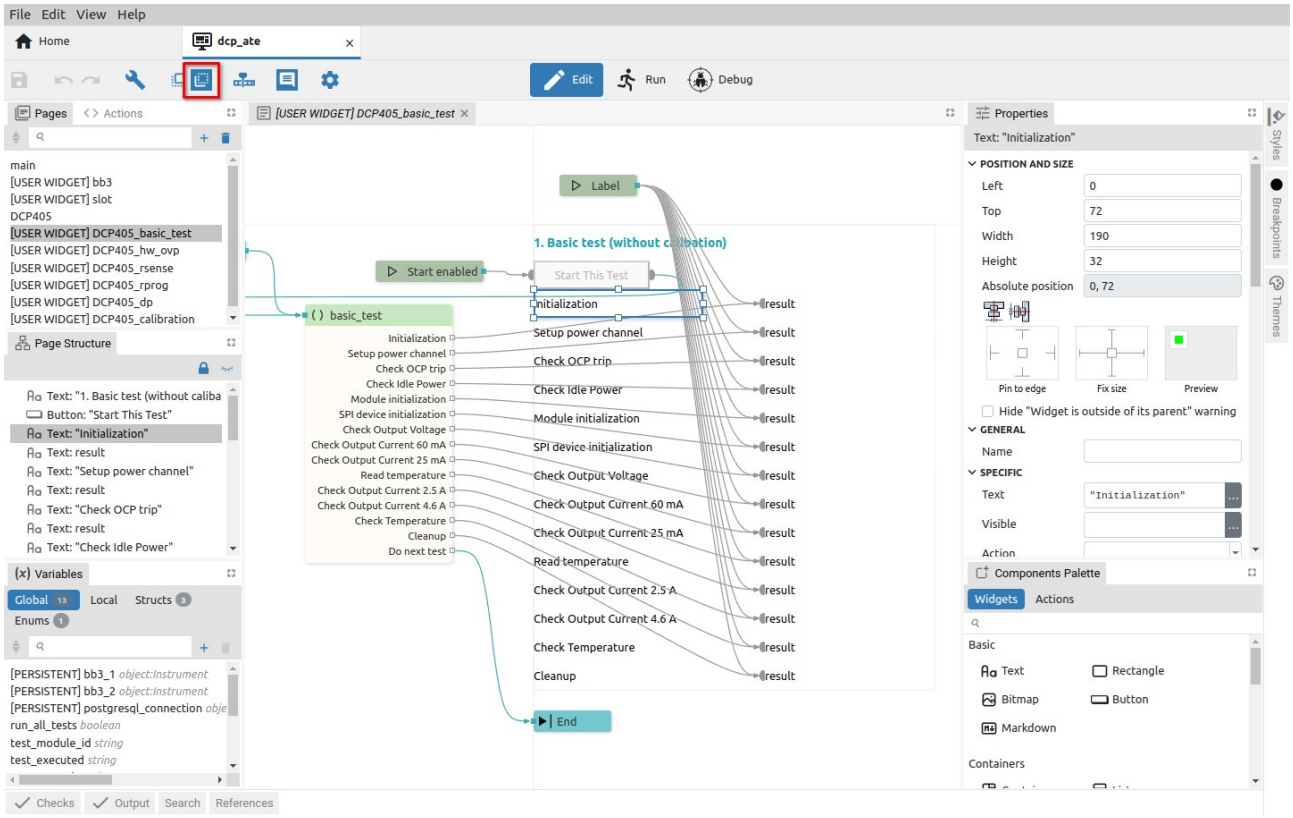


Fig. 28: Project page in back face view

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. 15).

The animation timeline editor is displayed in the space below the page editor (Fig. 29).

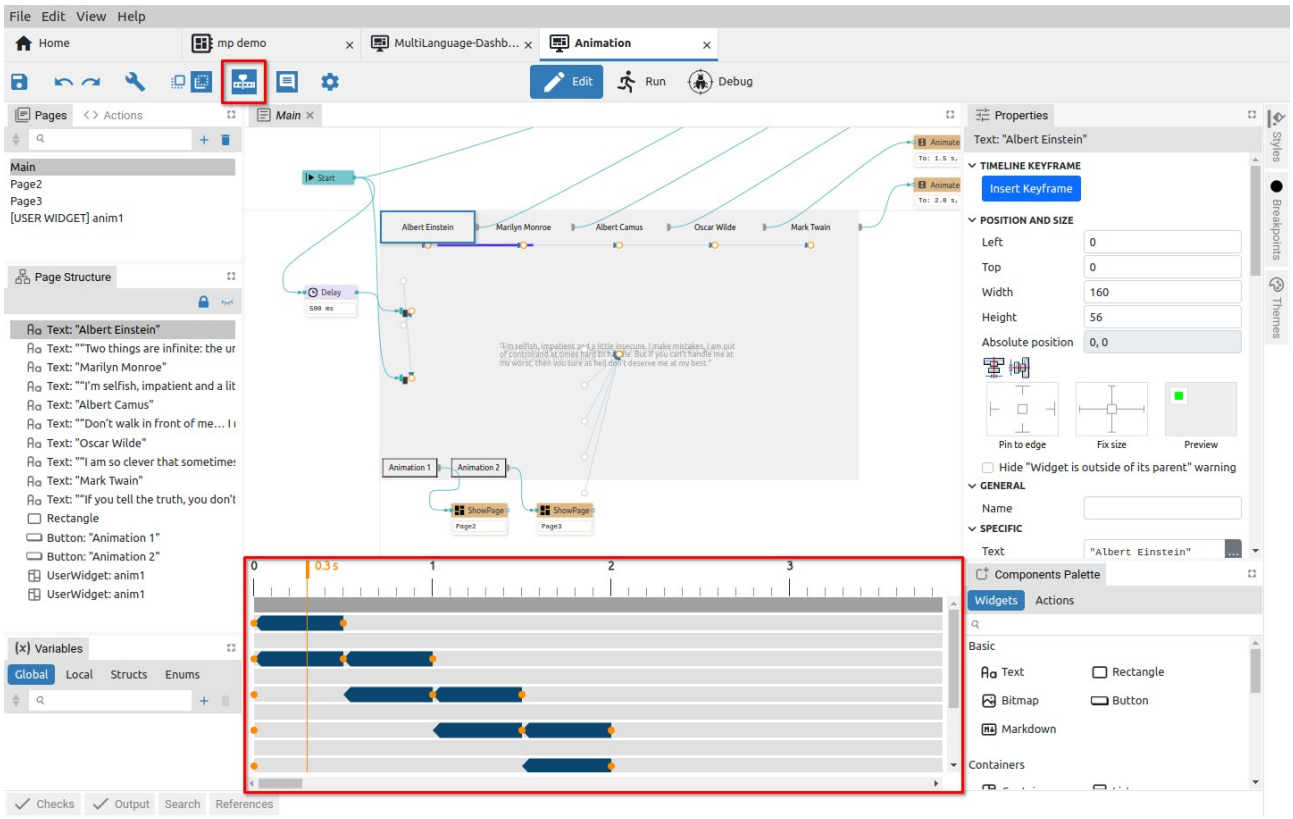


Fig. 29: Project animation timeline editor

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. 30).

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

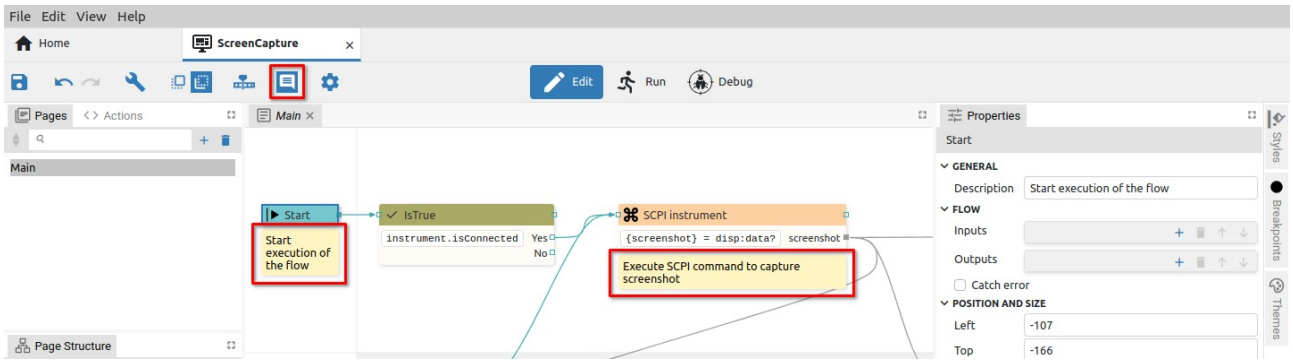


Fig. 30: Component descriptions are visible

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. 31 French (FR) is selected.

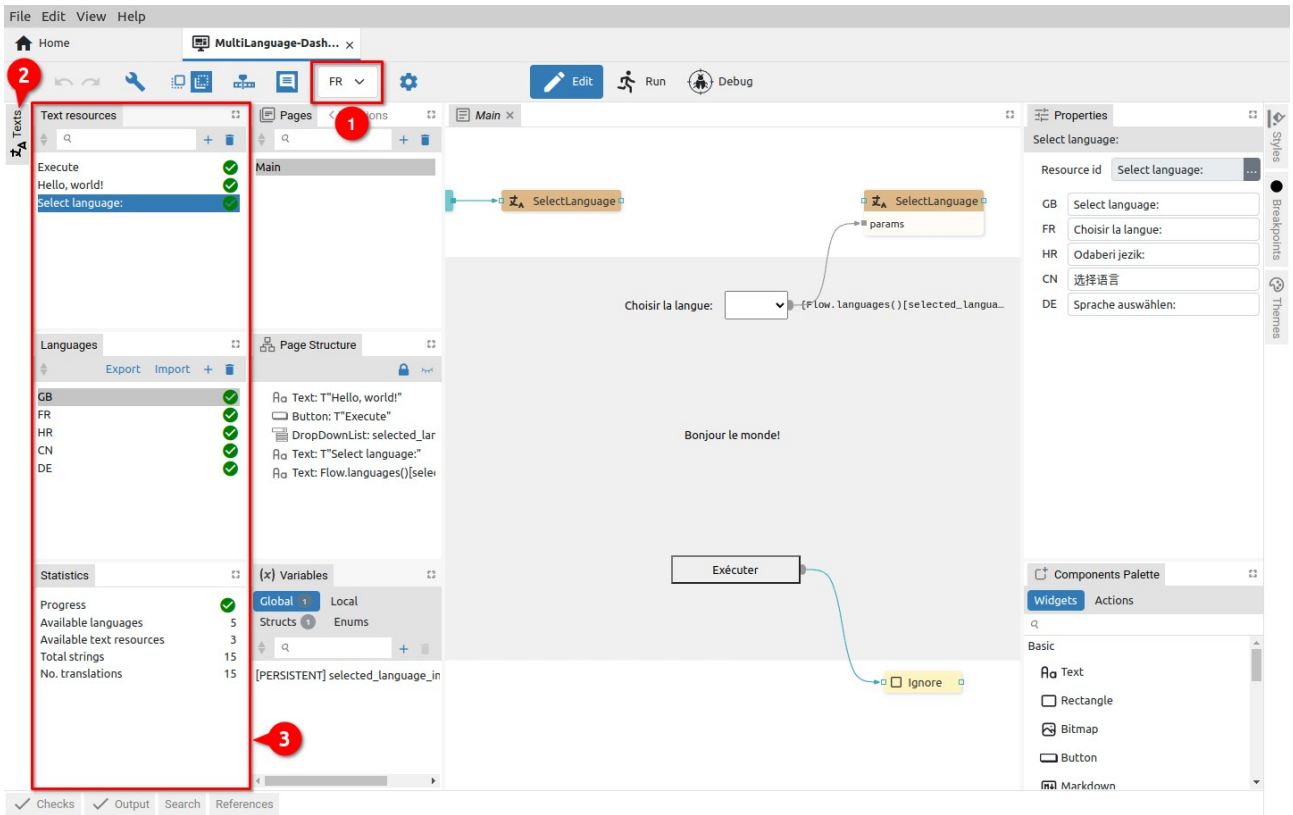


Fig. 31: Project language selector

P3.3. 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.

P4. Project editor panels

P4.1. Panel items

Panel items are marked in Fig. 32 and described below. As you can see, different panels can have different number of items.

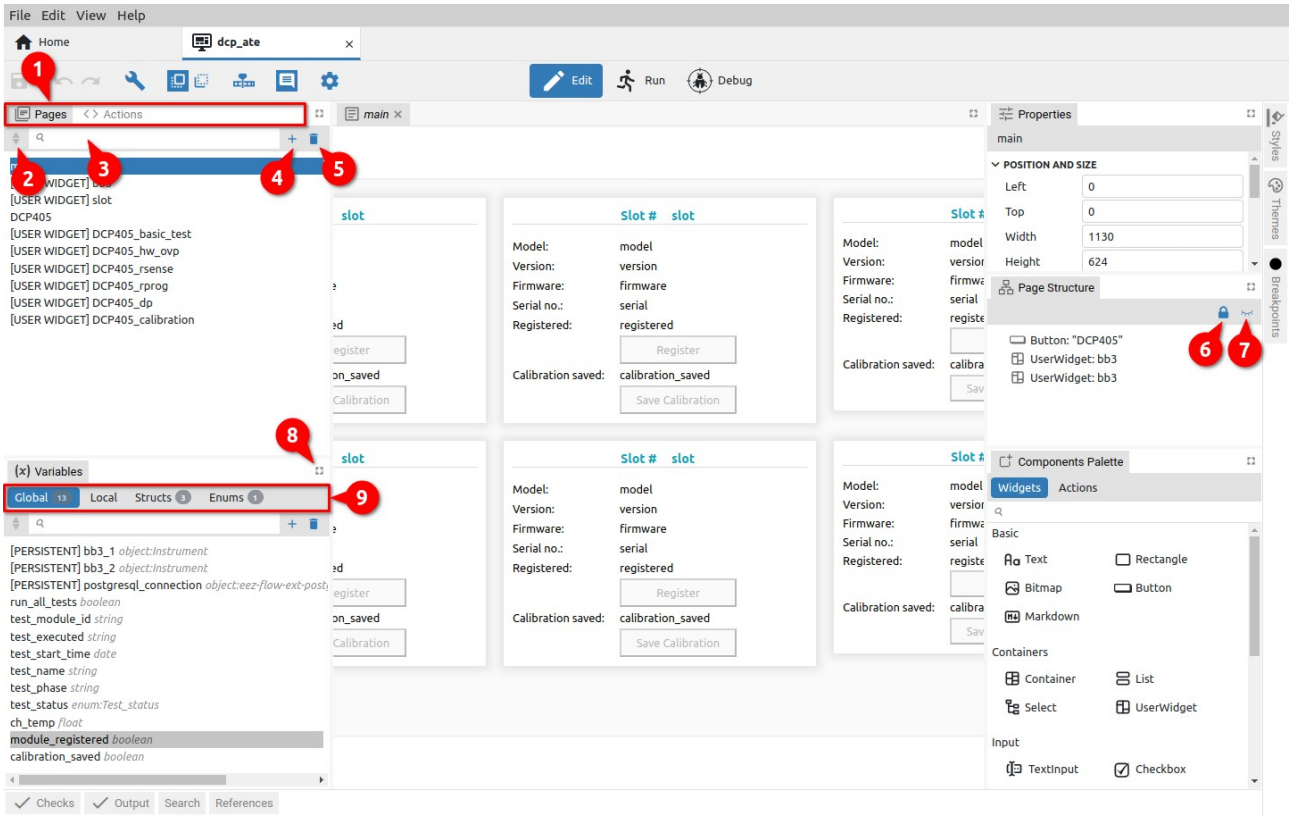


Fig. 32: Panel items

#	Item	Description
1	*Panel tabs*	Selecting panels within a tabset.
2	*Items sort order*	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).

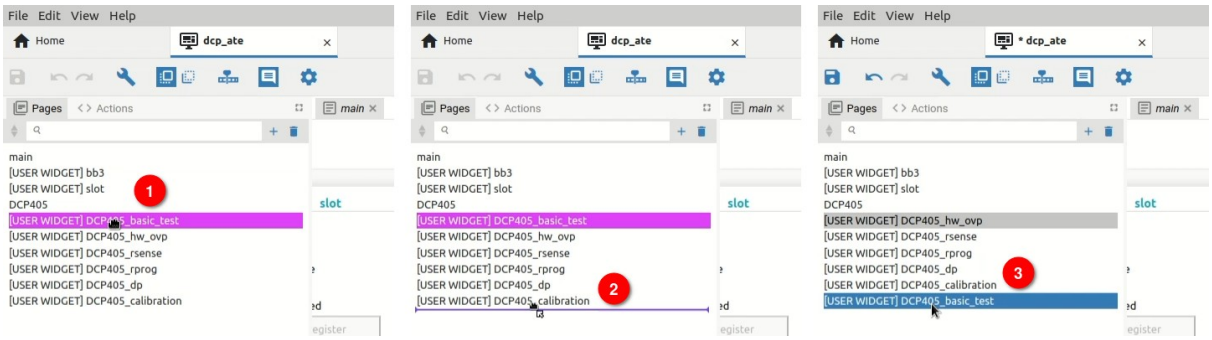


Fig. 33: Changing item position

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. 33).

3	*List filter*	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	*Add item*	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. 34 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.*

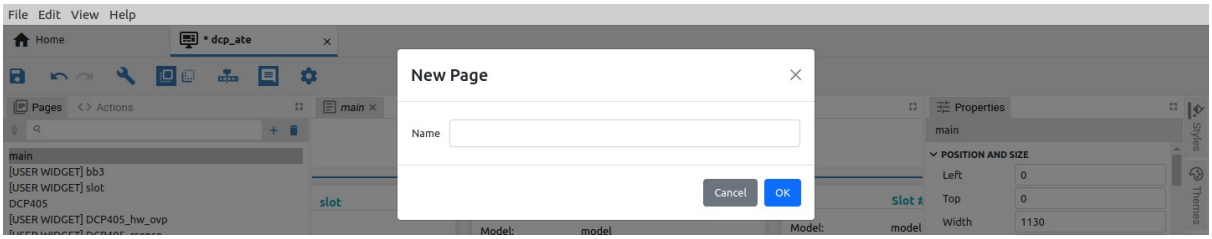


Fig. 34: Adding a new item (Page)

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

P4.2. Right-click menu

Right-click opens a context menu that generally contains options as in Fig. 35. The right-click menu for Widgets (Fig. 36) has a few more options and will be described in Chapter xx.

Fig. 35: Right-click menu (common)

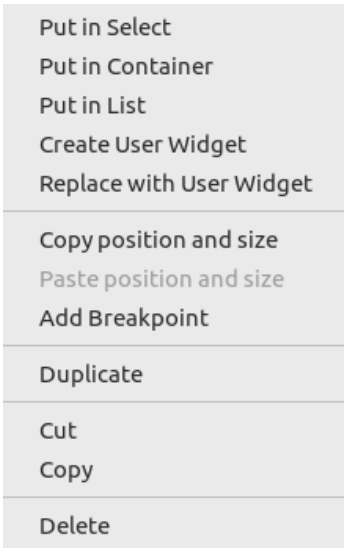


Fig. 36: Right-click menu (Widgets)

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, *main* will be duplicated in *main-1*.
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.

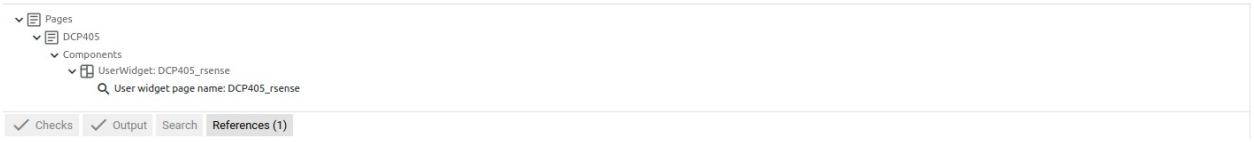


Fig. 37: Displaying the results of the Find all references *operation*

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 *Undo* option in the Toolbar.

P4.3. 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 *Structs* and *Enums* 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 *Bitmaps* 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 *Fonts* 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 *Texts* 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 *IEXT defs* feature is disabled in the project general settings. One project can define multiple IEXT extensions. The IEXT creation procedure is described in Chapter XX.

SCPI (EEZ-GUI only)	List of SCPI commands that will be accessible in IEXT. It will be displayed if the *SCPI* feature is disabled in the project general settings.
Shortcuts (EEZ-GUI only)	List of Shortcuts that will be accessible in IEXT. It will be displayed if the *Shortcuts* feature is disabled in the project general settings.
Changes	List of all commits if the project is in a git repository. It will be accessible if the *Changes* feature is disabled in the project general settings.
Checks	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.
Output	It shows the report after the build is complete and the errors found.
Search	Content search and replace. See Section P4.3.1.
References	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. 37).

P4.3.1. Search and Replace

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

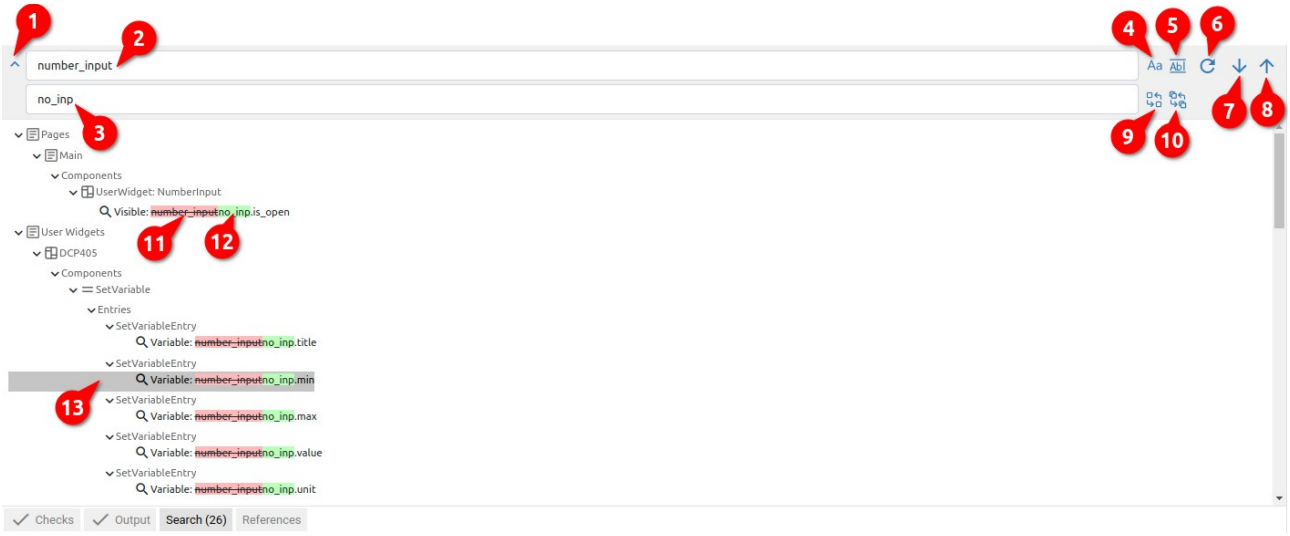


Fig. 38: Search and Replace panel

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

P4.4. Debug mode panels overview

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

P5. Project editors/viewers

P5.1. Editors

The central part of the workspace represents editors tabsets 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.

P5.1.1. 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. 40).

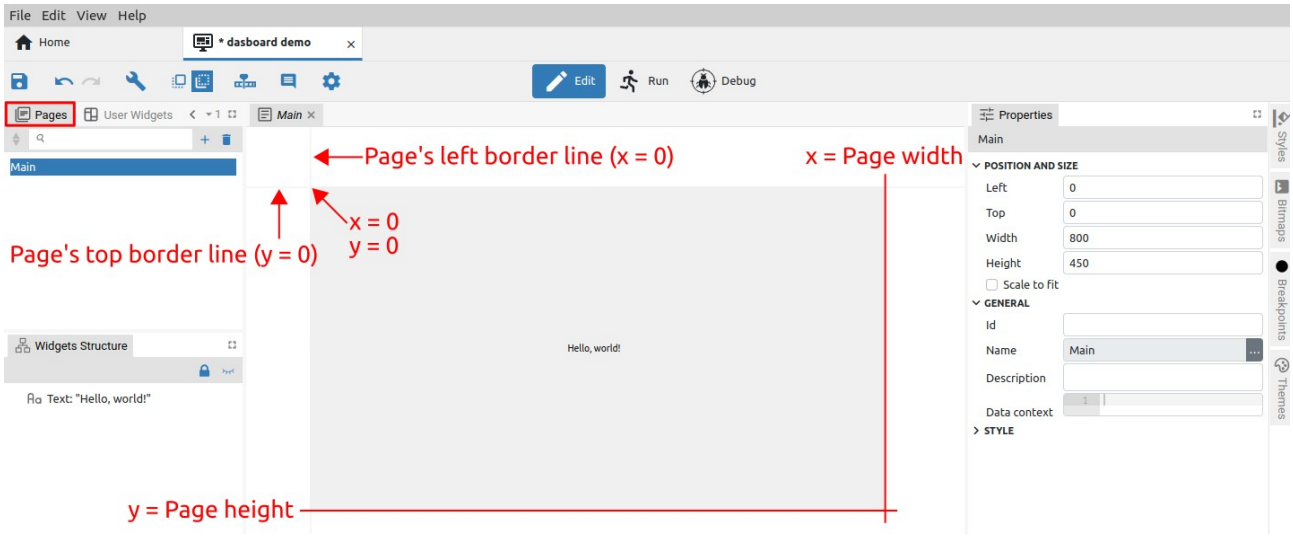


Fig. 39: Display of the page in the editor

P5.1.2. User Actions

The User Actions editor allows editing the selected Action from the User Actions panel (Fig. 40).

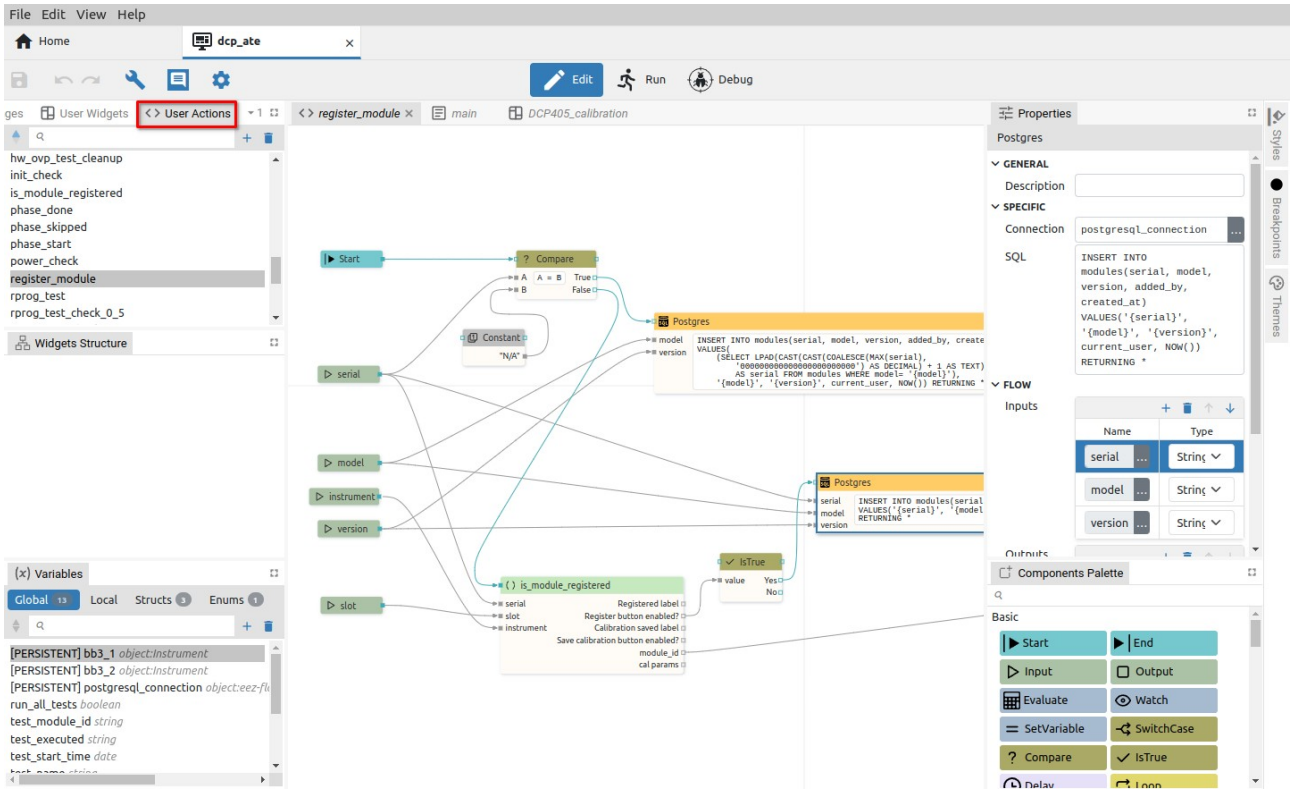


Fig. 40: User Actions editor page

P5.1.3. User Widgets

The User Widgets editor allows editing the selected Widget from the User Widgets panel (Fig. 41).

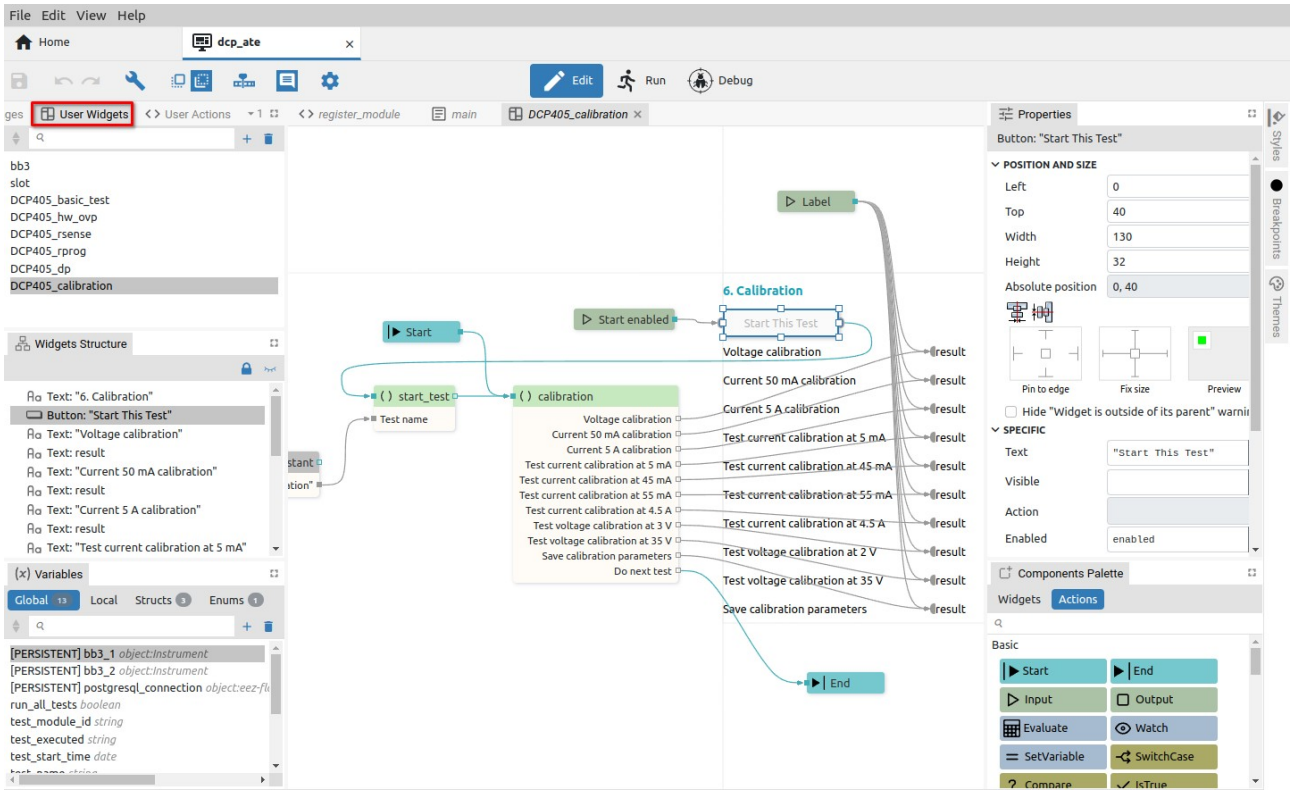


Fig. 41: User Widgets editor page

P5.1.4. 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.

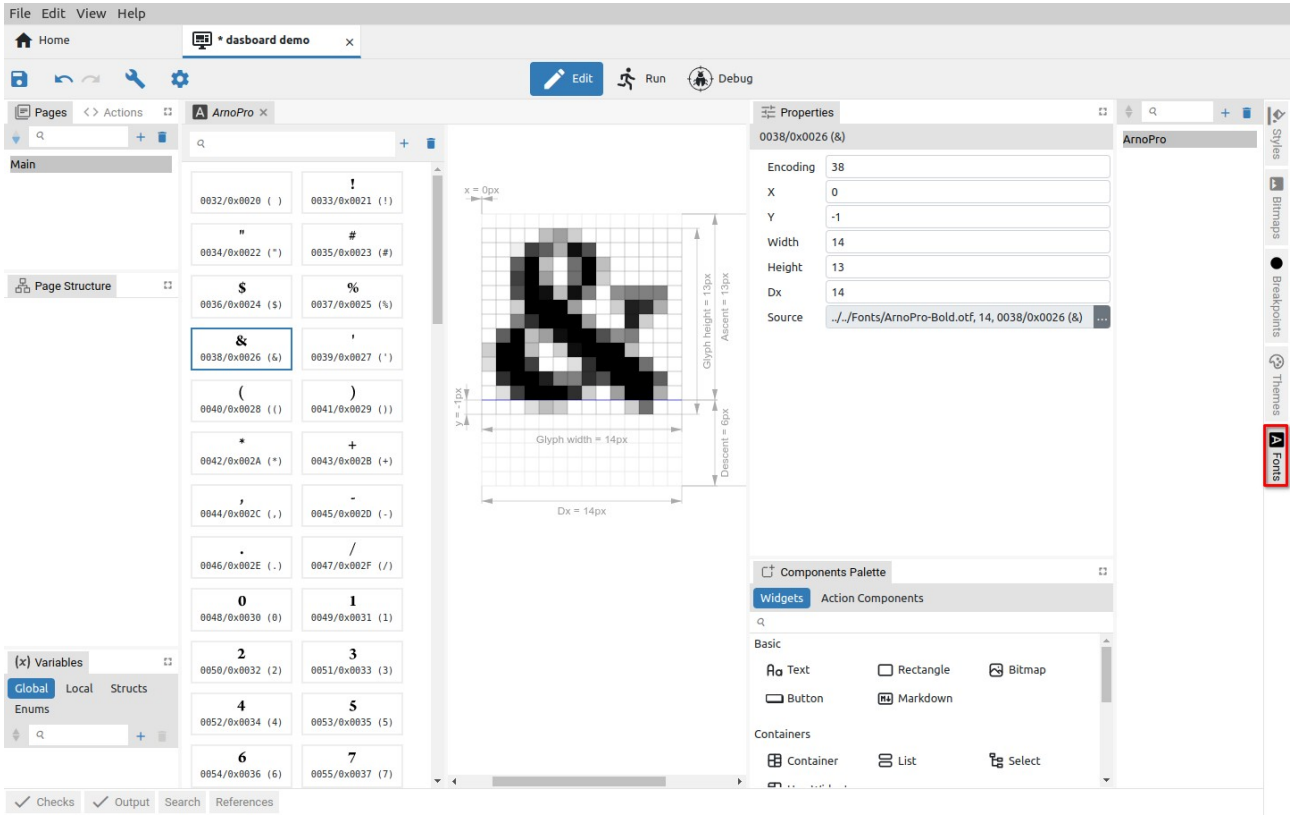


Fig. 42: Font editor page

P5.1.5. 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. 43).

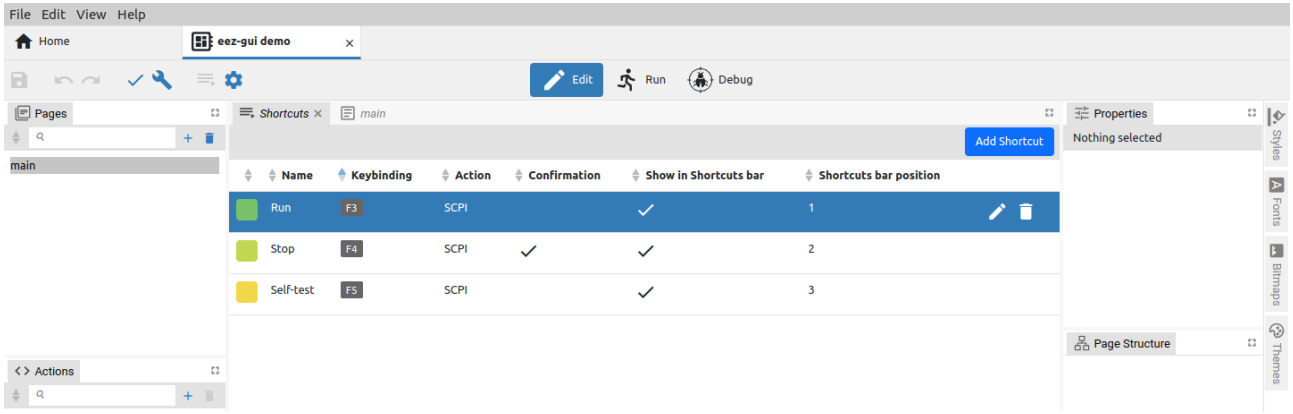


Fig. 43: Project shortcuts page

P5.1.6. MicroPython (EEZ BB3 only)

Opens the MicroPython text editor page (Fig. 44).

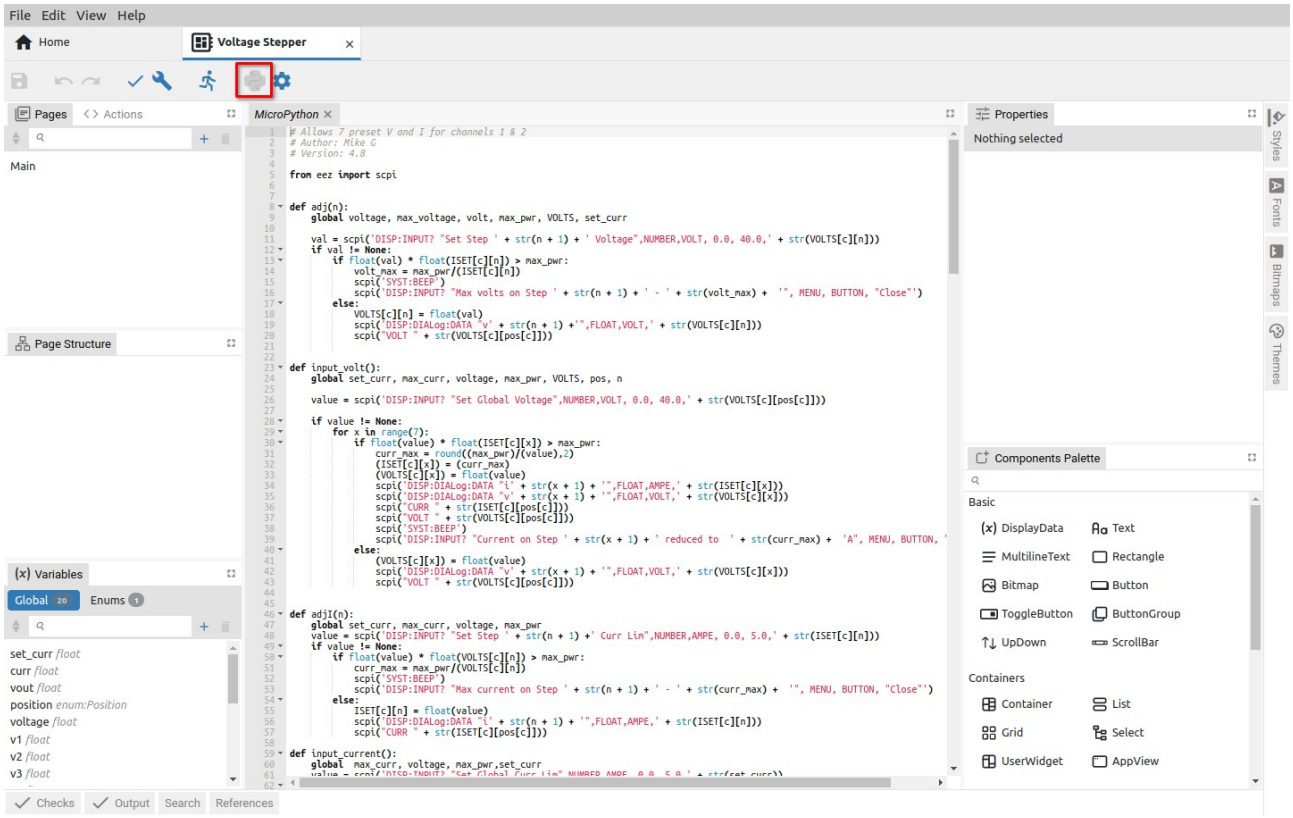


Fig. 44: MicroPython editor page

P5.1.7. 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.

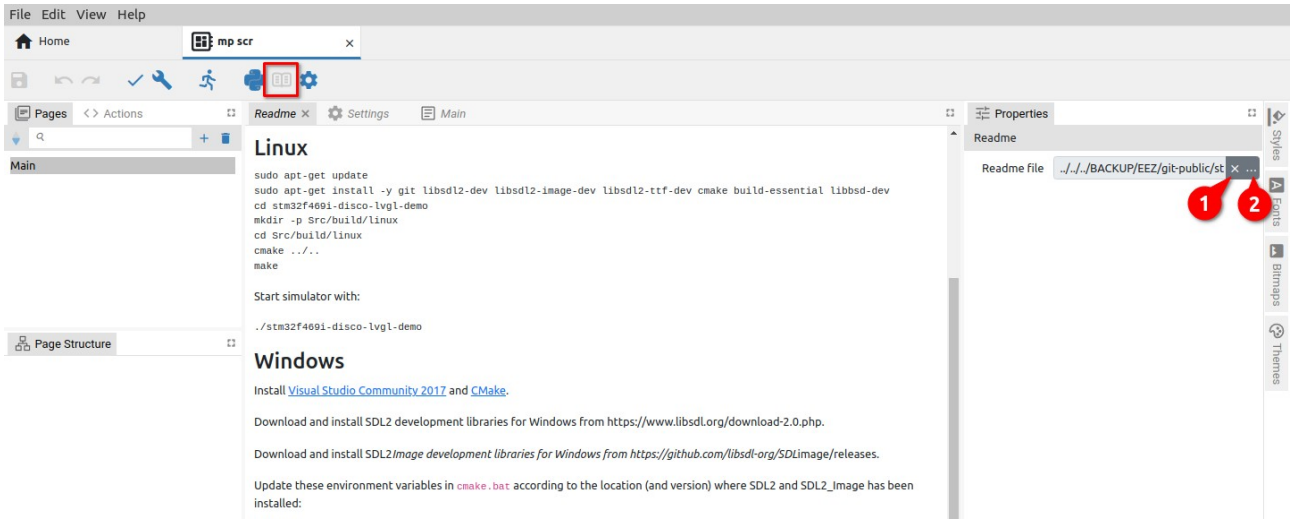


Fig. 45: Project Readme file

P5.1.8. Settings

The Settings page is used to edit the global parameters and features of the project (Fig. 46).

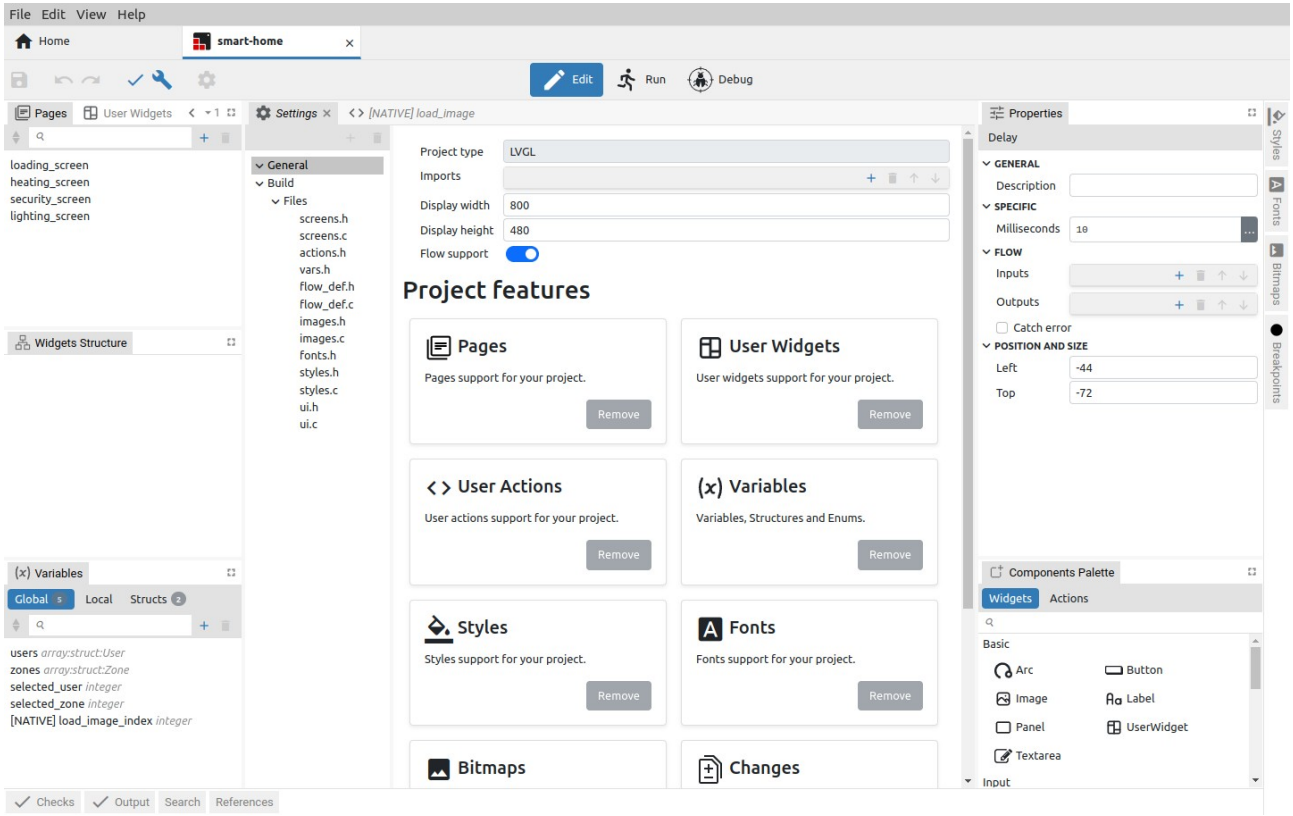


Fig. 46: Project settings page

P5.2. Viewers

P5.2.1. Page viewer

In *Run* mode, it is possible to see only the viewer of the currently active page (Fig. 47). In *Debug* mode, pages and Actions cannot be edited, so editors effectively become viewers (Fig. 48).

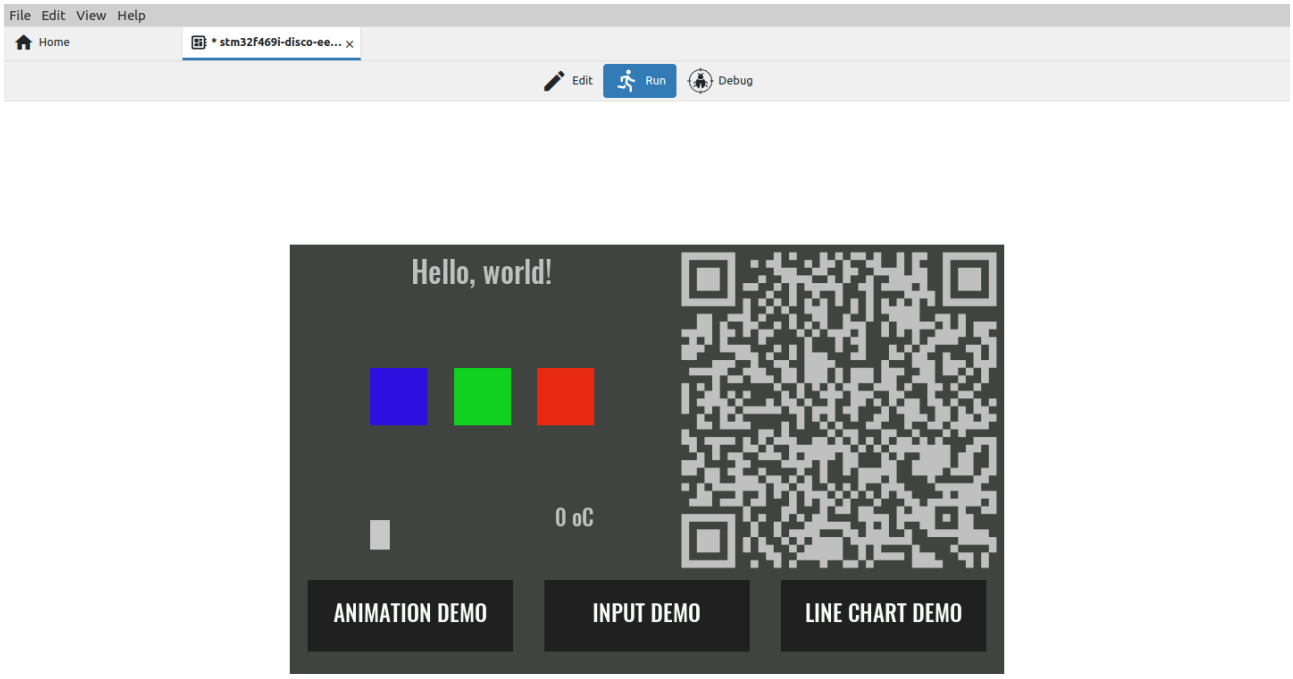


Fig. 47: Page view in Run mode

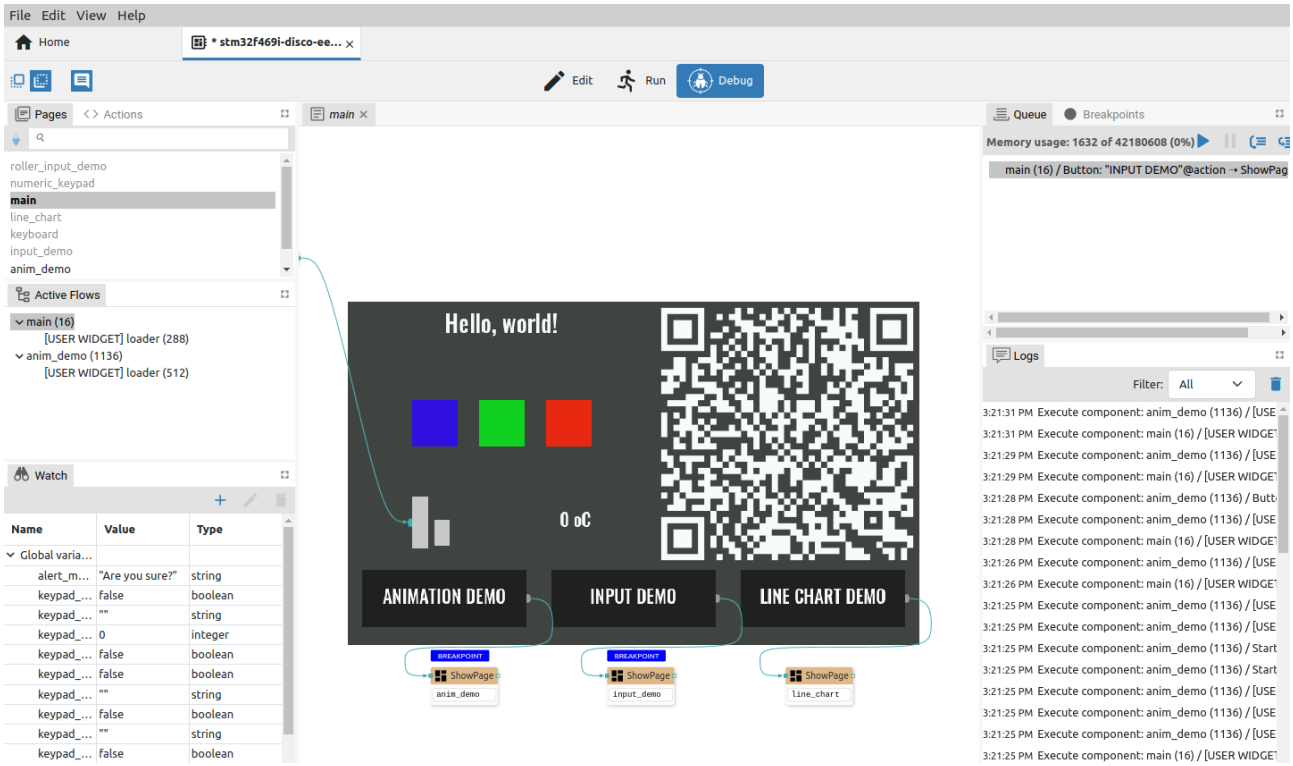


Fig. 48: Page view in Debug mode

P5.2.2. 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 break-points and see what values the component has on the inputs.

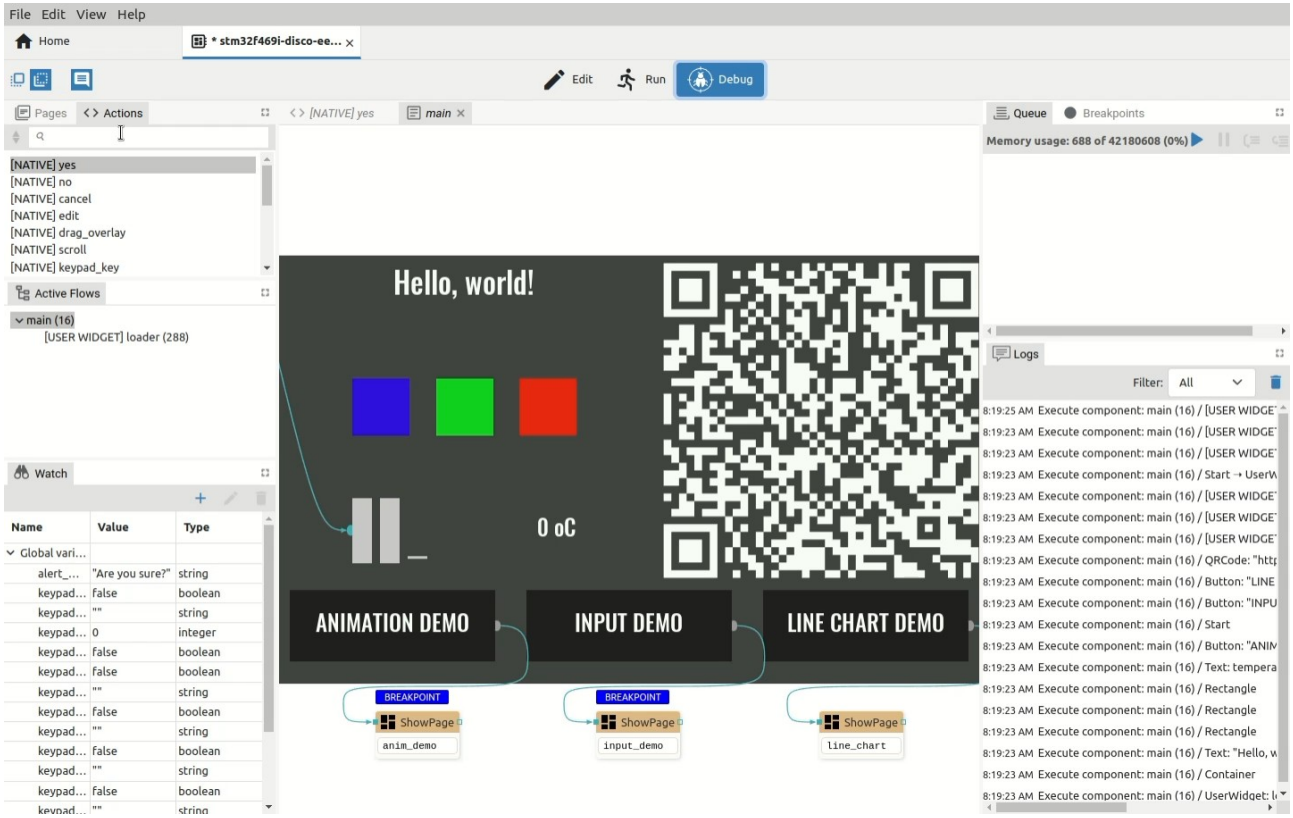


Fig. 49: Action viewer

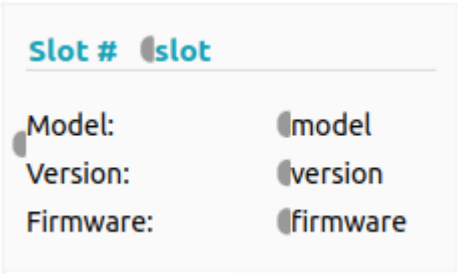
P6. EEZ Flow

P6.1. EEZ Flow basic concepts

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

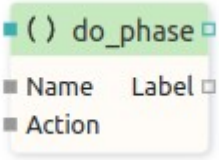
Flow can be an integral part of the page definition because the Widgets can be interactive and thus be an integral part of the Flow (connected to Actions and other Widgets by Flow lines). It is also possible to create a User Action for a Flow that does not have any graphic elements. The basic Flow elements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

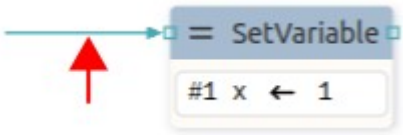


User Widget
User Widget is a convenient way to group part of a project that contains graphical elements for further reuse. *Input* and *Output* Actions are used to connect the User Widget with the rest of the Flow which are displayed as semicircles on the left or right side. A User Widget can be created 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.

P6.2. 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. 50).

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.

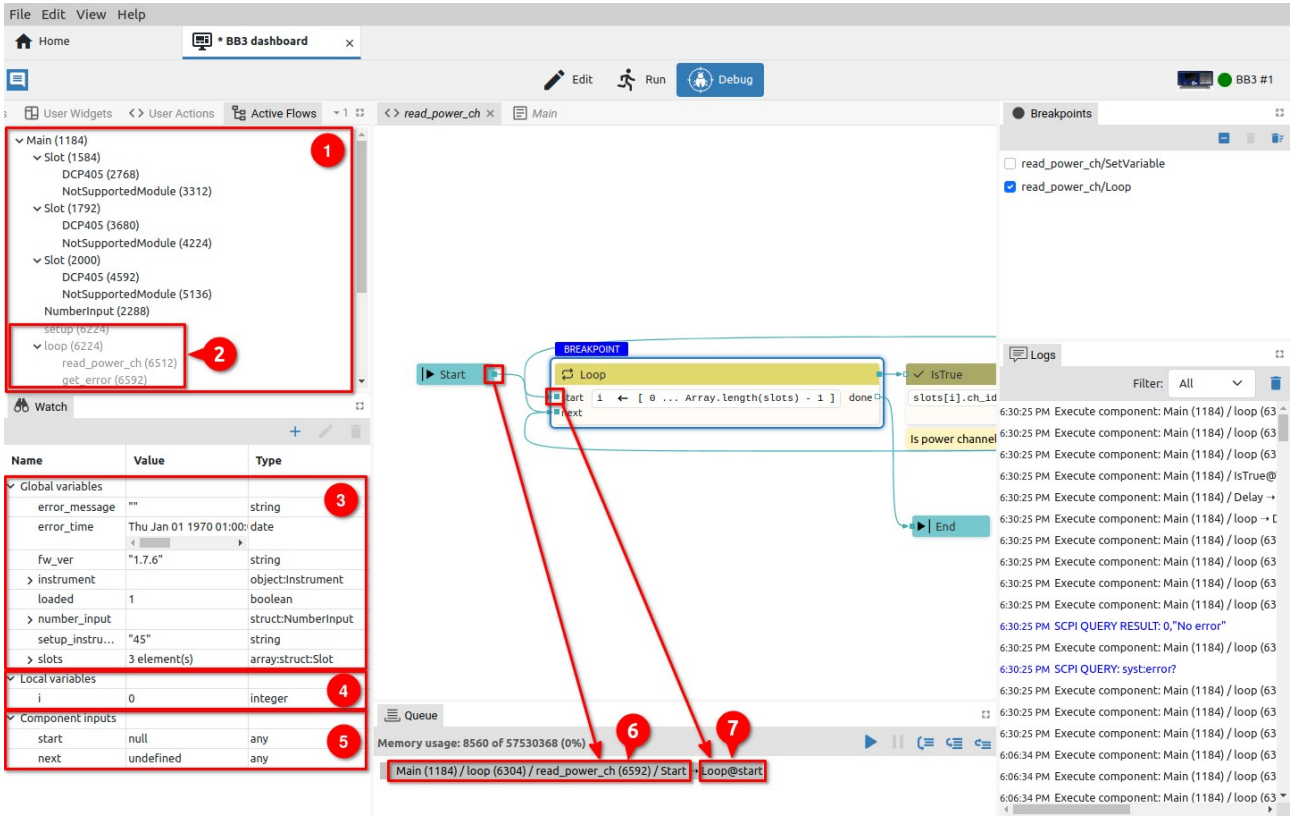


Fig. 50: Flow execution monitoring

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. 50 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.

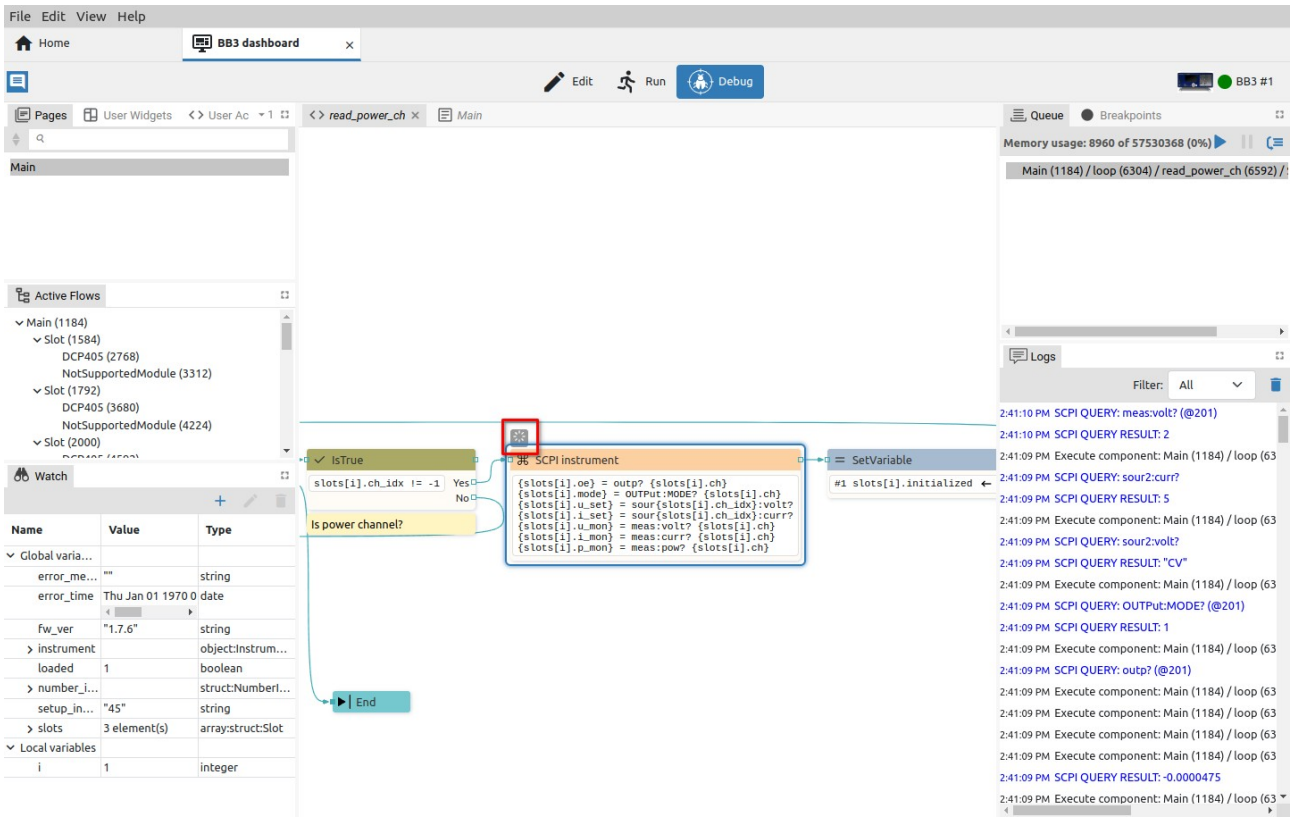


Fig. 51: Indication that the component preserves internal states

Components that preserve internal state, i.e. whose execution takes a long time, are also marked with a special icon in the debugger (Fig. 51). 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.

P6.3. Flow examples

Flow execution will depend on the way components are connected. In Fig. 52 shows four simple Flows that contain User Actions whose inputs are triggered in different ways. Fig. 53 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).

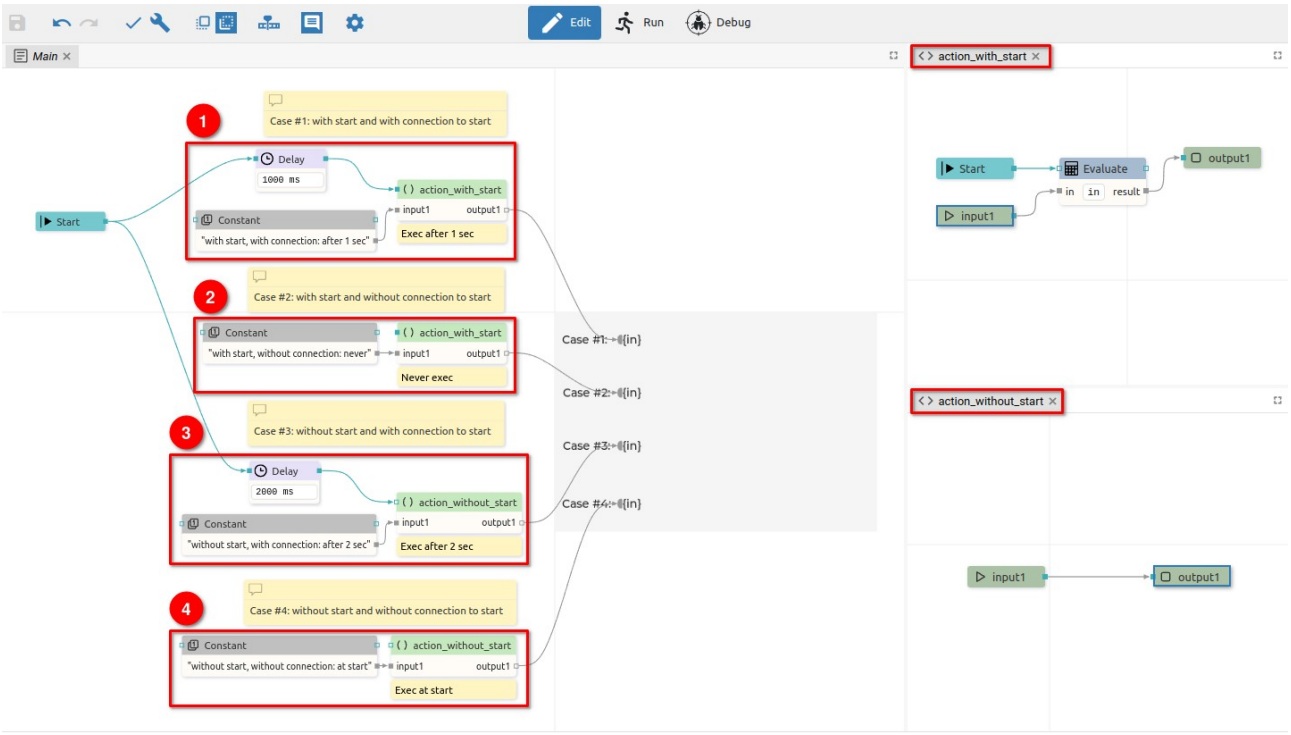


Fig. 52: Flow execution examples

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. 53: Execution results

P7. 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. 54

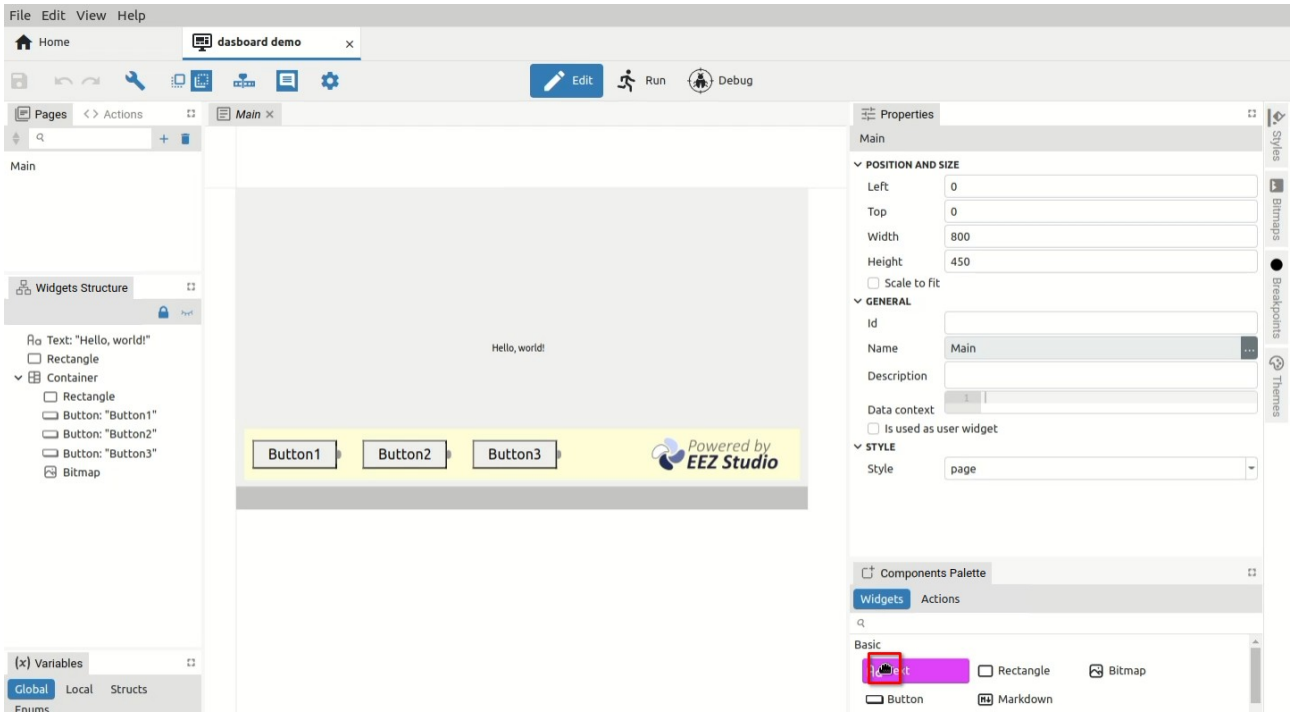


Fig. 54: Selection of Widget to add to the page

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. 55 where snap lines appear for horizontal or vertical centering.

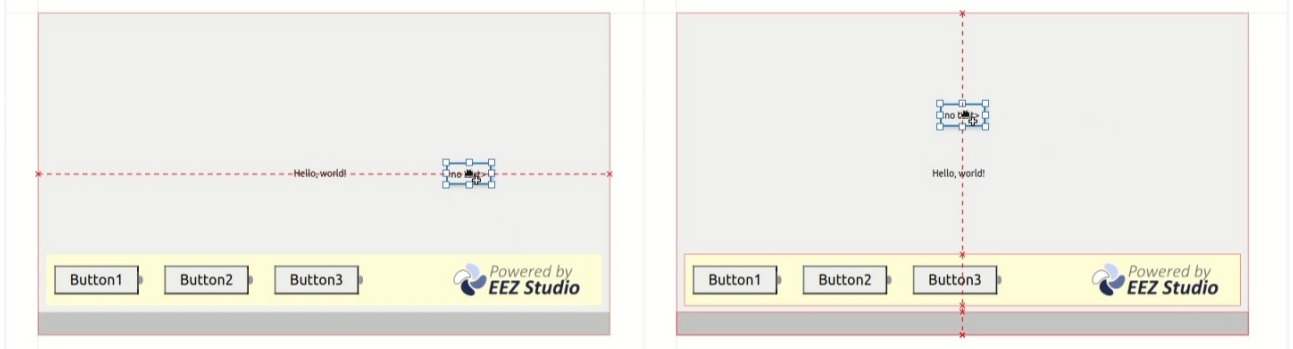


Fig. 55: Snap lines for centering in the page

Fig. 56: Snap lines for positioning versus other Widgets

Fig. 56 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.

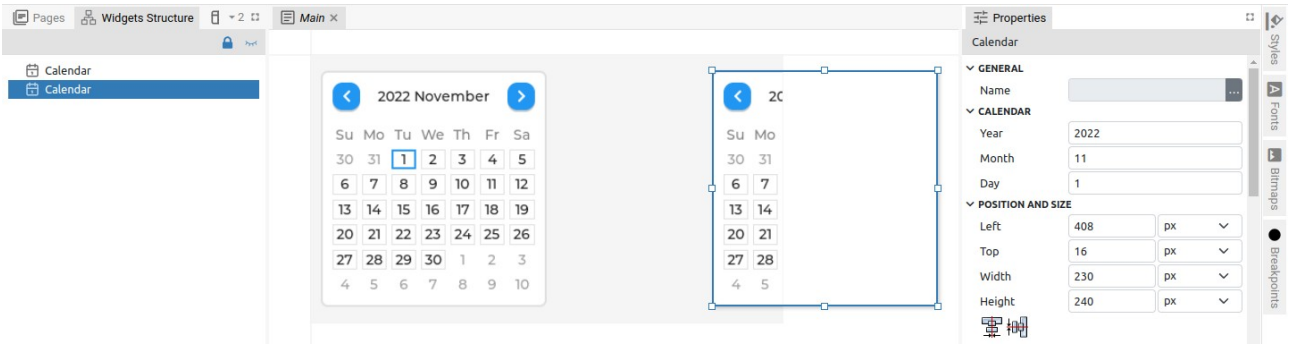


Fig. 57: Widget positioned outside the page

Please note that if the Widget in the *EEZ-GUI* and *LVGL* projects is set to protrude from the page (Fig. 57), 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. 58): 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. 59 when selecting the area that will include the Widgets we want to select.

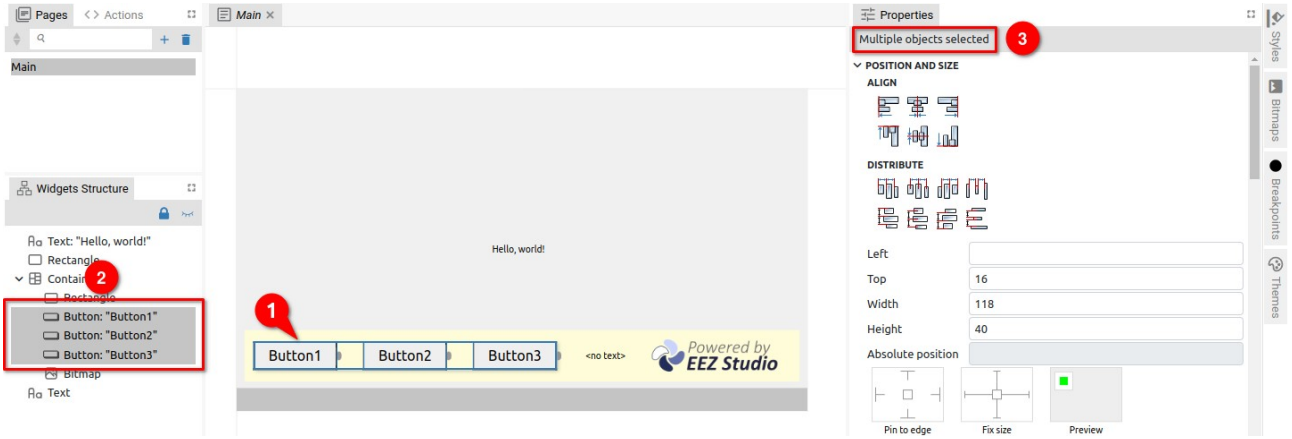


Fig. 58: Multiple Widgets selection

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).

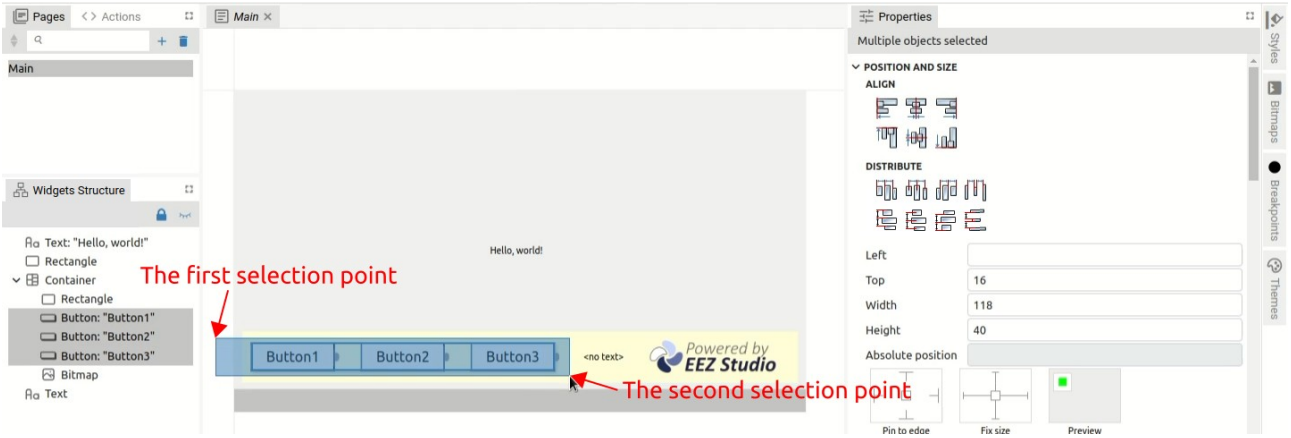


Fig. 59: "Rubber band" selection

As shown in Fig. 58, 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. 60.

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.

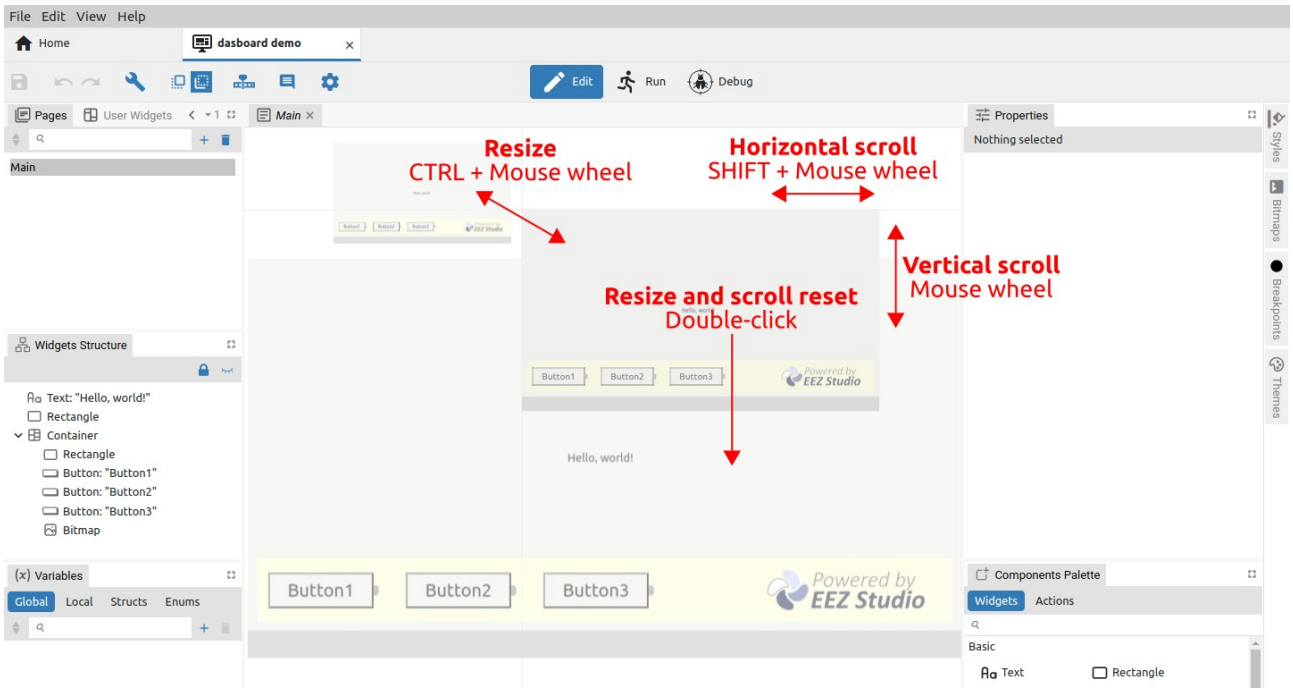


Fig. 60: Page resize and scroll

P7.1.1. Connecting Flow components

The connection (Flow line) between Flow components (Widgets and Actions) is used to define the flow of execution. In Fig. 55 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. 61: 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. 62 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.

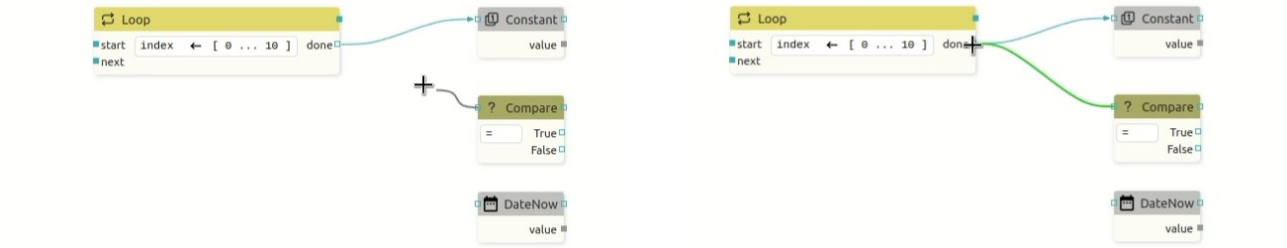


Fig. 62: Connecting the input to the output of the Widget

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. 63 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.

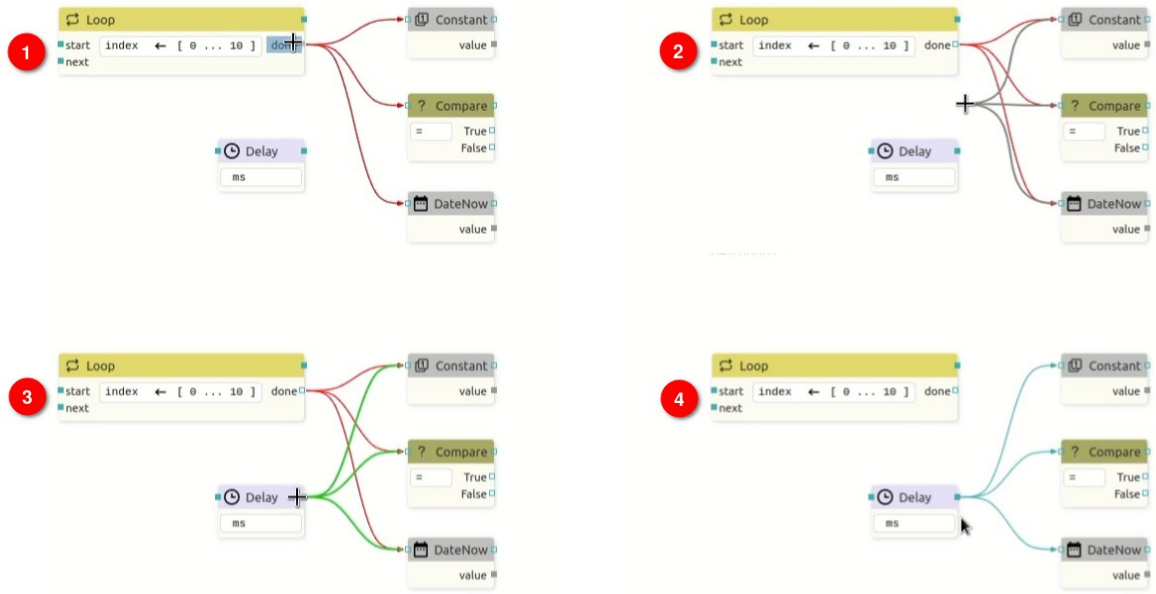


Fig. 63: Moving multiple connections

P7.1.2. 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).

P7.2. 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. 64)
- *LVGL* – Widgets from the open-source embedded graphics library LVGL. They can only be used in a project of type LVGL. (Fig. 65)

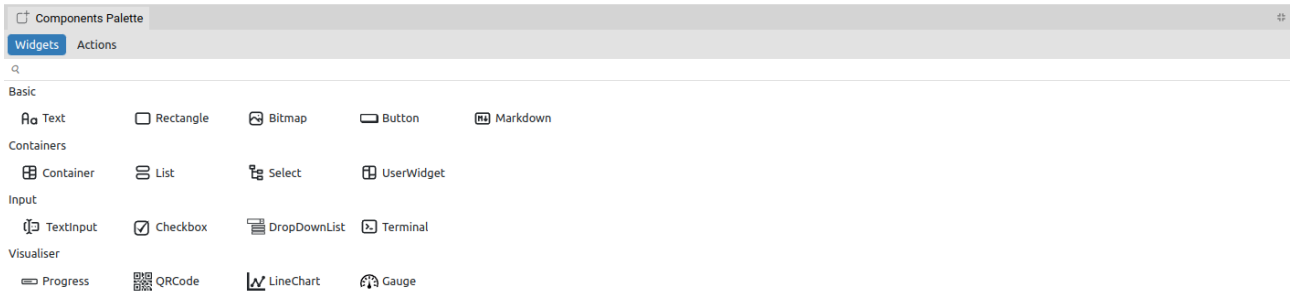


Fig. 64: Widgets palette for the EEZ-GUI project

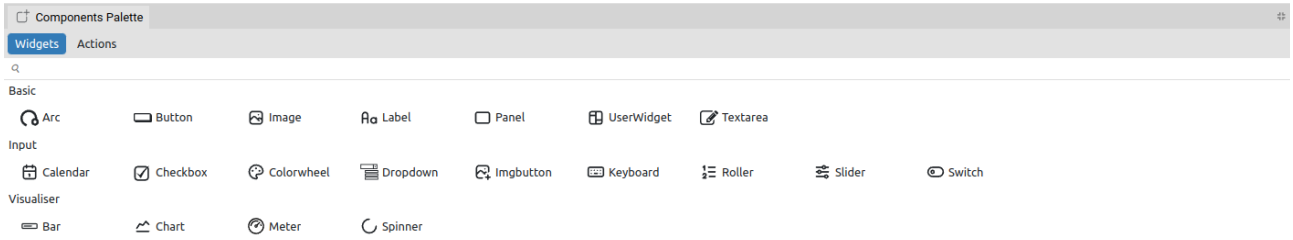


Fig. 65: Widgets palette for the LVGL project

P7.2.1. Widget component's items

Widget component items are shown in Fig. 66 and described below.

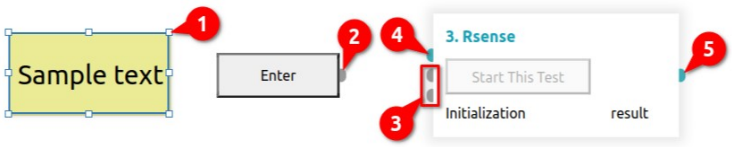


Fig. 66: Widget components items

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

P7.2.2. 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.

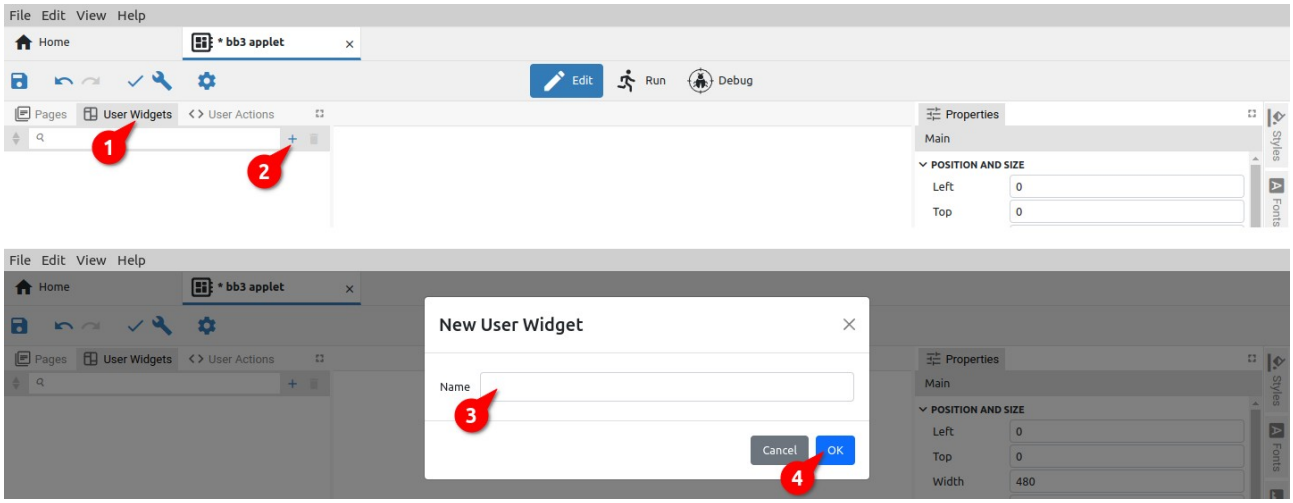


Fig. 67: Creating a new User Widget from the User Widgets *panel*

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. 67).

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).

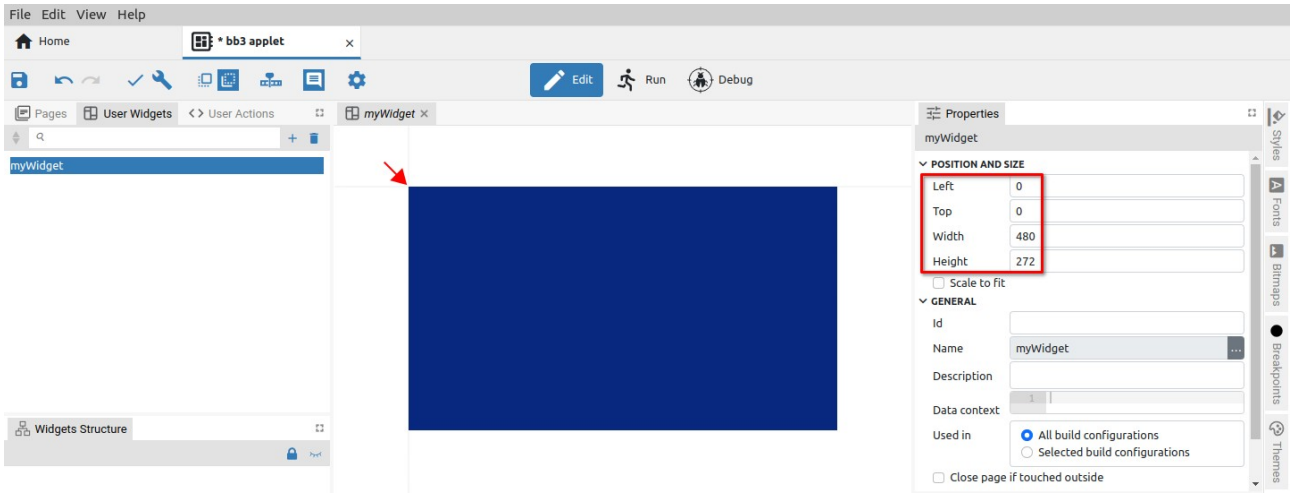


Fig. 68: Page editor of the newly created User Widget

User Widget can also be created by selecting one or more components on the currently displayed page (1) as shown on Fig. 69. By selecting the right-click menu option *Create User Widget*, a dialog box will appear as in the previous case (Fig. 67).

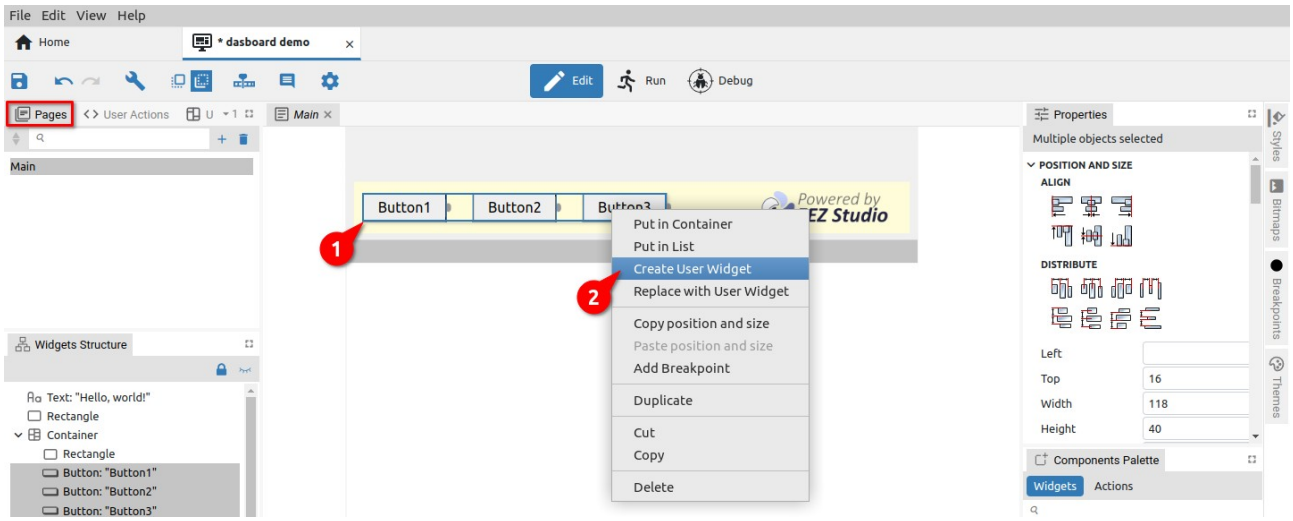


Fig. 69: Creating a new User Widget using the right-menu option

After successfully adding a new User Widget, it can be edited in the page editor (Fig. 70). 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.

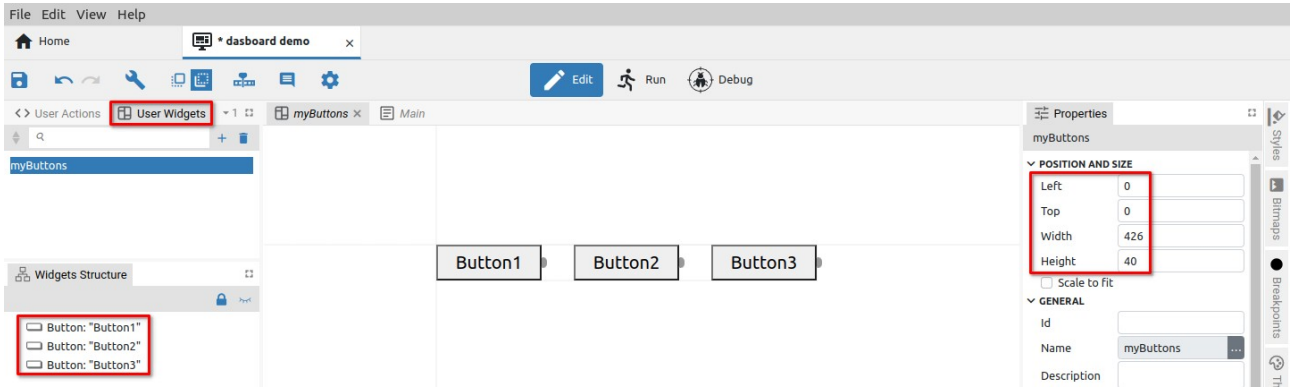


Fig. 70: The newly created User Widget in the editor

P7.3. 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. 71 shows the Actions for the Dashboard project, and Fig. 72 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. 71).

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. 72), from where they can be added to the project with drag and drop as any Action or Widget.

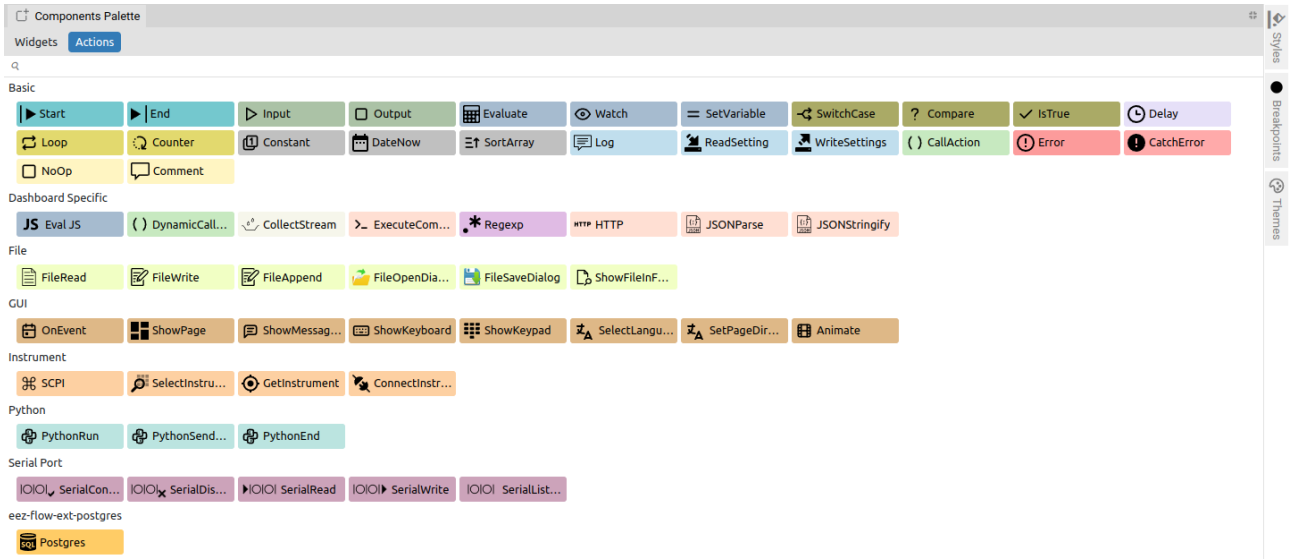


Fig. 71: Actions palette for the Dashboard project

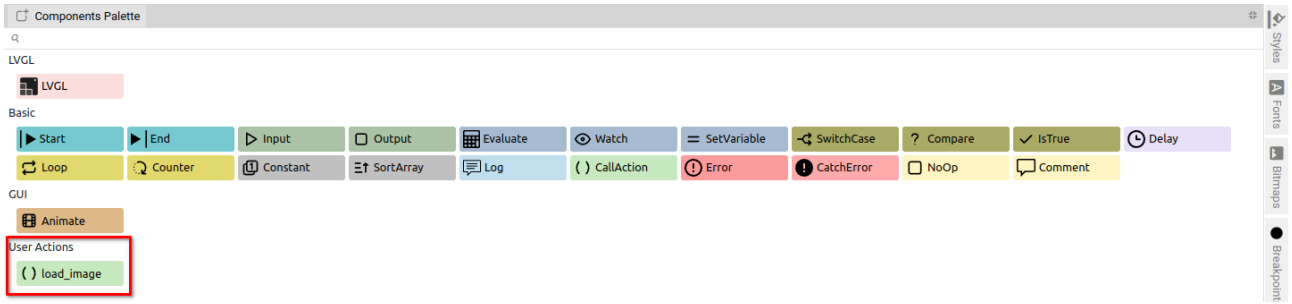


Fig. 72: Actions palette for the LVGL project

P7.3.1. Action component's items

Action component items are shown in Fig. 73 and described below.

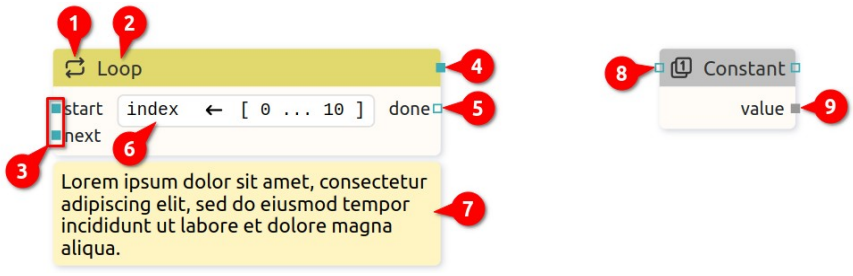


Fig. 73: Action components items

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

P7.3.2. 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.

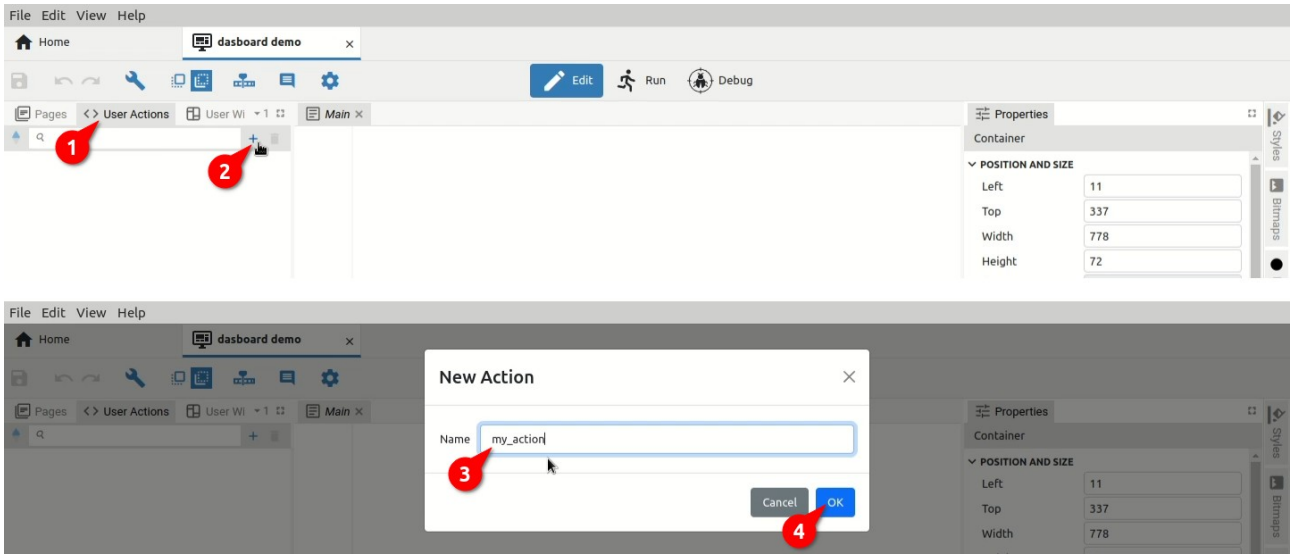


Fig. 74: Creating a new User Action

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.

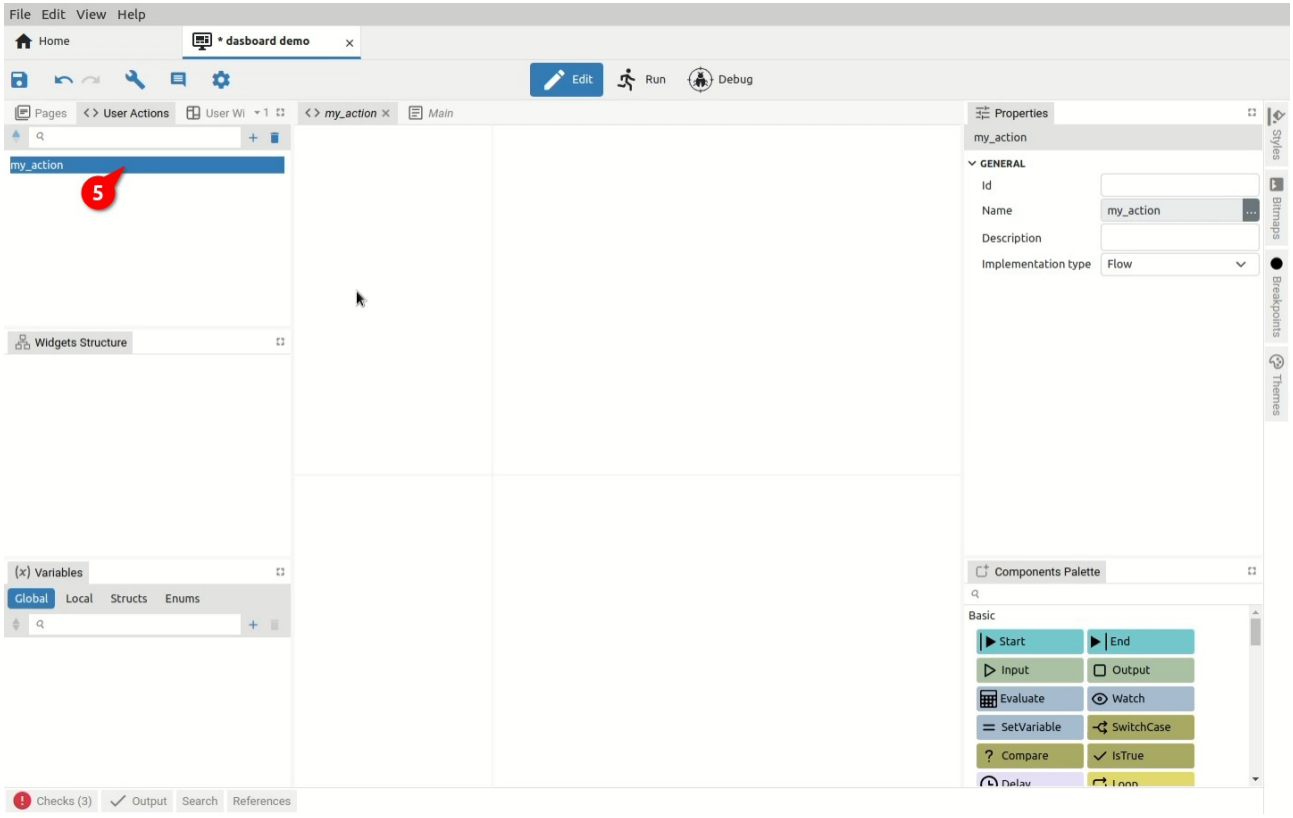


Fig. 75: Flow editor of the newly created User Action

Fig. 76 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.

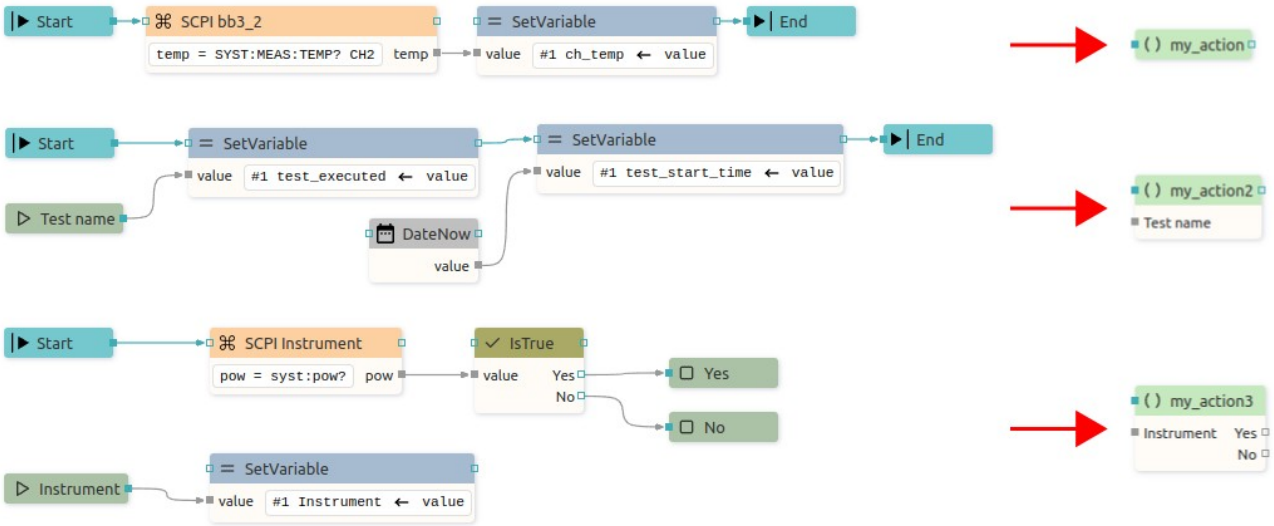


Fig. 76: User Action examples

P8. 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. 77), 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. 78, Fig. 79 and Fig. 80 shows *Properties* panels for different types of variables from different types of projects.

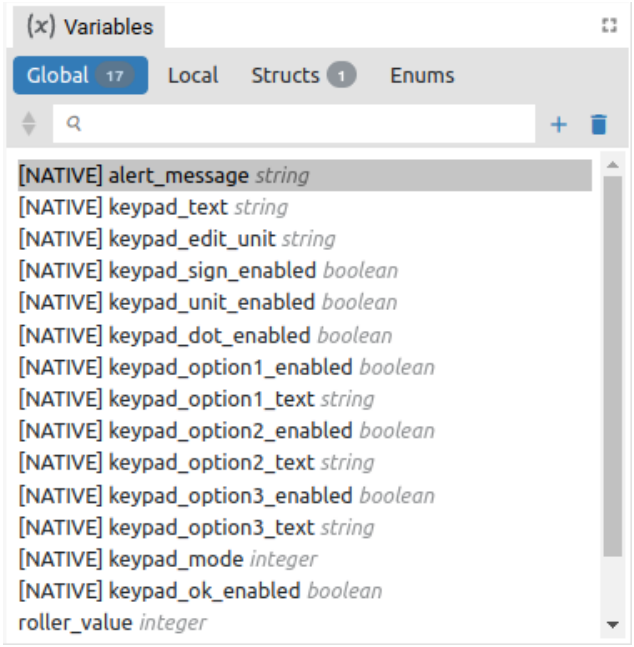


Fig. 77: Variables panel

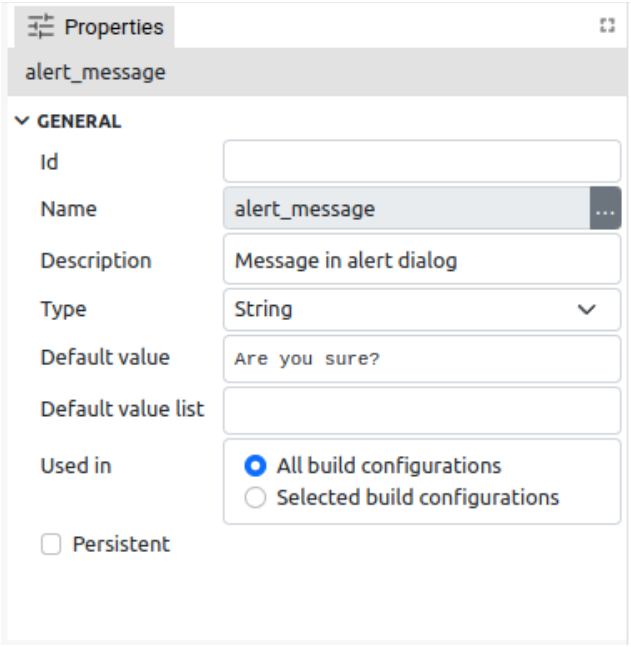


Fig. 78: Variable Properties panel (EEZ-GUI)

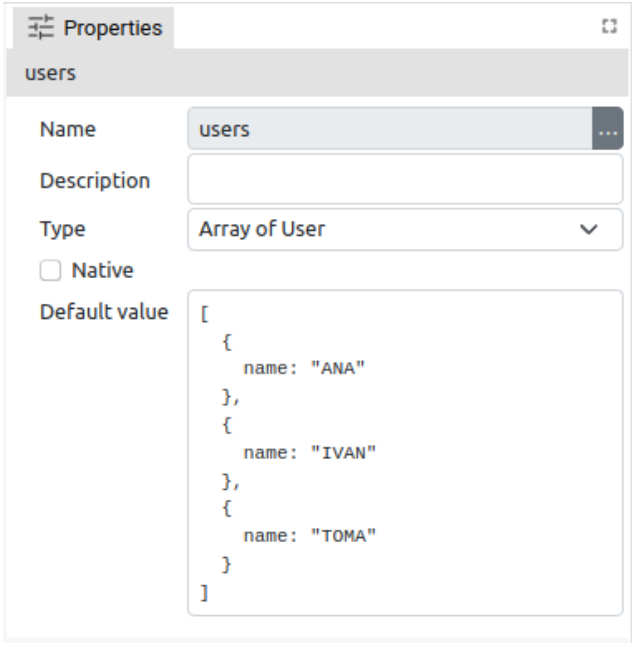


Fig. 79: Variable Properties panel (LVGL)

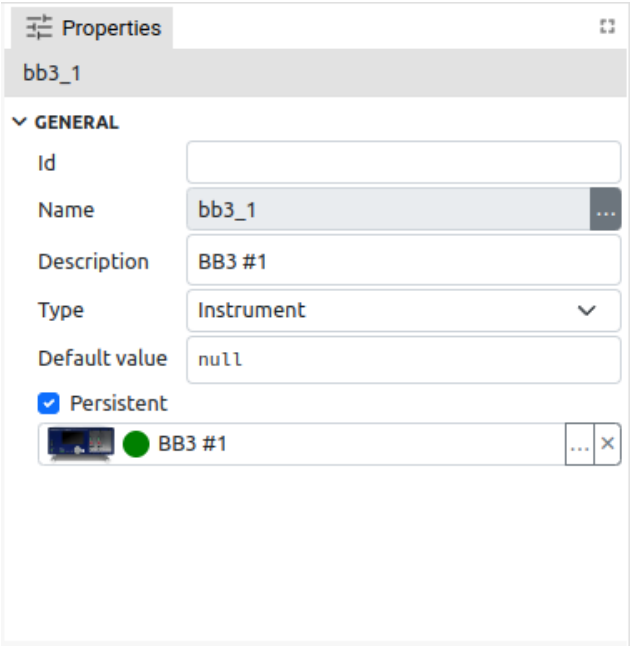


Fig. 80: Variable Properties panel (Dashboard)

Item	Description
Id	ID is *EEZ-GUI* project specific and is used in *Page*, *Action*, *Global Variable*, *Style*, *Font*, *Bitmap* and *Colors*. 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 *modular-psu-firmware.eez-project* 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. *Important: once the ID is set, it should not be changed, otherwise all BB3 scripts that depend on it should be rebuilt.*
Name	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.
Description	Optional field, contains a description of the variable.
Type	The type of data stored in variable. When adding a new variable, its suggested default value will depend on the selected type (0 for *Integer*, False for *Boolean*, etc.).
Native	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.
Default value	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"]`
Default value list	Only supported in *EEZ-GUI* project that does not have EEZ Flow enabled.
Used in	See *Configurations* in project *Settings*.
Persistent (Global variables only)	Stores the last value of the variable in the *.eez-runtime-settings* file, so that next time projects will have this value, instead of the default value. Supported only in Dashboard projects.

P8.1. Variables usage 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.

P8.2. Variable types

P8.2.1. Basic/Primitive types

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

P8.2.2. 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.

P8.2.3. 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.

P8.2.4. 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.

P8.2.5. Arrays

Array variable stores multiple data values.

P8.2.6. 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
Literal value	Example: `42`, `"Hello"`, `true`
Variable names	Example: `my_var`
Input names	Retrieves the data stored in data input using the name of that input. Example: `input_name`
Binary operator	Example: `my_integer_var + 1`
Logical operator	Example: `my_integer_var < 10`
Unary operator	Example: `-my_integer_var`

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

P8.2.7. Literals

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

P8.2.8. 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*
boolean	*integer*	*float*	*double*	*string*	*integer*	*err*
other_type	*err*	*err*	*err*	*err*	*err*	*err*

Subtraction -

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

Multiplication *

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

Division /

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

Remainder %

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

Left shift <<

arg1\arg2	integer	boolean	other_type
integer	*integer*	*integer*	*err*
boolean	*integer*	*integer*	*err*
other_type	*err*	*err*	*err*

Right shift >>

arg1\arg2	integer	boolean	other_type
integer	*integer*	*integer*	*err*
boolean	*integer*	*integer*	*err*
other_type	*err*	*err*	*err*

Binary AND &

arg1\arg2	integer	boolean	other_type
integer	*integer*	*integer*	*err*
boolean	*integer*	*integer*	*err*
other_type	*err*	*err*	*err*

Binary OR |

arg1\arg2	integer	boolean	other_type
integer	*integer*	*integer*	*err*
boolean	*integer*	*integer*	*err*
other_type	*err*	*err*	*err*

Binary XOR ^

arg1\arg2	integer	boolean	other_type
integer	*integer*	*integer*	*err*
boolean	*integer*	*integer*	*err*
other_type	*err*	*err*	*err*

P8.2.9. Logical operators

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

Type	Description
`==`	
`!=`	
`<`	
`>`	
`<=`	
`>=`	
`&&`	
`||`	

P8.2.10. Unary operators

Type	Description
`+`	
`-`	
`~`	
`!`	

P8.2.11. 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.

P8.3. Functions

P8.3.1. System

System.getTick

Retrieves the number of milliseconds that have elapsed since the flow execution was started.

Parameters
None

Return value
Value in milliseconds. Return type is `Integer`.

P8.3.2. Flow

Flow.index

Index of current element in the List and Grid widget. Check the description of these two widget for the more information.

Parameters

Name	Type	Description
`index`	`Integer`	???

Return value
Element index. Return type is `Integer`.

Flow.isPageActive

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
True if page is active, False if page is not active. Return type is `Boolean`.

Flow.pageTimelinePosition

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
Timeline position. Return type is `Boolean`.

Flow.makeValue

Creates a new value of type Struct.

Parameters

Name	Type	Description
`structName`	`String`	Structure name.
`value`	`JSON`	Structure name.

Return value
Created struct value. Return type is `Struct`.

Flow.makeArrayValue

Creates a new value of type array.

Parameters

Name	Type	Description
`value`	`JSON`	Array value.

Return value
Created array value. Return type is `Array`.

Flow.languages

Retrieves a list of languages defined in multi-language project as array of strings.

Parameters
None

Return value
Array of languages. Return type is `Array:string`.

Flow.translate

Translate text resource ID, same as `T"textResourceID"`.

Parameters

Name	Type	Description
`textResourceID`	`String`	Text resource ID.

Return value
Translated string. Return type is `String`.

Flow.parseInteger

Parse integer value given as string.

Parameters

Name	Type	Description
`str`	`String`	Input string.

Return value
Parsed integer value. Return type is `Integer`.

Flow.parseFloat

Parse float value given as string.

Parameters

Name	Type	Description
`str`	`String`	Input string.

Return value
Parsed float value. Return type is `Float`.

Flow.parseDouble

Parse double value give as string.

Parameters

Name	Type	Description
`str`	`String`	Input string.

Return value
Parsed double value. Return type is `Double`.

P8.3.3. Date

Date.now

Returns current date.

Parameters
None

Return value
Current datetime. Return type is `Now`.

Date.toString

Converts given date to string.

Parameters

Name	Type	Description
str `date`	`Date`	Input date.

Return value
Date string. Return type is `String`.

Date.toLocaleString

Converts given date to locale string.

Parameters

Name	Type	Description
str `date`	`Date`	Input date.

Return value
Date string. Return type is `String`.

Date.fromString

Converts string to date.

Parameters

Name	Type	Description
`dateStr`	`String`	Input string.

Return value
Date. Return type is `Date`.

Date.getYear

Get year from date.

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Year. Return type is `Integer`.

Date.getMonth

Get month from date (1 to 12).

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Month. Return type is `Integer`.

Date.getDay

Get day of the month from date (1 to 31).

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Day. Return type is `Integer`.

Date.getHours

Get hours from date (0 to 23).

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Hours. Return type is `Integer`.

Date.getMinutes

Get minutes from date (0 to 59).

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Minutes. Return type is `Integer`.

Date.getSeconds

Get seconds from date (0 to 59).

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Seconds. Return type is `Integer`.

Date.getMilliseconds

Get milliseconds from date (0 to 999).

Parameters

Name	Type	Description
`date`	`Date`	Input date.

Return value
Milliseconds. Return type is `Integer`.

Date.make

Make a date from arguments.

Parameters

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

Return value
Constructed date. Return type is `Date`.

P8.3.4. Math

Math.sin

Returns the sine of a number in radians.

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number representing an angle in radians.

Return value
The sine of x, between -1 and 1, inclusive. Return type is `Float|Double`.

Math.cos

Returns the cosine of a number in radians.

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number representing an angle in radians.

Return value
The cosine of x, between -1 and 1, inclusive. Return type is `Float|Double`.

Math.pow

Returns the value of a base raised to a power.

Parameters

Name	Type	Description
`base`	`Integer|Float|Double`	Year
`exponent`	`Integer|Float|Double`	Month

Return value
A number representing base taken to the power of exponent. Return type is `Float|Double`.

Math.log

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

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number greater than or equal to 0.

Return value
The natural logarithm (base e) of x. Return type is `Float|Double`.

Math.log10

Returns the base 10 logarithm of a number.

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number greater than or equal to 0.

Return value
The base 10 logarithm of x. Return type is `Float|Double`.

Math.abs

Returns the absolute value of a number.

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number.

Return value
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. Return type is `Integer|Float|Double`.

Math.floor

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

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number.

Return value
The largest integer smaller than or equal to x. It's the same value as *-Math.ceil(-x)*. Return type is `Integer|Float|Double`.

Math.ceil

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

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number.

Return value
The smallest integer greater than or equal to x. It's the same value as *-Math.floor(-x)*. Return type is `Integer|Float|Double`.

Math.round

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

Parameters

Name	Type	Description
`x`	`Integer|Float|Double`	A number.

Return value

The value of x rounded to the nearest integer. Return type is `Integer|Float|Double`.

Math.min

Returns the smallest of the numbers given as input parameters.

Parameters

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

Return value
The smallest of the given numbers. Return type is `Integer|Float|Double`.

Math.max

Returns the largest of the numbers given as input parameters.

Parameters

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

Return value
The largest of the given numbers. Return type is `Integer|Float|Double`.

P8.3.5. String

String.length

Returns the length of the string.

Parameters

Name	Type	Description
`string`	`String`	A string.

Return value
The length of a string. Return type is `Integer`.

String.substring

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

Return value
A new string containing the specified part of the given string. Return type is `String`.

String.find

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

Parameters

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

Return value
The index of the first occurrence of substring found, or -1 if not found. Return type is `String`.

String.padStart

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

Parameters

Name	Type	Description
`string`	`String`	A string.
`targetLength`	`Integer`	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.
`padString`	`String`	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
A String of the specified targetLength with padString applied from the start. Return type is `String`.

String.split

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

Return value
An Array of strings, split at each point where the separator occurs in the given string. Return type is `Array:string`.

P8.3.6. Array

Array.length

The number of elements in given array.

Parameters

Name	Type	Description
`array`	`Array`	An array.

Return value
The length of an array. Return type is `Integer`.

Array.slice

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	Optional	Type	Description
`array`		`Array`	An array.
`start`		`Integer`	Zero-based index at which to start extraction.
`end`	Yes	`Integer`	Zero-based index at which to end extraction.

Return value
A new array containing the extracted elements. Return type is `Array`.

Array.allocate

Creates a new array of given size.

Parameters

Name	Type	Description
`size`	`Array`	A size number.

Return value
A new array. Return type is `Array`.

Array.append

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	Type	Description
`array`	`Array`	An array.
`value`	`Any`	Element value to be appended.

Return value
A new array with appended element. Return type is `Array`.

Array.insert

Inserts an element to an existing array at given position and returns a new array. The original array will not be modified.

Parameters

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

Return value
A new array with inserted element. Return type is `Array`.

Array.remove

Removes from an existing array an element at given position and returns a new array. The original array will not be modified.

Parameters

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

Return value
A new array with element removed. Return type is `Array`.

Array.clone

Deep clone of the array.

Parameters

Name	Type	Description
`array`	`Array`	An array.

Return value
A new array. Return type is `Array`.

P8.3.7. LVGL

LVGL.MeterTickIndex

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

Parameters
None

Return value
Index number. Return type is integer.

P8.4. 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. 81). Expressions can be entered manually or using the *Expression Builder*.

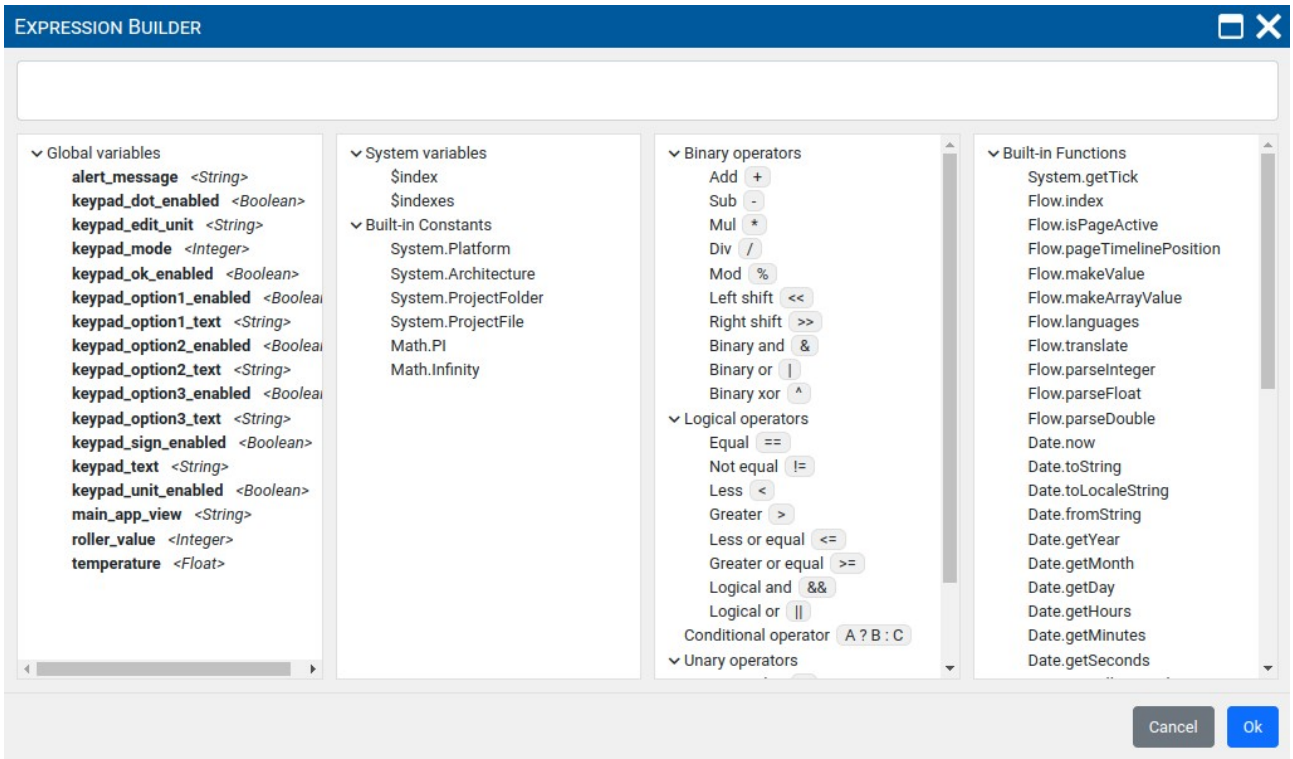


Fig. 81: Expression builder

EEZ Studio Instrument

Table of Contents

I1. Home page instrument sections

The top of the home page contains general options (1) for working with instruments (Fig. 1). Instrument specific *History*, *Shortcuts and Groups* and *Notebooks* options can also be accessed through the I*nstruments 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. 2).

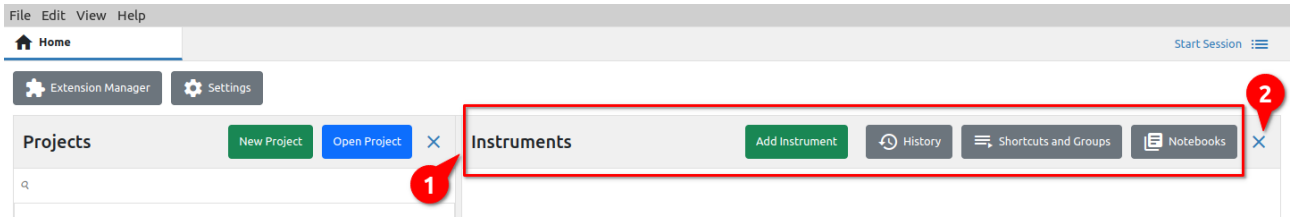


Fig. 1: Home page instrument options

Fig. 2: Home page "Show instruments" option

I1.1. 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.

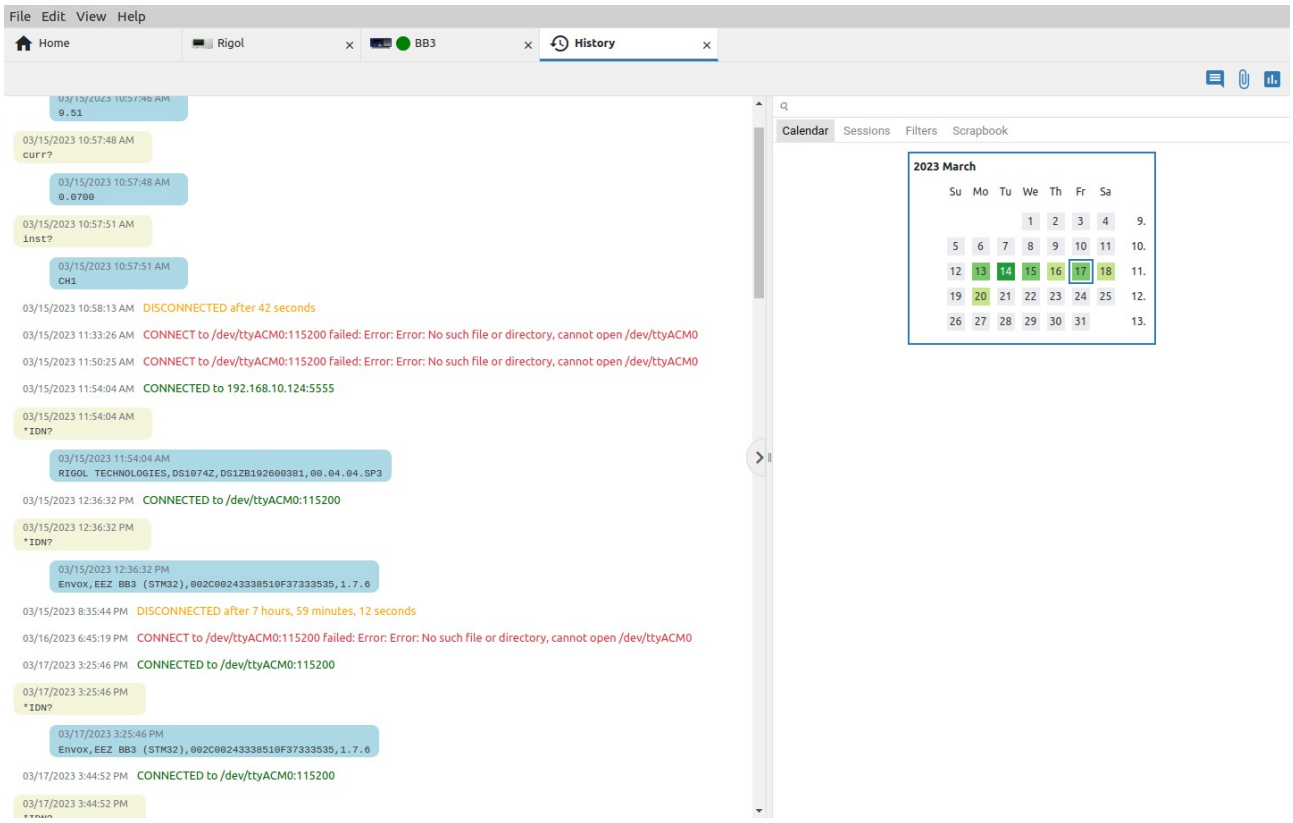
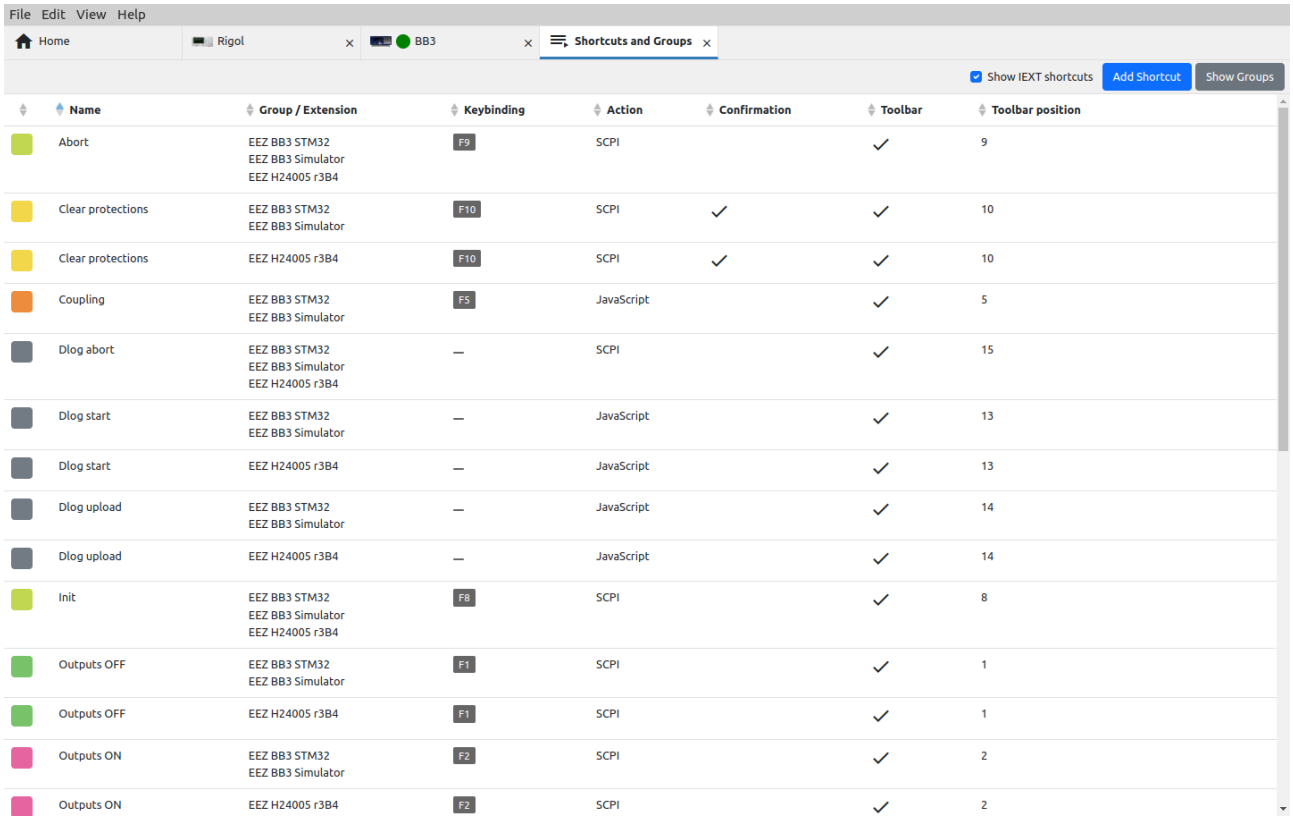


Fig. 3: Instruments History view

I1.2. 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.



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. 4: Instruments Shortcuts and Groups view

I1.3. 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.

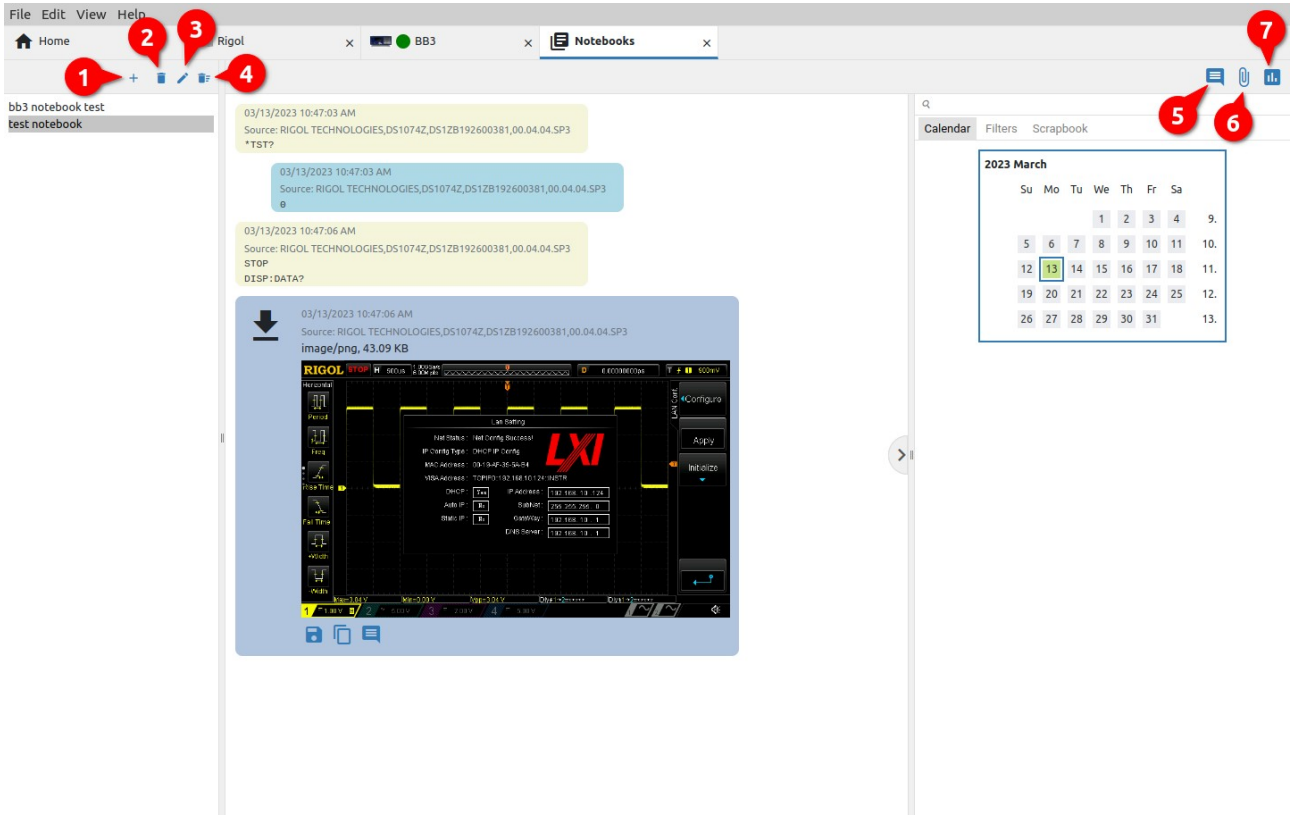


Fig. 5: Instrument Notebooks view

#	Option	Description
1	*Add / Import notebook*	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. 7: (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.

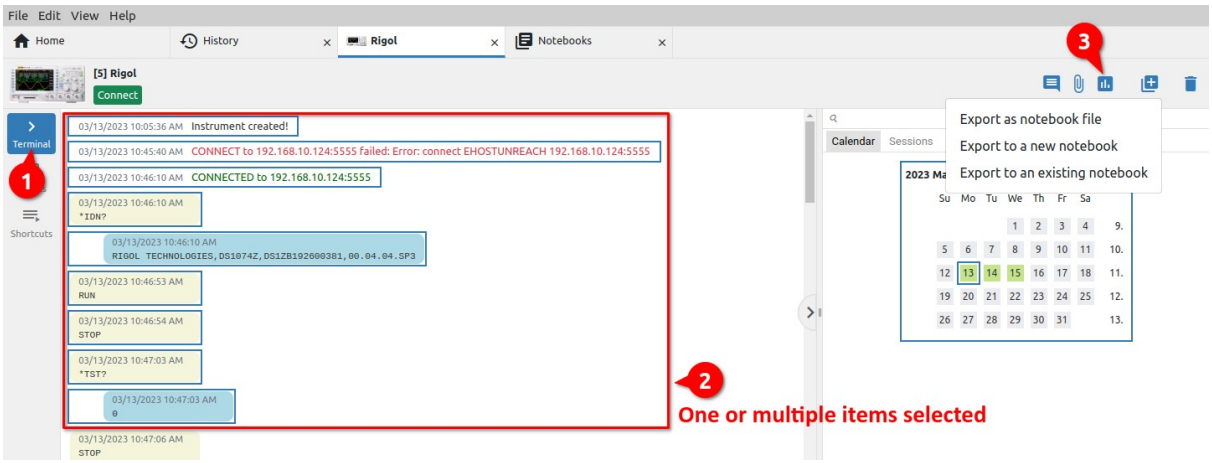


Fig. 6: Adding items to the notebook

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. 7) that have been removed from the list and offers the possibility to restore (return to the list) or permanently delete the notebook.

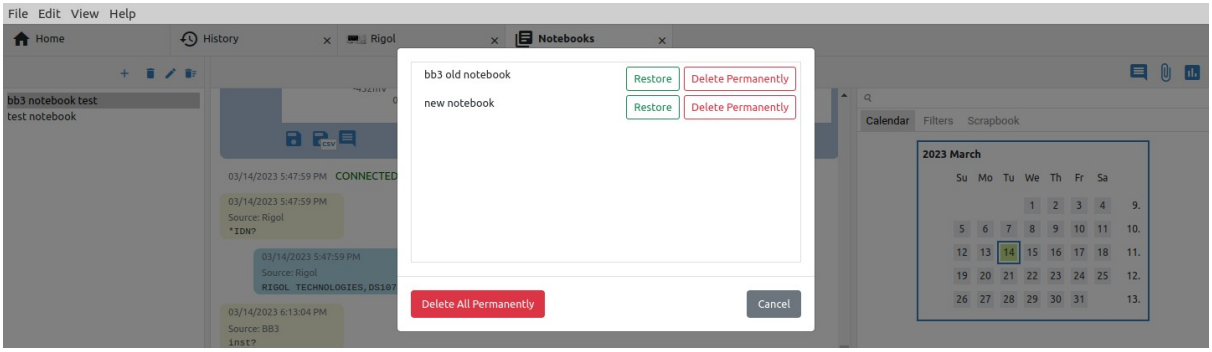


Fig. 7: Deleted notebooks

5	*Add note*	Adding a note to the notebook (Fig. 8). The number of notes is not limited and the last added note will appear at the bottom of the notebook (Fig. 9).

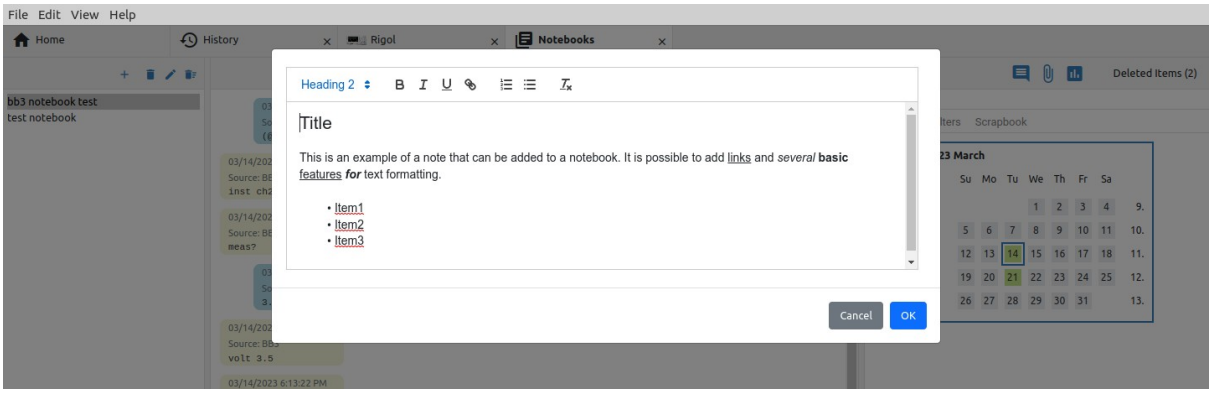


Fig. 8: Adding a new note to the notebook

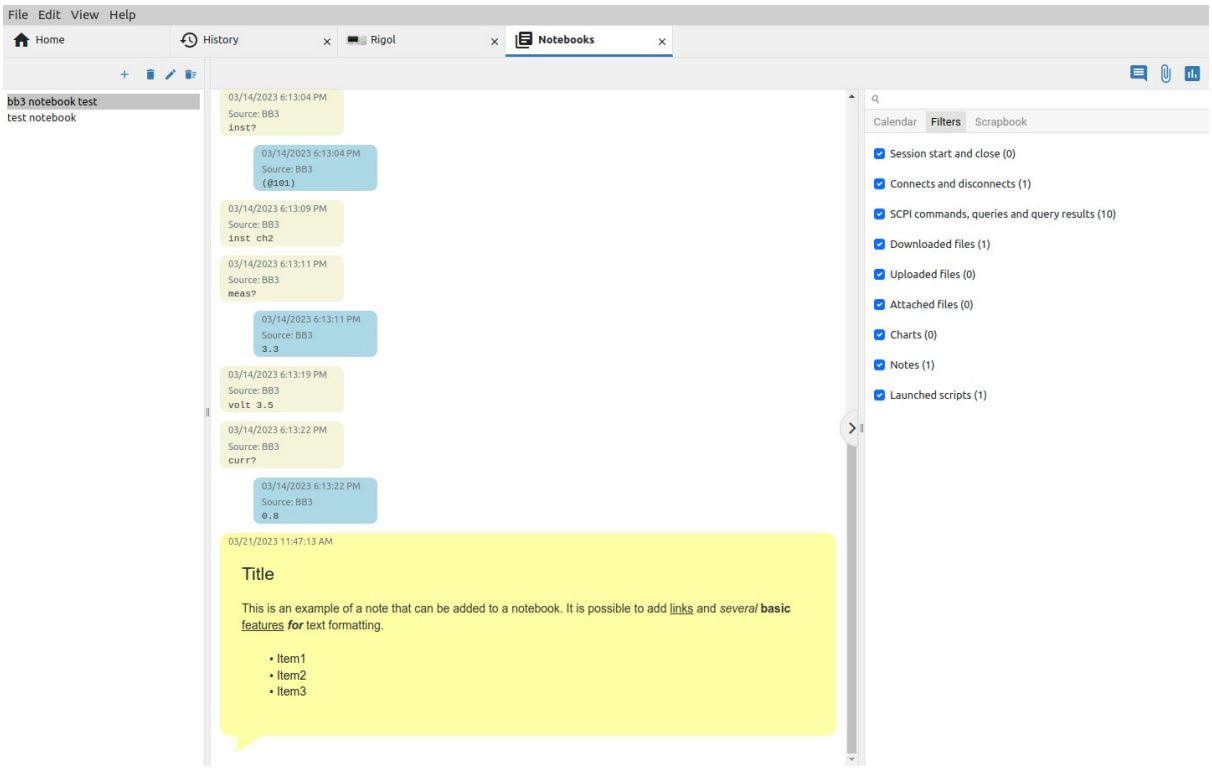


Fig. 9: Newly added note in the notebook

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. 10).

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.

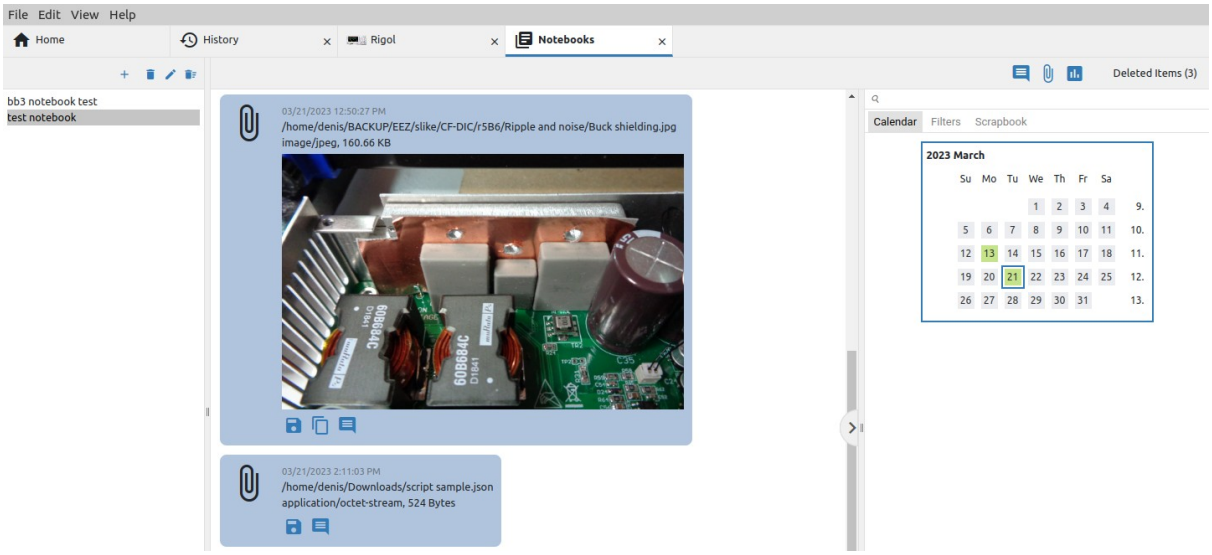


Fig. 10: Files imported into the notebook

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. 11.

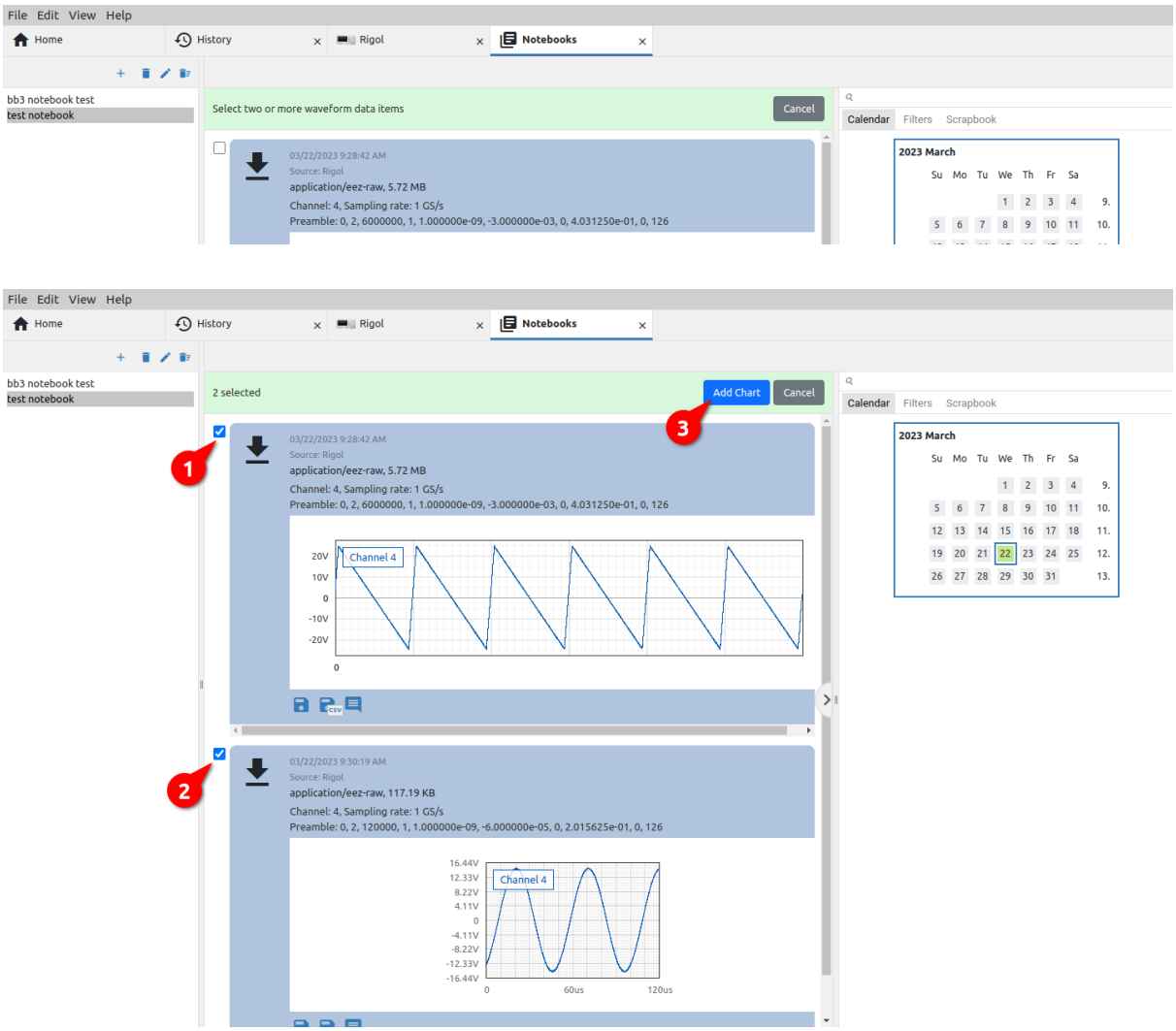


Fig. 11: A selection of graphs to add to the notebook

A successfully created graph will appear at the end of the notebook and will have a graph icon in the upper left corner (Fig. 12).

Fig. 12: Newly created graph added to notebook

I1.4. 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. 13.
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.

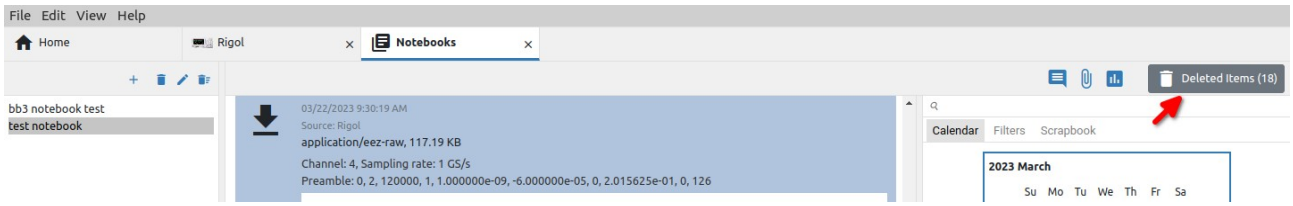


Fig. 13: Deleted items counter

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. 14).

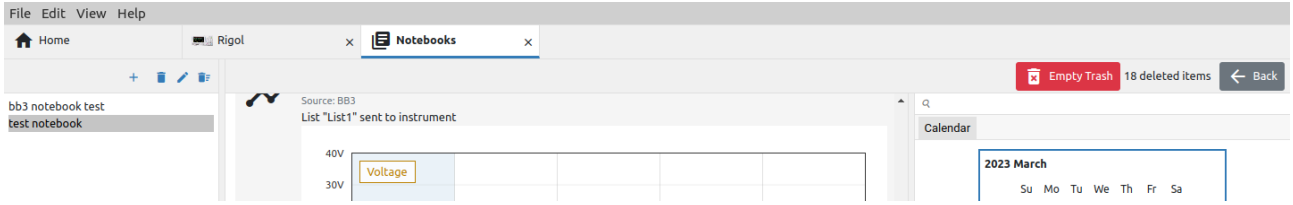


Fig. 14: Empty trash option (no selected items)

If one or more items are selected from the list of deleted items, options for restore (2) or purge (3) will appear (Fig. 15).

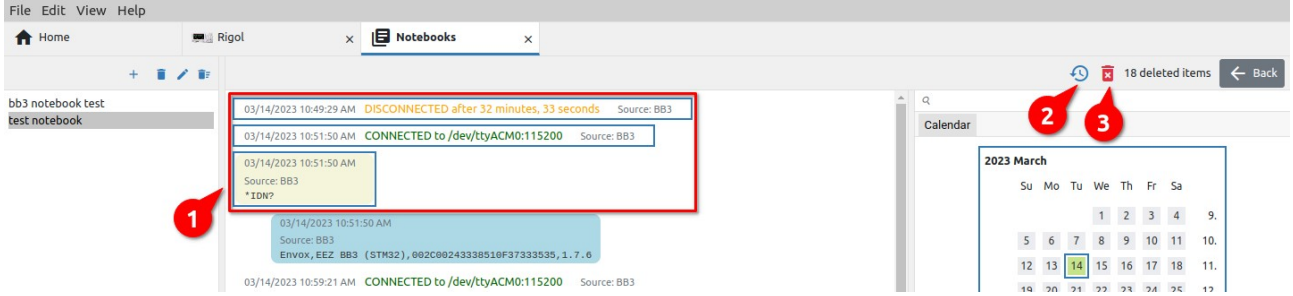


Fig. 15: Selection of deleted items for restore or purge

I1.5. 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?).

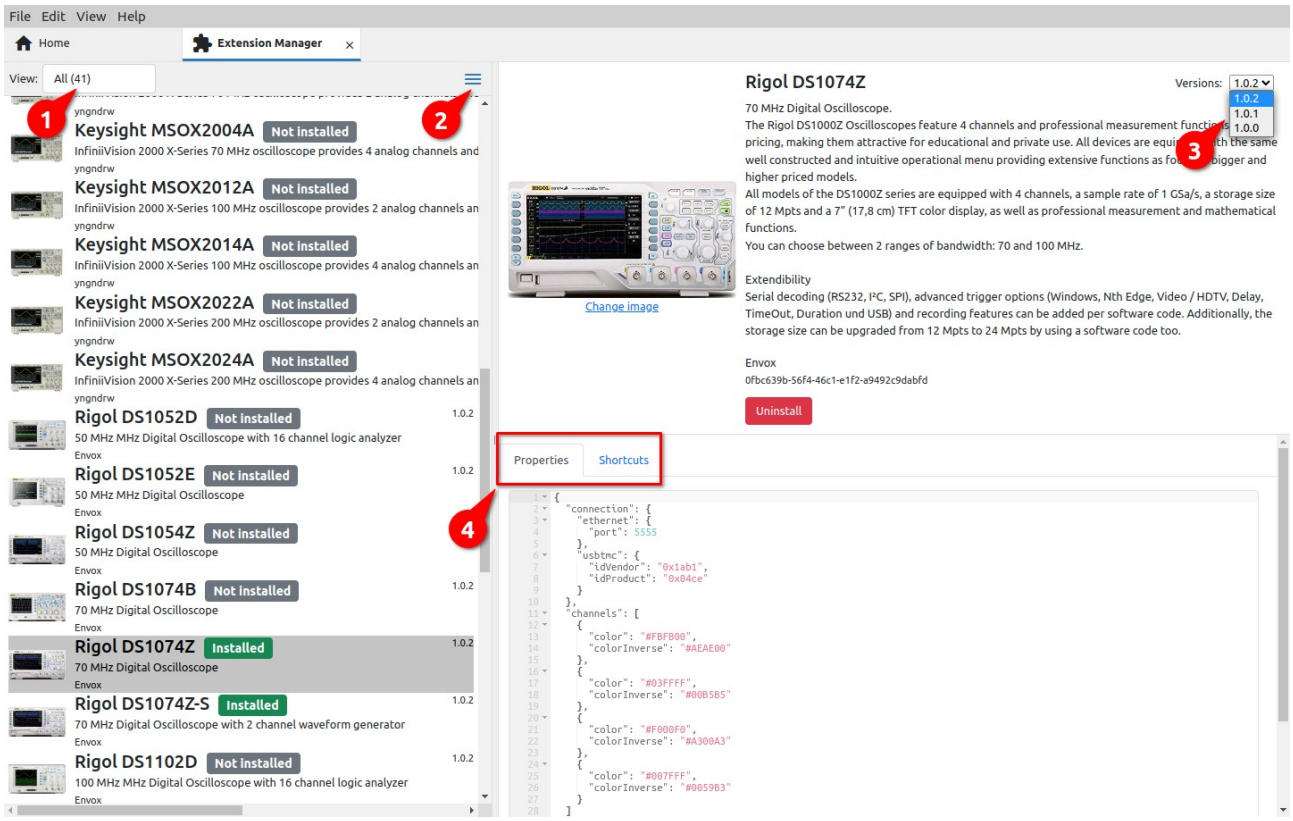


Fig. 16: Instrument extension (IEXT) Manager view

#	Option	Description
1	*View*	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	*Update / Install actions*	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 *Upgrade Catalog* option. The *Install extension* option allows installing an IEXT that is not in the catalog (from local storage).
3	*Versions*	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. 17.

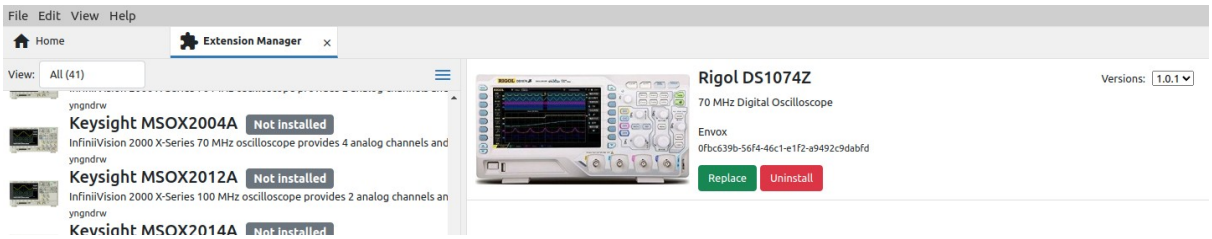


Fig. 17: Changing installed IEXT version

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.

I1.6. Add instrument

By using *Add instrument* (Fig. 1), only those instruments for which there is an IEXT in the IEXT catalog can be added to the workbench.

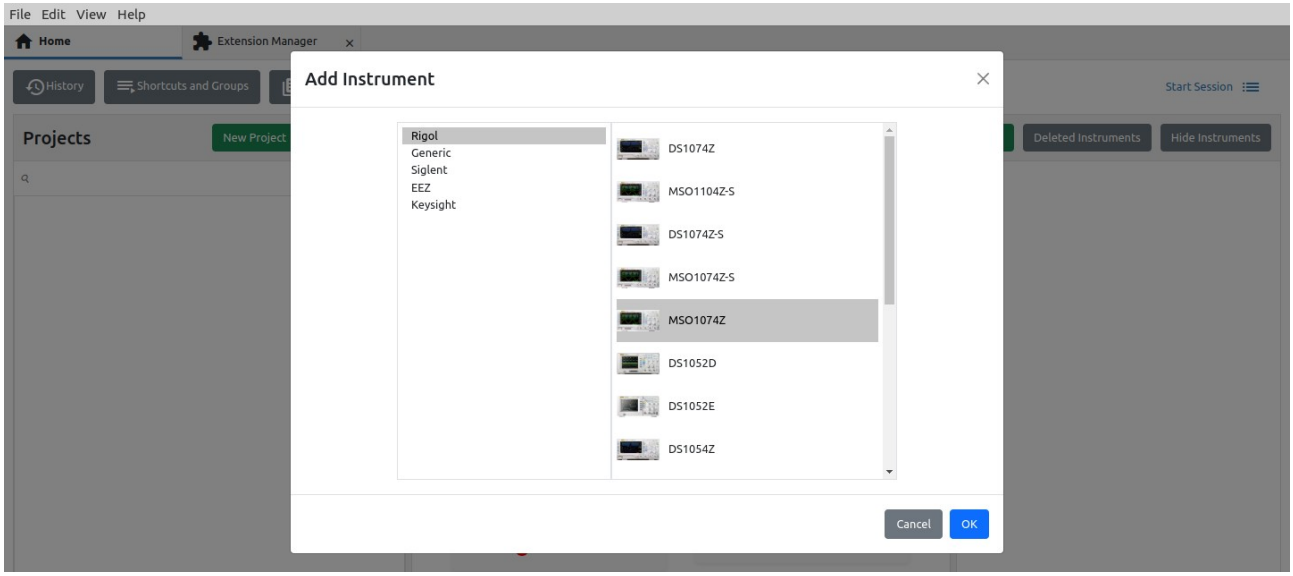


Fig. 18: Add instrument to workbench

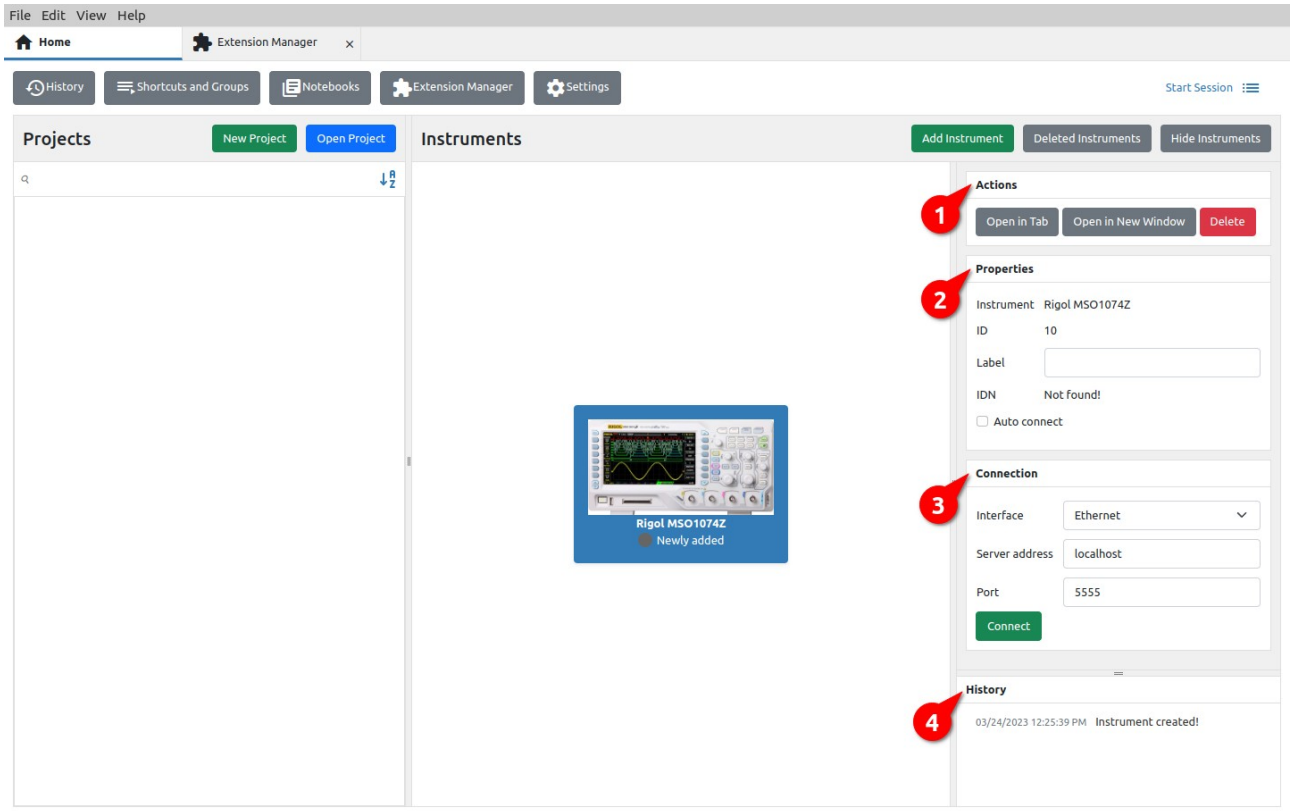


Fig. 19: New added instrument

A successfully added instrument will appear on the workbench (Fig. 19) with the label *Newly added*, and when selected, the sidebar will have the following sections:

#	Option	Description
1	*Actions*	Basic set of actions for displaying the instrument in a separate tab or new window and for removing it from the workbench.
2	*Properties*	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	*Connection*	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. 20.
4	*History*	Preview history of interaction with the instrument using *Terminal*.

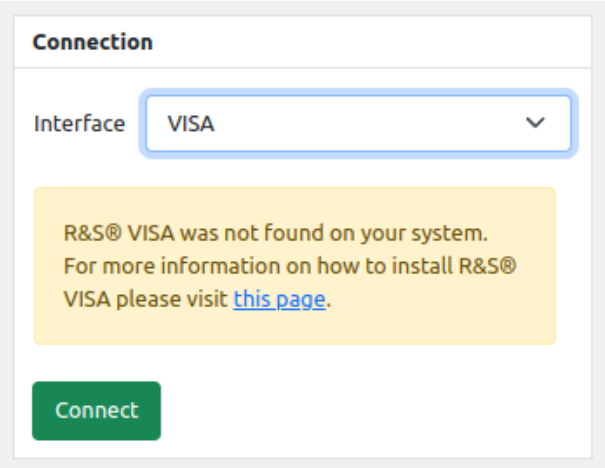


Fig. 20: VISA driver error message

I1.7. Establishing a connection with the instrument

Connection to the instrument added to the workbench will be possible as shown in Fig. 21: 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. 22 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.

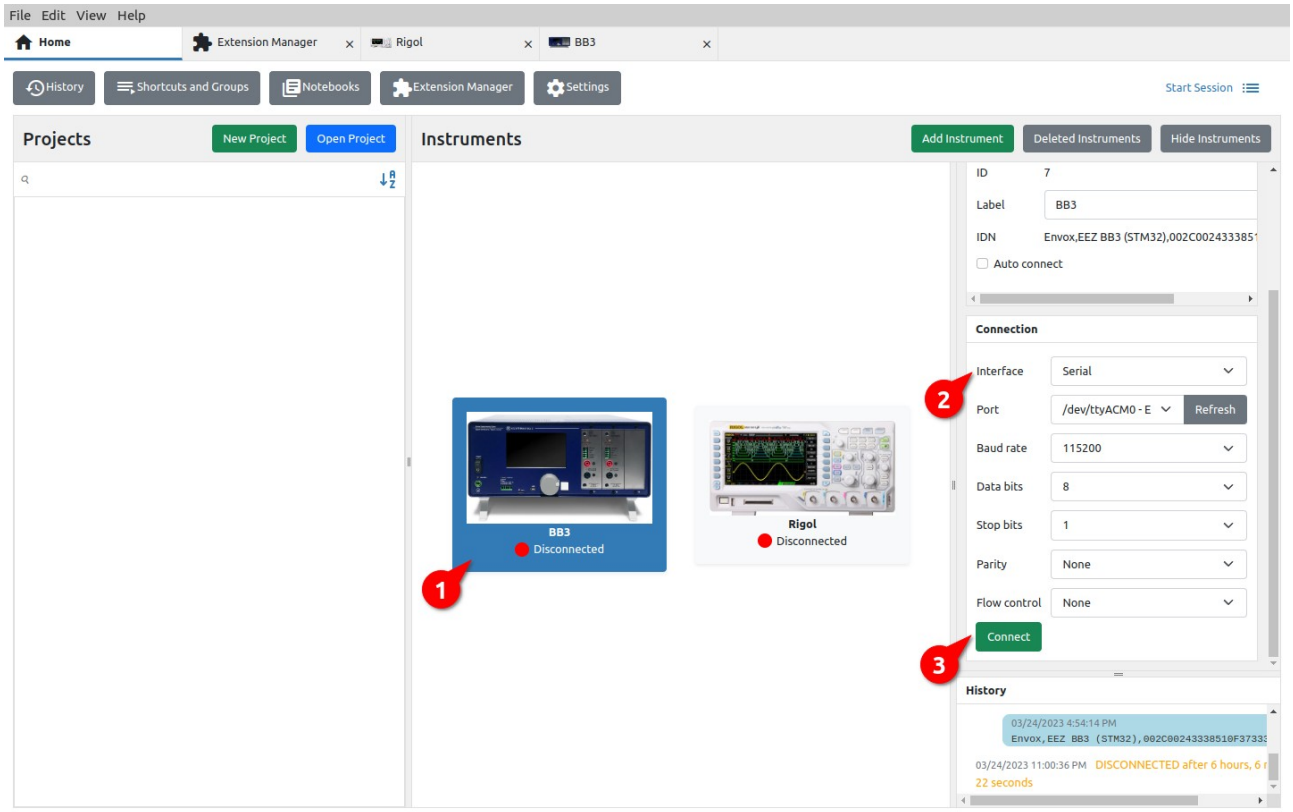


Fig. 21: Selecting an instrument on the workbench to establish a connection

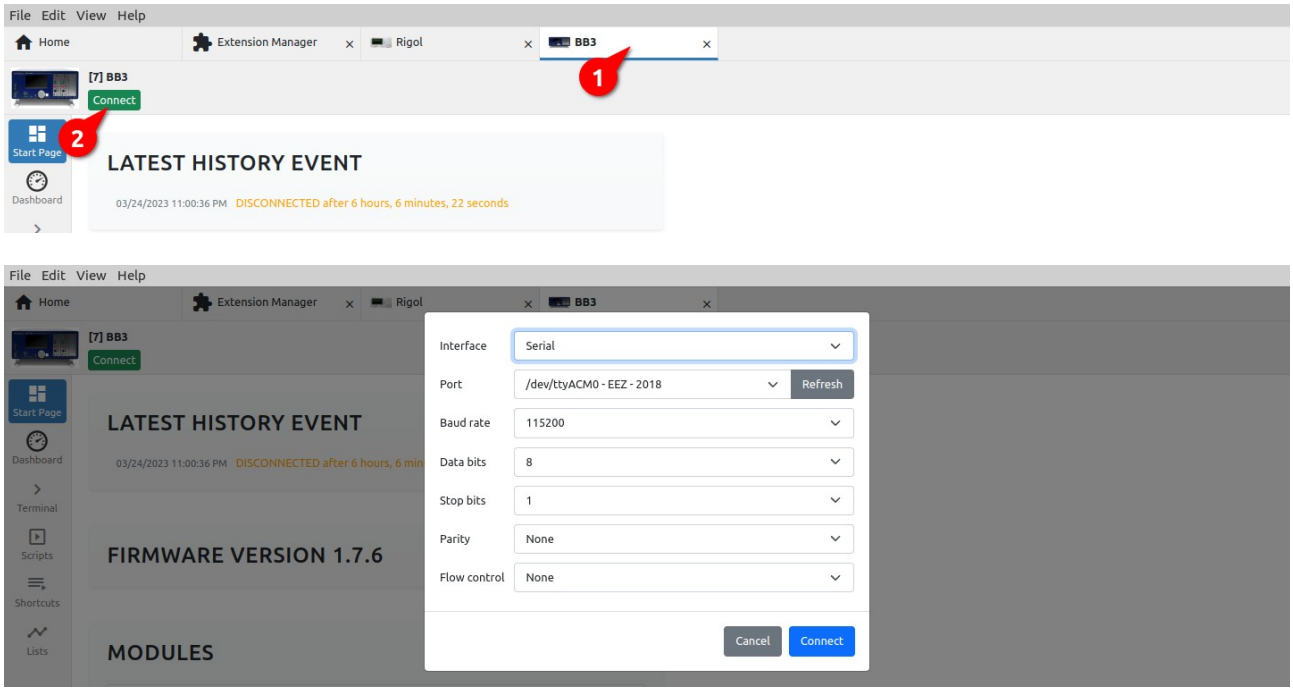


Fig. 22: Establishing a Connection from the Instrument tab

Once the connection is established, it will be possible to close the connection by selecting the Disconnect button (Fig. 23).

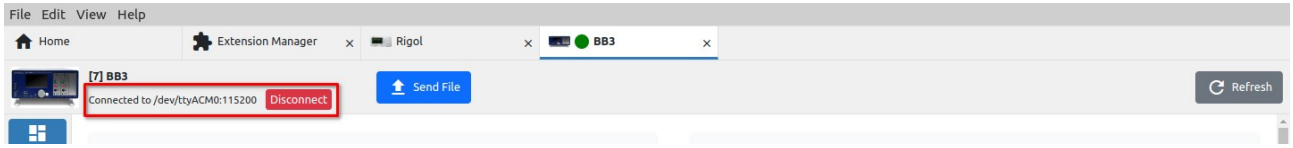


Fig. 23: Option to close the connection

I2. 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.

I2.1. Start page (EEZ BB3 only)

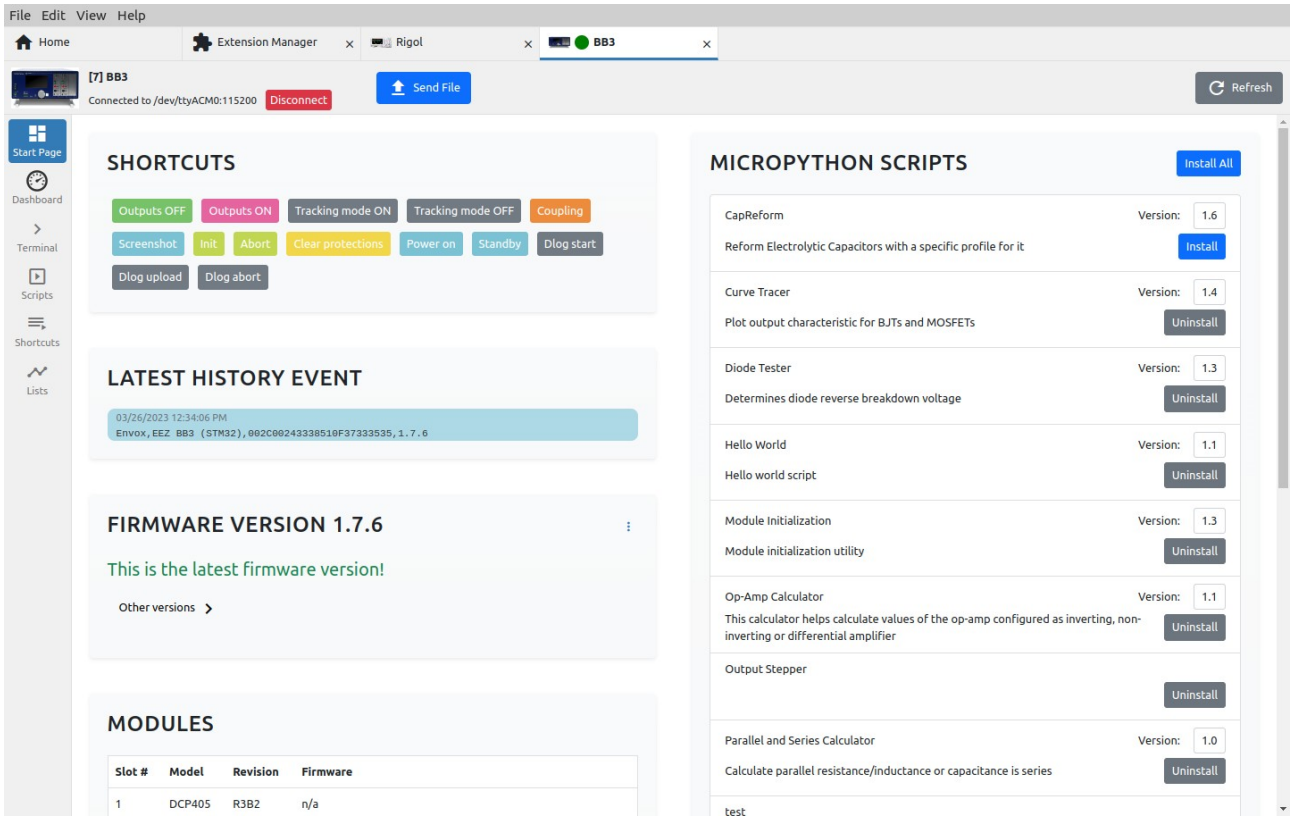


Fig. 24: EEZ BB3 start page

Section / option	**Description**
Send file	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.

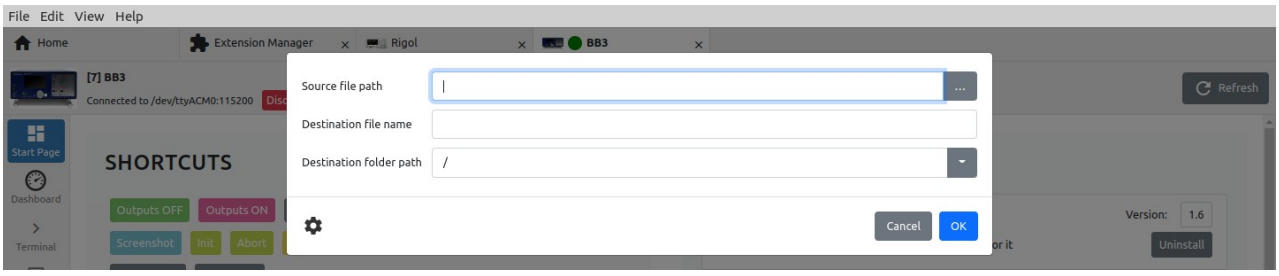


Fig. 25: Sending a file to EEZ BB3

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. 26.

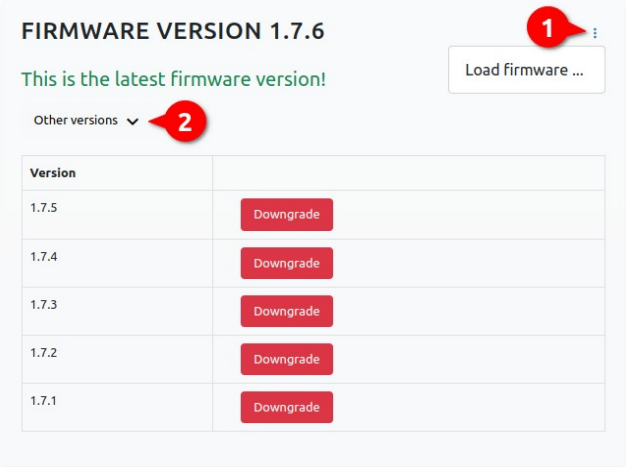


Fig. 26: EEZ BB3 firmware version section

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.

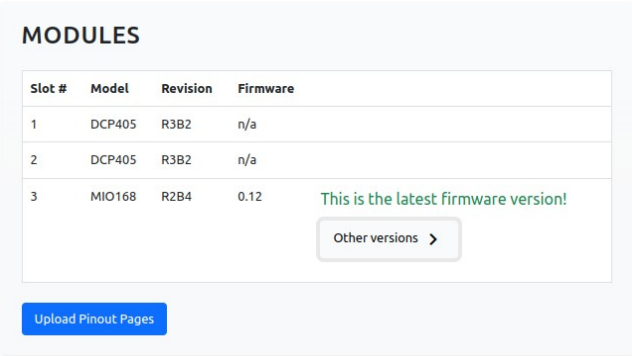


Fig. 27: EEZ BB3 modules section

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 I2.4.). and which are located on EEZ BB3. Lists can be downloaded, uploaded and edited.

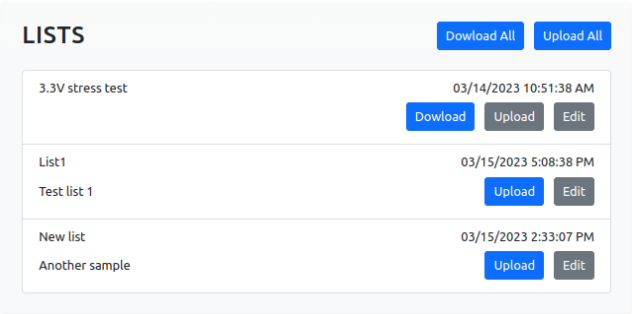


Fig. 28: EEZ BB3 program lists

I2.2. Dashboard

Fig. 29 shows an example Dashboard that enables simple operations with EEZ BB3 modules.

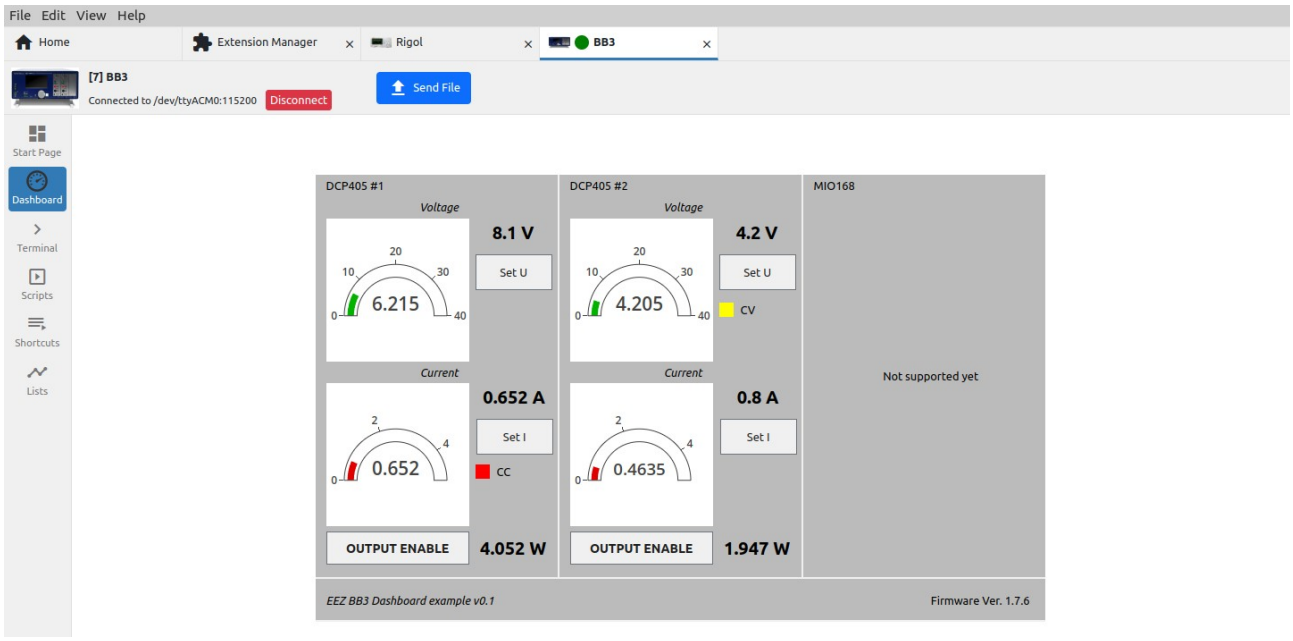


Fig. 29: Instrument dashboard example

I2.3. 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).

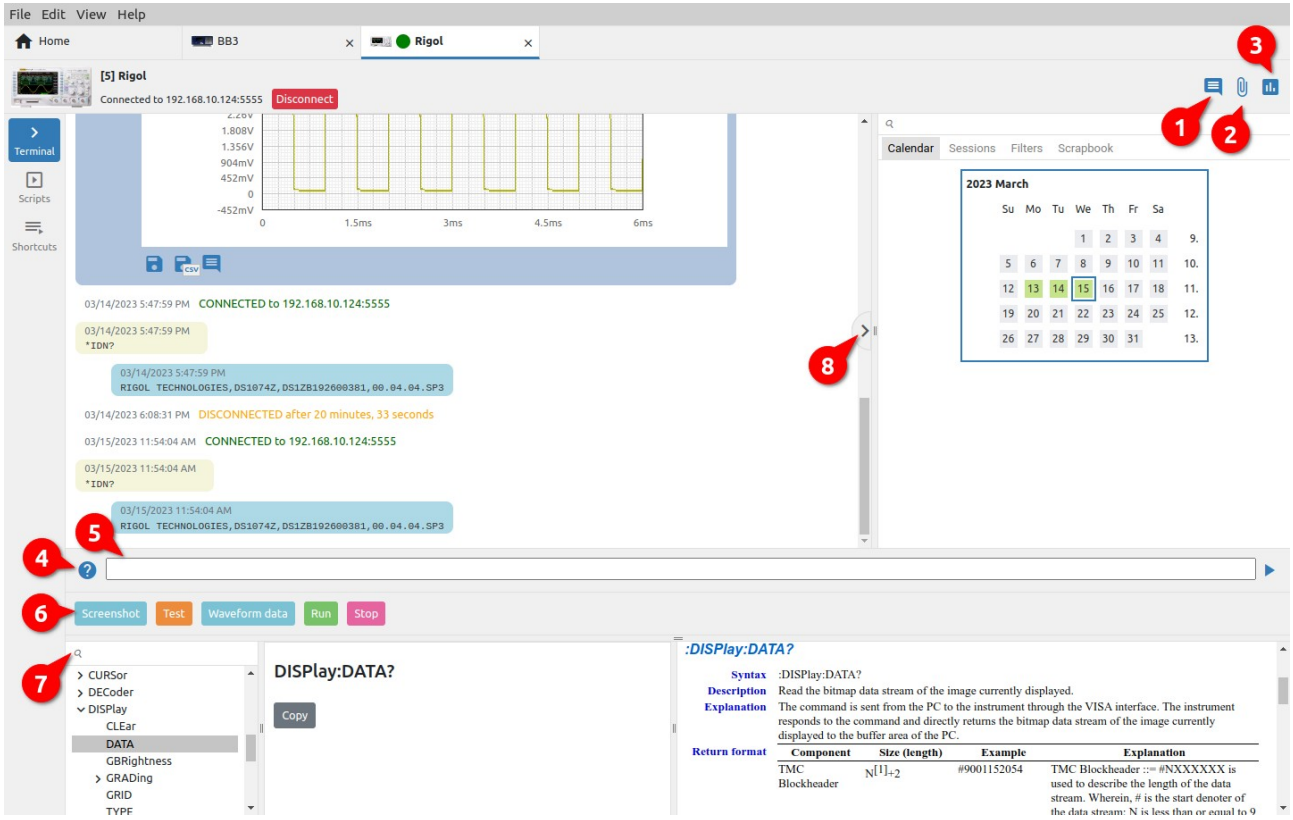


Fig. 30: Instrument terminal

#	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 *Terminal* 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.

I2.4. 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.).

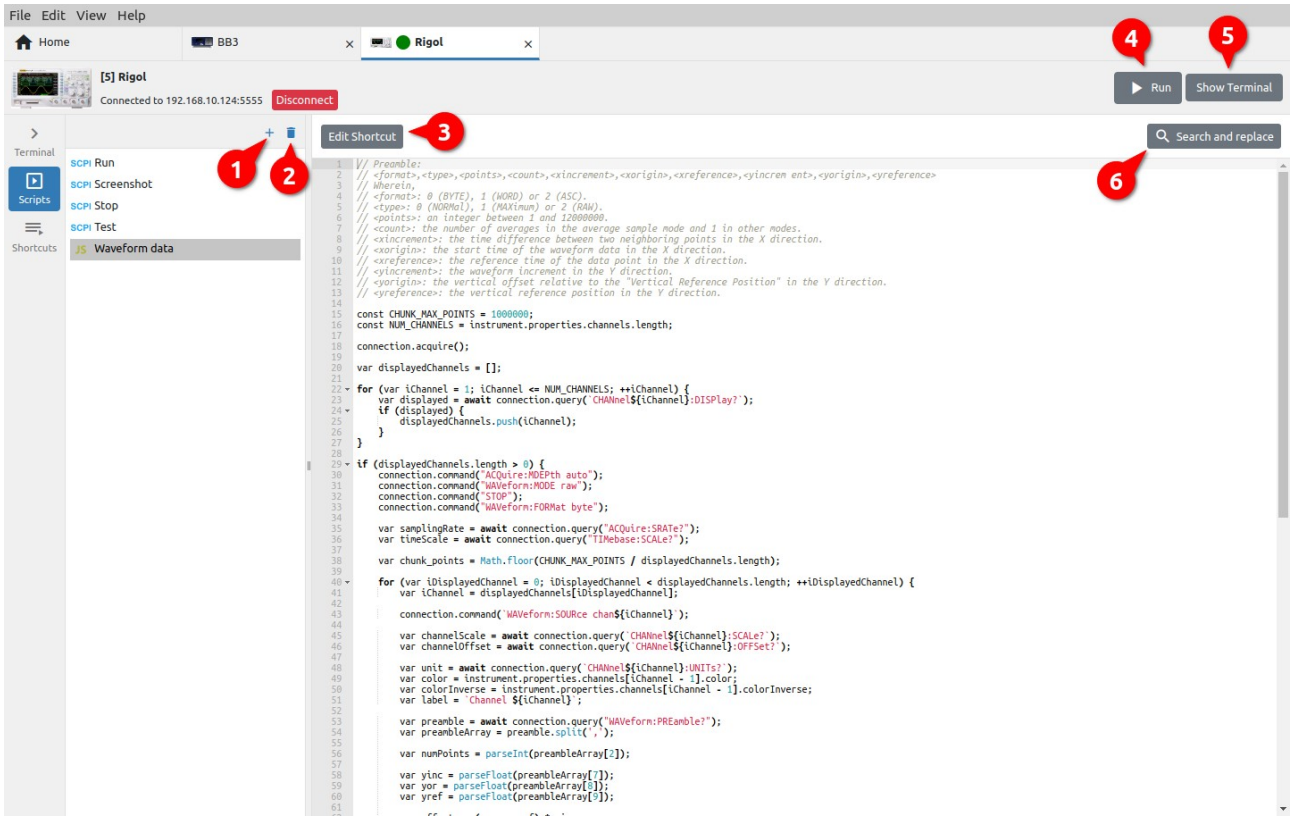


Fig. 31: Instrument scripts

#	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).

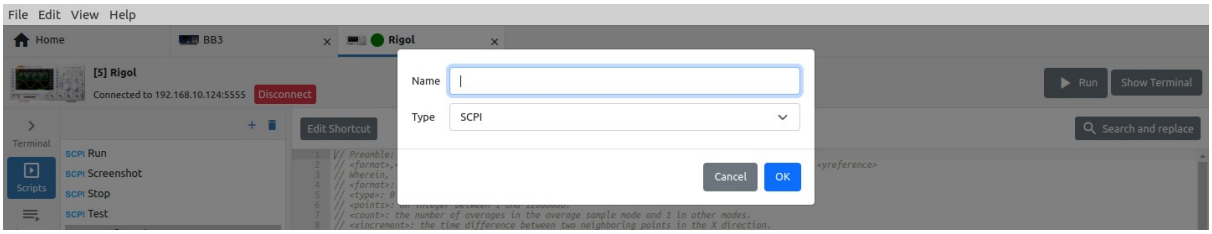


Fig. 32: Adding a new script

The content of the script is entered in the editor (Fig. 33).

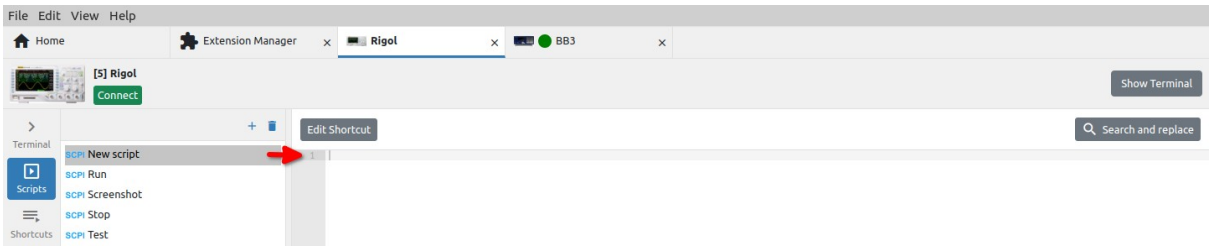


Fig. 33: Script editing

2	*Delete script*	Deleting the selected script.
3	*Edit shortcut*	Editing a script shortcut (see Section I2.4.1.)
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.

I2.4.1. Edit script shortcut

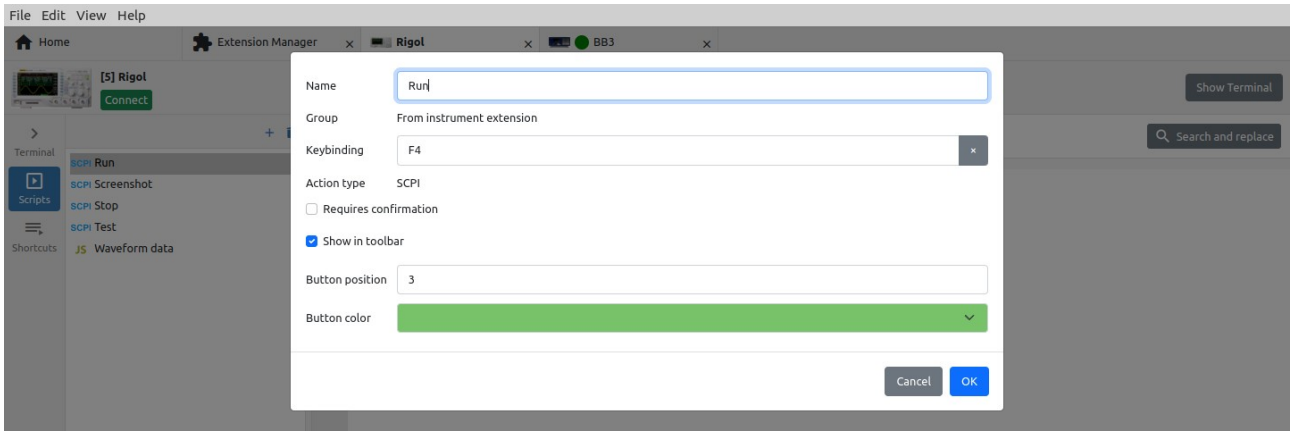


Fig. 34: Script shortcut editing

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 *From instrument extension* will be displayed.

Keybinding	A key or a combination of several keys (e.g. with SHIFT, ALT, CTRL) that will start the execution of the script.
Action type	Script type: SCPI, JS or MicroPython (EEZ BB3 only).
Requires confirmation	Displays a dialog box to confirm the execution of the script.
Show in Shortcuts bar	Determines whether the shortcut button will be displayed in the *Terminal*'s *Shortcuts bar*.
Button position	The position of the shortcut button in the *Shortcuts bar*. 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.
Button color	Color coding of shortcut button.

I2.5. Shortcuts

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

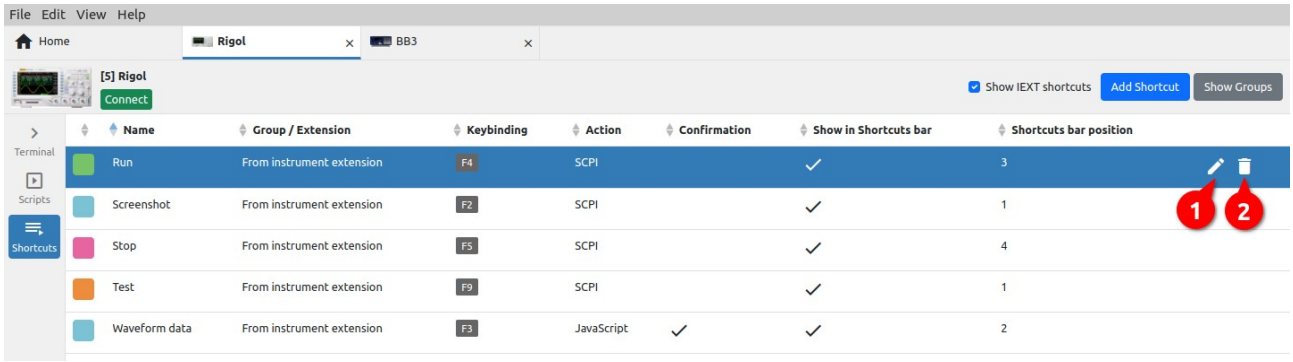


Fig. 35: Instrument shortcuts

#	Option	Description
1	*Edit shortcut*	Editing the shortcut (see Section I2.4.1.)
2	*Delete shortcut*	Deleting an existing shortcut.
	Show IEXT shortcuts	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. 36.

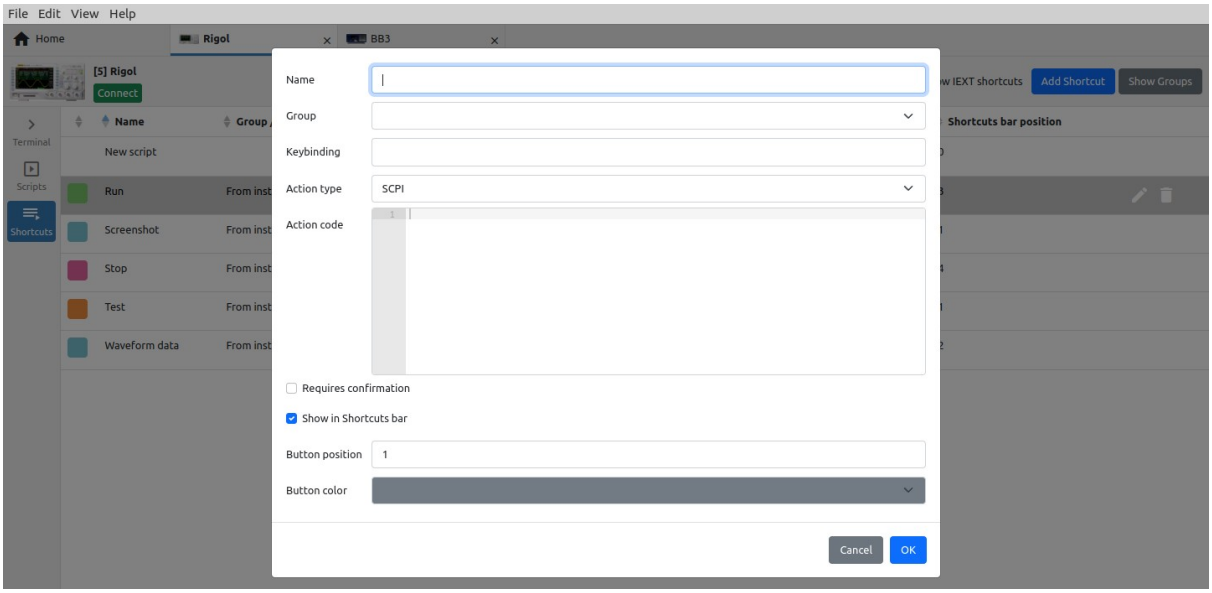


Fig. 36: Add new shortcut

Show Groups / Show Shortcuts	Toggle between displaying a list of shortcuts and groups (Fig. 37) of shortcuts.

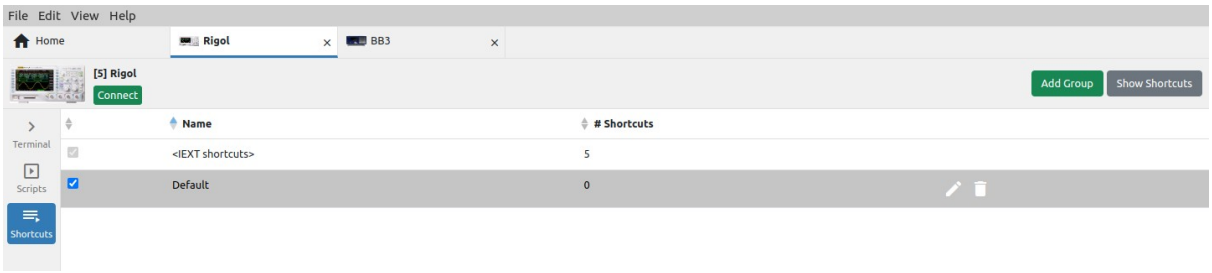


Fig. 37: Instrument shortcut groups

I2.6. 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.

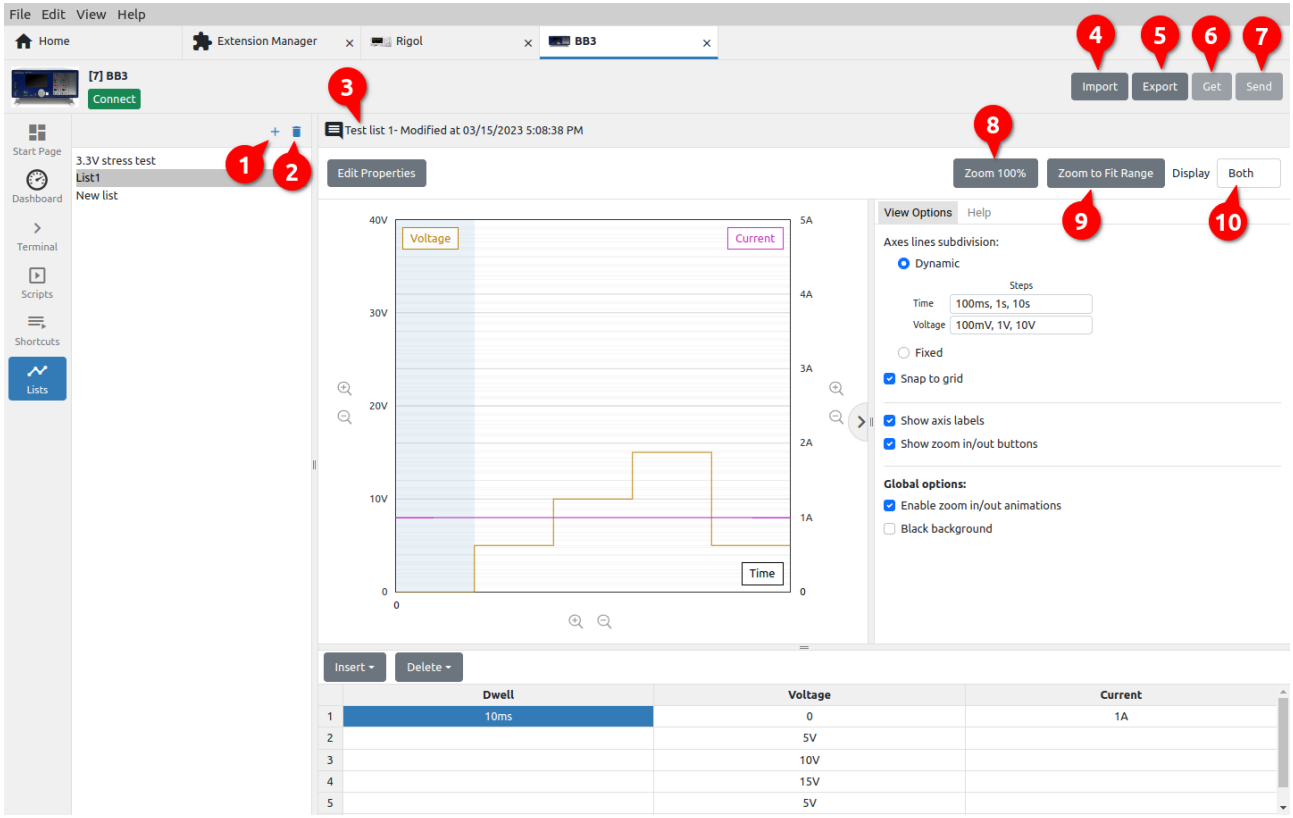


Fig. 38: Instrument programming lists

#	Option	Description
1	*Add list*	Creating a new list. The parameters of the list can be specified through a table (Fig. 38) 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. 39) 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. 40).

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

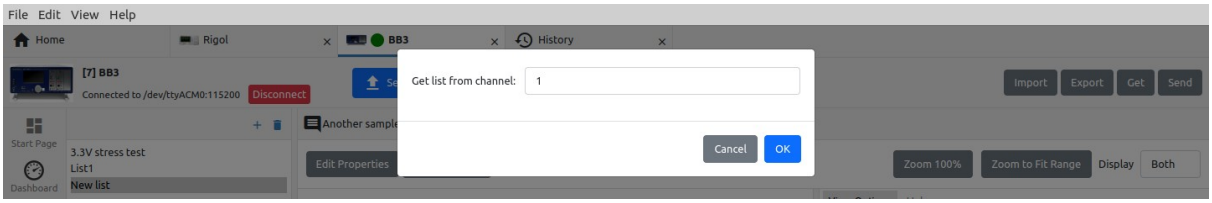


Fig. 39: List source selection

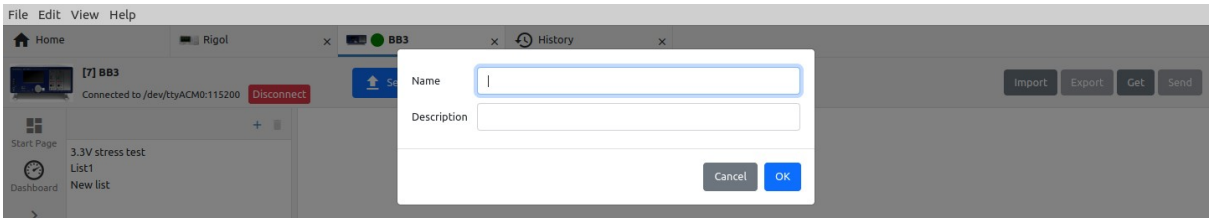


Fig. 40: Imported list parameters

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).

I2.6.1. Editing a list using a table

Editing the list via the table is shown in Fig. 38. 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. 41 and Fig. 42 shows all options for inserting new lines and deleting existing ones.

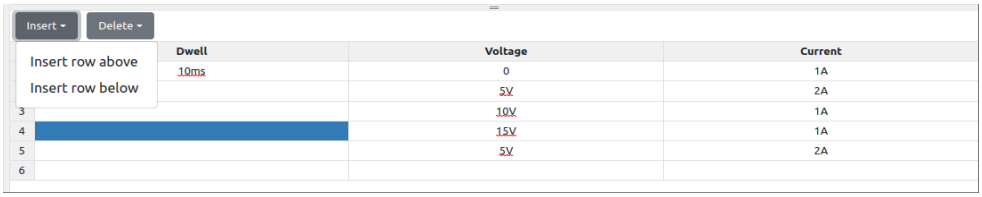


Fig. 41: Table insertion options

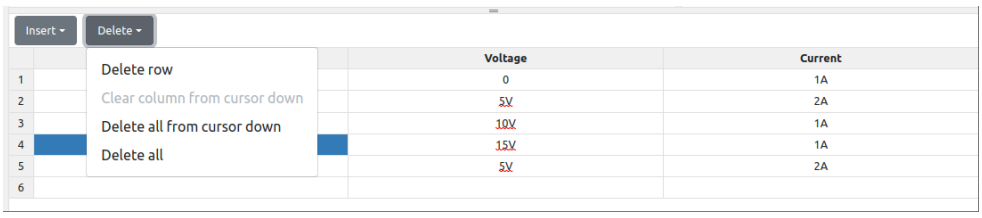


Fig. 42: Table deletion options

I2.6.2. 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.

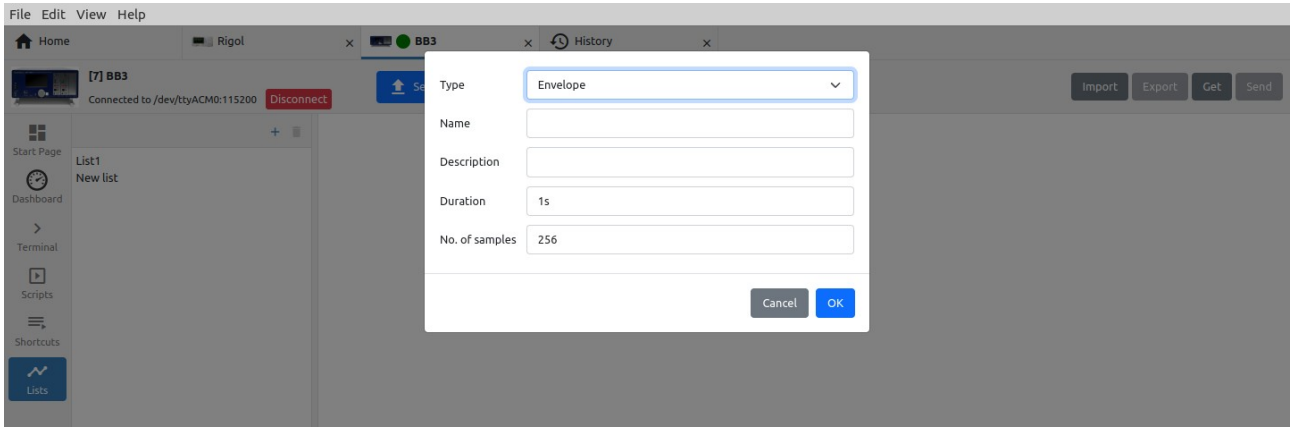


Fig. 43: Adding a new list in envelope mode

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. 43). 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.

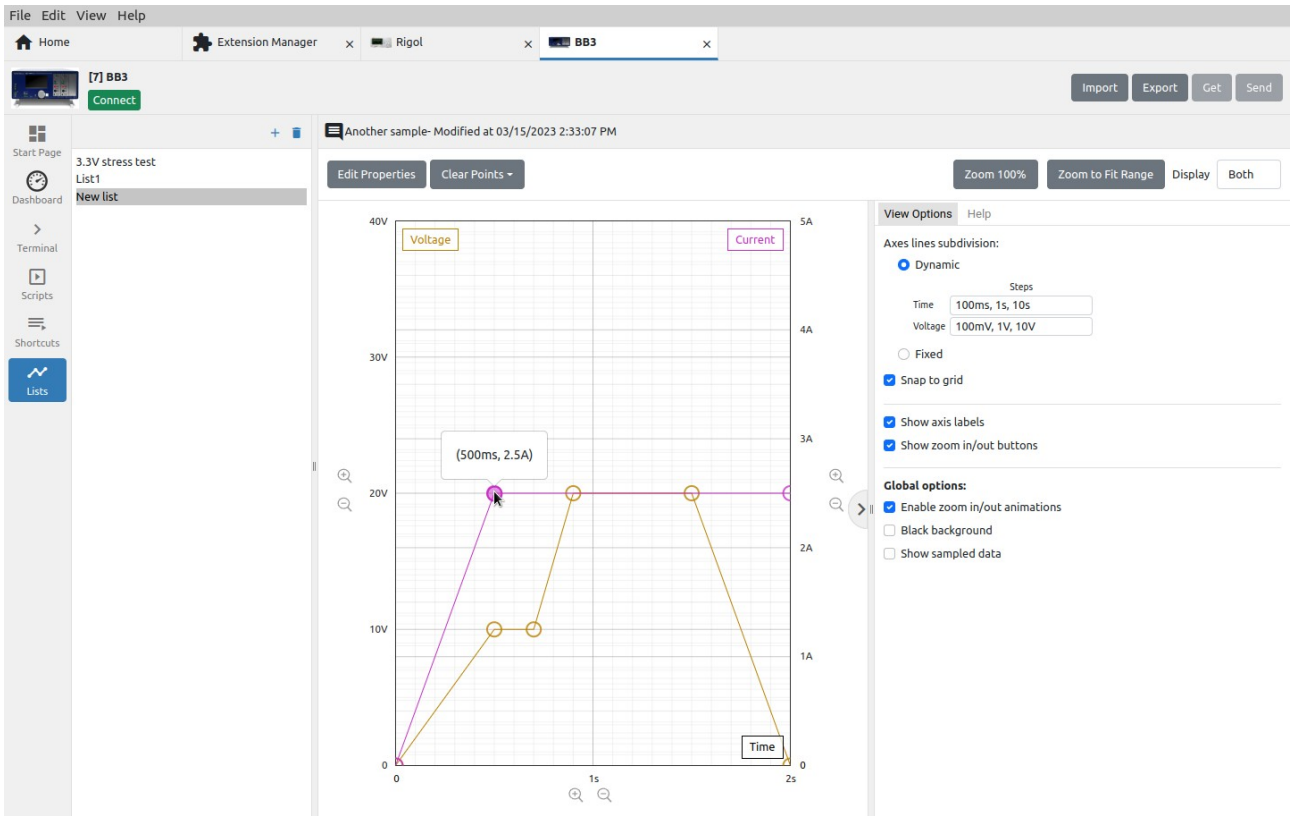


Fig. 44: Graph editing in envelope mode

The example in Fig. 44 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.

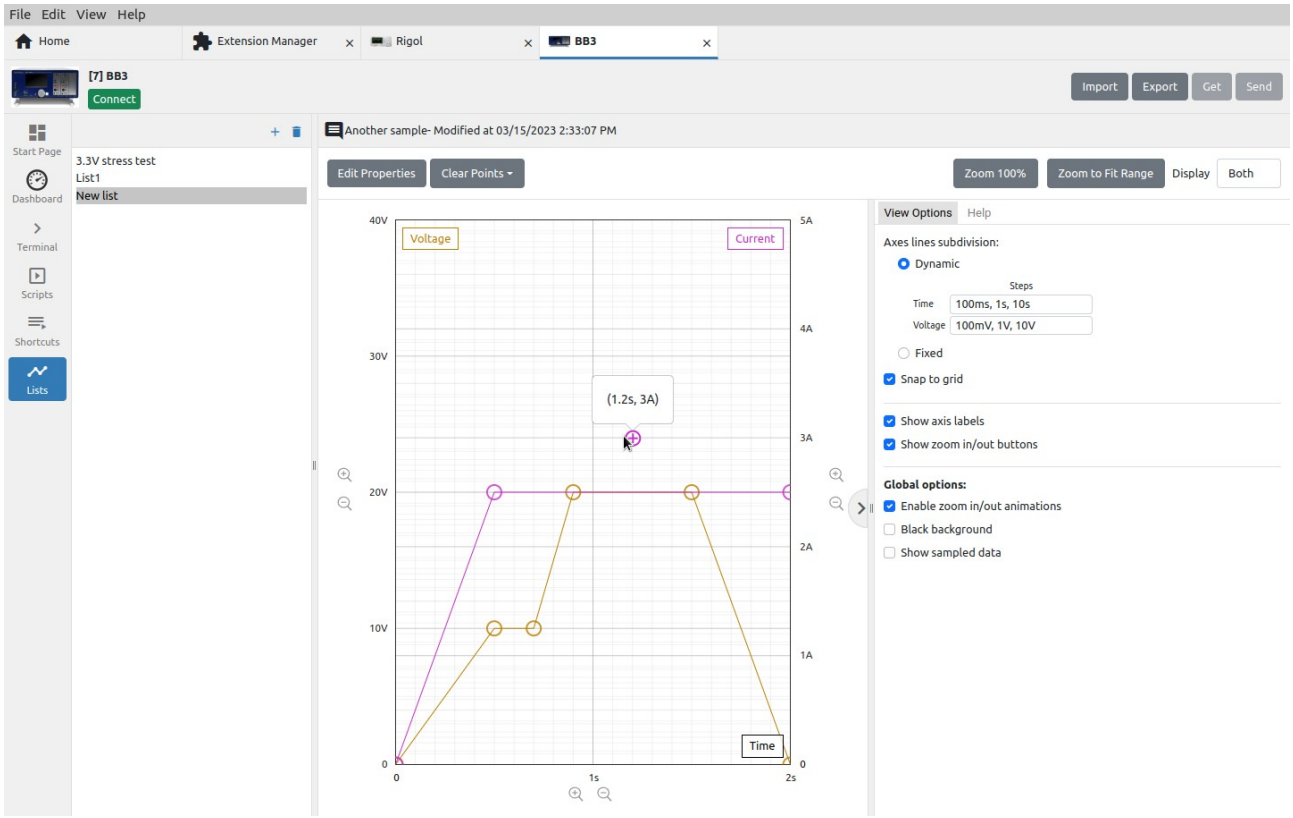


Fig. 45: Adding a new point in envelope mode

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. 46.

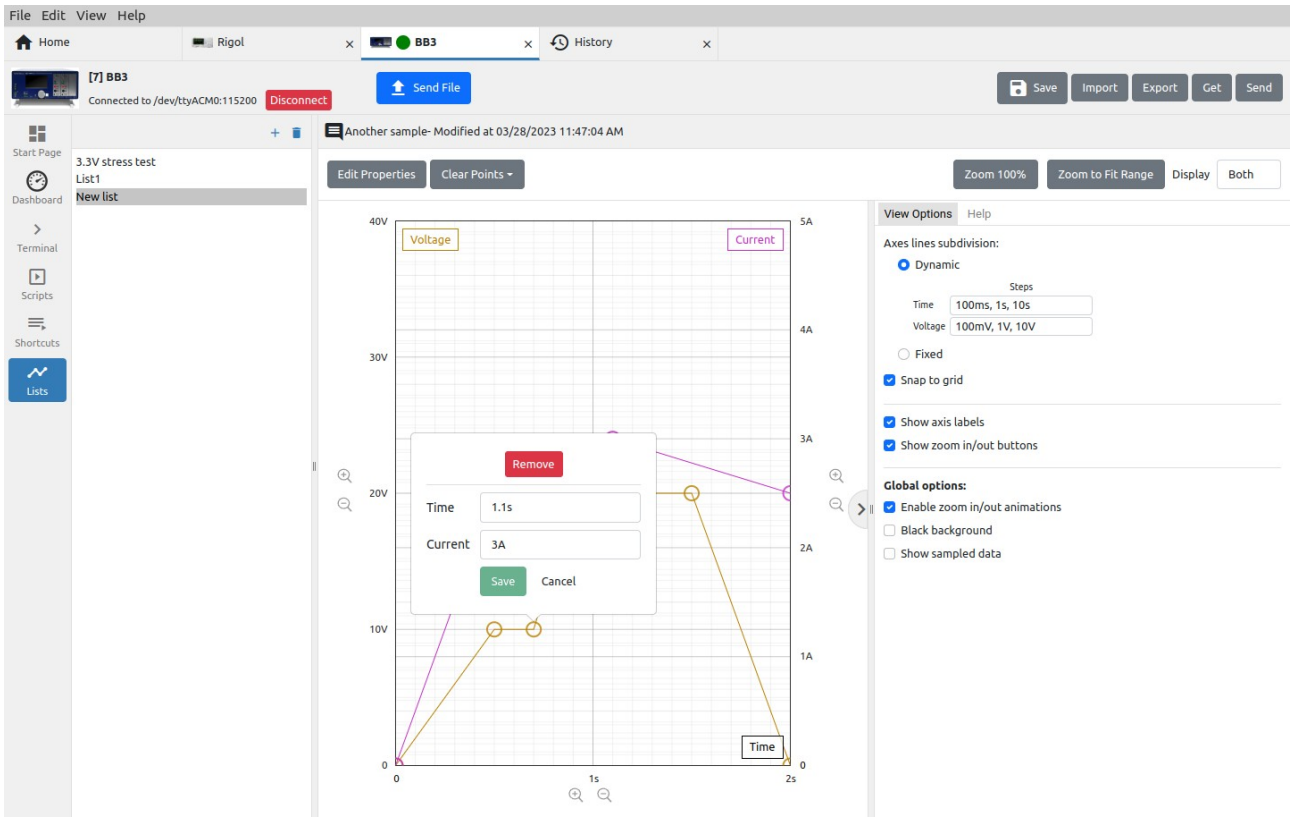


Fig. 46: Program point editing in envelope mode

I2.6.3. List view options

The display of the graph can be dynamically changed (Fig. 38)depending on the resize of the window or the number of graticules can be fixed (Fig. 47).

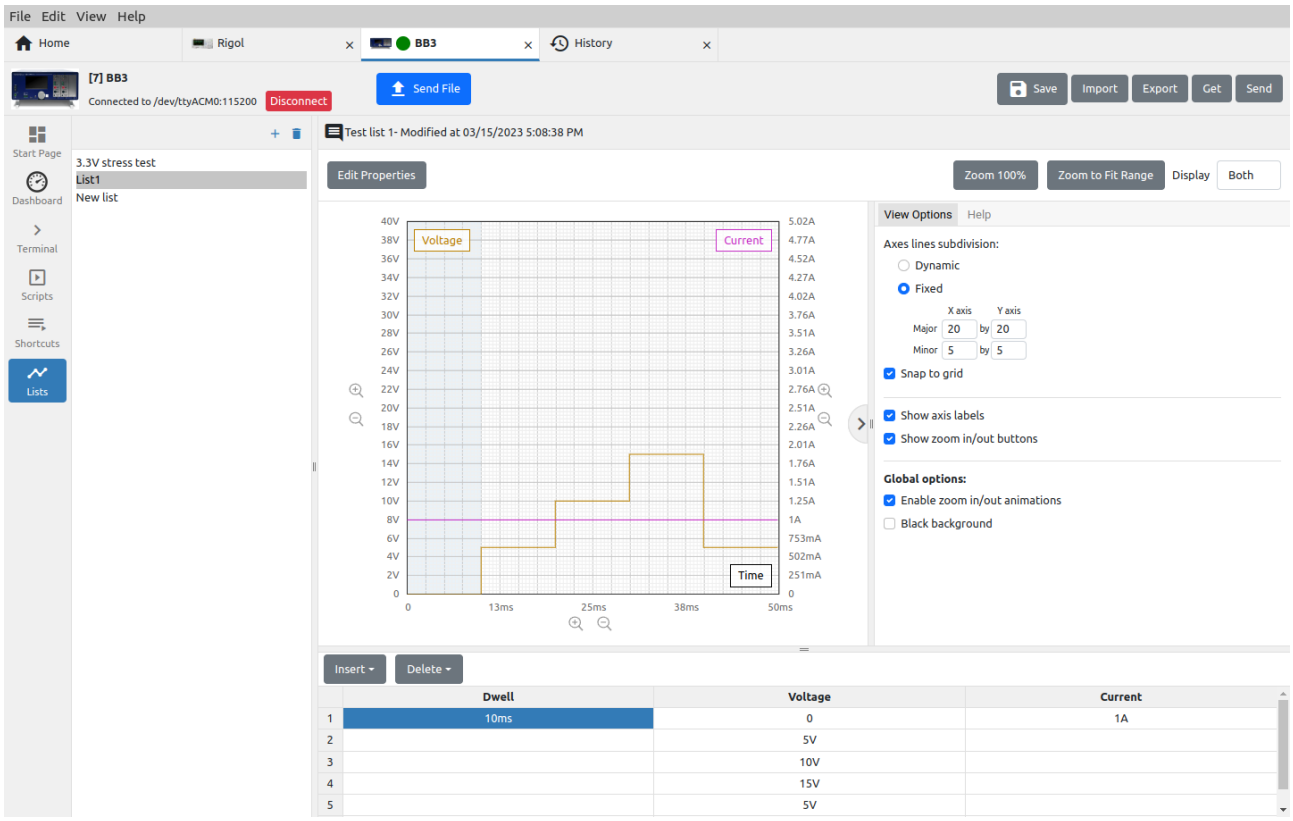


Fig. 47: Fixed graph view

I2.6.4. 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. 48.

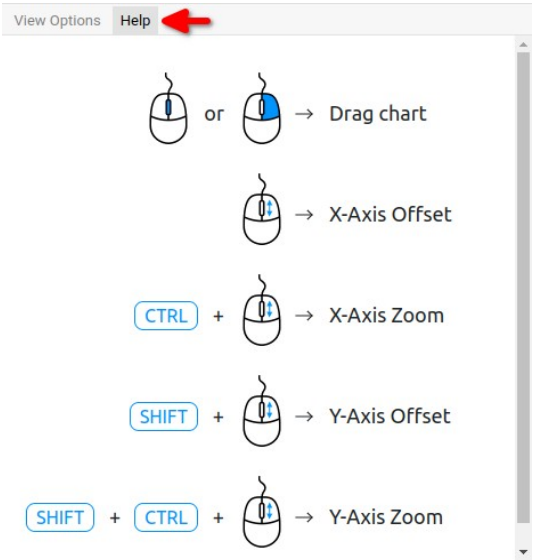


Fig. 48: Graph navigation and zoom help

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

Version: M15 DRAFT
Date: 2023-06-16